SentiSight 2.1 SDK

Developer's guide

Developer's guide version: 2.1.0.3

Publish date: 4/28/2011

Table of Contents

1 Introduction

It is a natural ability for human to recognize objects. Every day we face numerous objects we have seen before and we can identify them in the same or different environment without even thinking about it. There exist many tasks where object recognition process must be automated and the number of such tasks grows constantly. SentiSight library is intended for the developers who want to use computer vision based object recognition in their applications. Thus, its main features are object learning and recognition. The first part, object learning, is dedicated to extraction of object model from a sequence of images containing the object. The second part, recognition, stands for object recognition in a test image.

1.1 Funcionality

SentiSight SDK is intended for vision based object recognition. Generally, it enables learning of object model and recognizing the object in a test frame. Learning of model object means extraction of single model from a set of images containing the object. Recognition (see page 8) part compares a test image with the model and answers the question, if this image contains the object or not.

These functions will be further explained in Section SentiSight API (see page 8) and demonstrated in Chapter Tutorials using demonstration applications.

SentiSight SDK requires object images to meet certain constraints to ensure optimal recognition performance.

SentiSight SDK incorporates auxiliary libraries:

- NCore (see page 24)

- NImages (see page 98)

- NVideo (see page 161)

- NDeviceManager (see page 85)

that provides infrastructure and functionality for working with images. .NET wrapper of these libraries are:

- Neurotec (see page 219)

- Neurotec.Images (see page 241)

- Neurotec.Video (see page 317)

- Neurotec.DeviceManager (see page 305)

1.2 Before You Begin

Before using SentiSight SDK components you must activate your license (For details please review `Activation.pdf` file).

1.3 System Requirements

Minimum requirements for system:

- PC with 1.4 GHz processor supporting SSE2 technology
- 256 MB of RAM
- Microsoft Windows 2000/2003/XP/Vista/7 or Linux (based on glibc 2.3.4 or compatible) operating system
- Microsoft DirectX 9.0 or later
- .NET framework 2.0
- Optionally, video capture device (web camera)

1.4 Licensing

To develop a product based on SentiSight 2.1 SDK technology, an integrator should obtain a license. Integrators can develop only an end-user product using SDK and sell/install the product to their own customers. For more information please review 'licensing model section of www.neurotechnology.com website.

A license is required for each running instance of SDK components. The following license types are available:

- Single computer license (see page 2)
- Enterprise license (see page 3)

SentiSight 2.1 SDK includes:

- 1 installation license

SentiSight 2.1 SDK customers can also obtain additional licenses for their product installation or development at any time.

Please also refer to SentiSight 2.1 SDK Software License Agreement (Documentation\license.html) for all licensing terms and conditions.

1.4.1 Single Computer License

A single computer license allows to install and run a SentiSight 2.1 SDK components on one computer processor core. Component license will not be lost if computer will be reinstalled.

The following license management options are available:

- license activation online by communicating with Neurotechnology's server
- license activation by email
- license activation using volume license manager (see page 3)
- license management using volume license manager (see page 3) on LAN or Internet

1.4.2 Volume License Manager

Volume license manager is used on site by integrators or end users to manage obtained licenses for SentiSight 2.1 SDK components. It consists of license management software and a dongle, which is used to store the number of obtained licenses. An integrator or an end-user can use the volume license manager in the following ways:

- **Activating the single computer licenses**. An installation license for a SentiSight 2.1 SDK component will be activated for using on a particular computer. The license quantity for the SentiSight 2.1 SDK component in the license manager will be decreased by the amount of activated licenses.
- **Managing the single computer licenses on LAN or Internet**. The license manager allows to manage installation licenses for SDK components across the computers on LAN or Internet. The number of managed licenses for a SentiSight 2.1 SDK component is limited by the number of licenses in the license manager. No license activation is needed and the license quantity is not decreased. Once issued, the license is assigned to certain computer on the network.
- **Using a license manager as a dongle**. The volume license manager containing at least one license for a SentiSight 2.1 SDK component can be used as a dongle that allows to run SentiSight 2.1 SDK component installation on a particular computer.

Additional SentiSight 2.1 SDK component installation licenses for the license manager can be purchased anytime. Neurotechnology will generate a special update file and send it to you. Then you should enter file to the license manager to add purchased licenses.

1.4.3 Enterprise Licensing

SentiSight 2.1 SDK enterprise license allows an **unlimited use** of SentiSight 2.1 SDK components (Extractor and Matcher) in the end-user products within the certain territory, market segment or project. These limitations would be included in the licensing agreement.

The enterprise license price depends on the application size and the number of potential application's users within the designated territory, market segment or project. SentiSight 2.1 SDK enterprise licenses are provided only for big projects.

2 What's new

Version 2.1.0.3

1. Added support for Axis M1114 (see page 354) network camera.

Version 2.1.0.2

1. WSQ and IHead multi-thread issues were fixed (see NImages (see page 98) Library Change log (see page 342)).
2. Reading frames from multiple files issue fixed (see NVideo (see page 161) Library Change log (see page 345)).

Version 2.1.0.1

1. Added support for Pixord N606 (see page 21) network camera.

Version 2.1.0.0

1. Renewed tutorials for all programming languages
2. Added new images comparison tutorial
3. Added new SentiSight sample for C# programming language
4. Added new wxWidget light sample
5. SentiSight uses NDeviceManager (see page 85) (Neurotec.DeviceManager (see page 305)) library instead of CameraMan
6. Added C++ wrapper

Version 2.0.0.2

1. Added Points property to the SEShape class in .NET Reference
2. Added new C# tutorial for image comparison

Version 2.0.0.0

1. Algorithm changes (see page 4)
2. Interface changes (see page 5)
3. New algorithm demo changes (see page 5)

2.1 Algorithm Changes

The following changes were made to SentiSight algorithm:

1. Greatly improved recognition quality.
2. Added finding and counting of the number of the same object instances in a scene.
3. Approximate estimation of the region an object occupies in a scene.
4. Ability to compare two pictures and find perspective transformation between them.

Principle description of some algorithm changes

Adding of Shapes

Now it is possible to pass not only mask to determinate which part of the object to add to the model but also a new geometric feature - user defined shape.

In SentiSight 2.0 shape is represented as a collection of clockwise or counterclockwise arranged points. Shape cannot be self intersecting, and is closed which means the last point in a shape conceptually connects to the first. Thus, before using shape user must check if closing edge is not intersecting all others and shape is closed. Not closed and empty shapes (not valid shapes) are not accepted by learning functions.

Neither mask nor shape does not represent actual border of the object by itself. Such scenario is completely up to a user. User can mark actual border of the object by these parameters, can mark different parts of the object by mask and shape or can mark in completely free form depending on usage scenario of the system. Only thing bearing in mind is that information marked by mask should be usable for recognition, because from this region information is added to the model.

Shape is not used in the learning process it is used in the recognition stage to return the region defined by shape if object representing by the model containing this shape is recognized. If it is possible, the system automatically perceptively aligns shape to the input image, if not, shape is aligned by similarity transform (rotation, translation, and scale).

2.2 Interface Changes

The following changes for SentiSight interface were made:

1. Low/High profile learning. Different choices between speed and recognition quality.
2. High/Low speed recognition:
    1. "High Speed", - as fast as SentiSight 1.1., cannot find all instances of an object (finds only one instance).
    2. "Low speed", - slower, but returns all detected instances of an object, better recognition quality.
3. All interface functions are revisited.
4. For Linux SDK reading from video files is added.

2.3 New Algorithm Demo Application

The new Algorithm demo application:

1. Completely new demo application with new convenient user interface.
2. Off line working mode in Learning and Recognition (see page 11) stages.

3 Migration Guide

Migrating from version 2.0 to version 2.1

Since SentiSight version 2.1 some changes to SentiSight library interface were made. Please, refer to SentiSight library change logs (sections SentiSight Library (see page 345) and Neurotec.SentiSight (see page 352)) to see detailed information. In order to use SentiSight 2.1 you should update your code and recompile it.

You should note that some interface changes were made to NImages (see page 98) and NCore (see page 24) libraries too. Read more about changes in these libraries in Change log (see page 339) section.

4 Overview

4.1 Main SentiSight functions

SentiSight API is intended for vision based object recognition. Its main features are object learning and object recognition. The overall usage and functionality of this library is documented in Section SentiSight API (see page 8). Reference could be found in Section SentiSight Library (see page 168). The constraints relating the object are discussed in Section Constraints (see page 14).

4.1.1 Object Learning

In order to recognize an object in an image, the appearance of the object should be memorized. The process of memorizing an appearance of the object from images with various poses is called object learning. A set of images containing the object should be provided to the algorithm and the algorithm extracts so called model - a symbolic representation of the object. It is highly recommended to provide information about exact location of the object in the image. This can be done by the shapes of the object. Shapes explicitly specifies the object. Thus, only object specific information will be included into model template. The quality of object recognition highly depends on model created by object learning part. Thus, a set of images of the object should contain all possible poses of the object - the three dimensional rotations (off plane rotations) are highly recommended. Also, it is recommended that images of the object would be taken under different light conditions or using different light sources in order to improve invariance to diverse light conditions. It is recommended to use shapes.

Main SentiSight library functions are these:

- SECreate (see page 172) - allocates memory for a new model
- SELrnAddToModel (see page 174) - adds a new image to the existing model
- SELrnGeneralizeModelEx (see page 176) - compresses the model

.NET methods:

- CreateModel (see page 291) - creates a new model associated to the SEEngine object
- AddToModel (see page 275) - adds a new image to the model
- GeneralizeModel (see page 276) - compresses contents of the model

4.1.1.1 Object Learning using Foreground/Background Separator

Special functions intended to separate foreground (the object) from background in an image.

Main functions:

- SESepAccumulateBackground (see page 196) - accumulates images to form background
- SESepSeparate (see page 202) - separates foreground (the object) from background

.NET methods:

- AccumulateBackground (see page 286) - adds an image to form background

- Separate (see page 289) - separates foreground (the object) from background

4.1.2 Object Recognition

Recognition (see page 11) is a process of identification whether an image contains an object or not. Recognition (see page 11) compares models which were learnt by object learning part, with current test image or test model, and returns a comparative score (score or similarity). A high similarity score suggests that the test image contains one of the learnt objects. On the other hand, a low similarity score implies that the test image contains noise, background or unknown object.

Functions for adding and removing models in and from recognition module:

- SERecAddModel (see page 182) - adds a model to recognition module
- SERecRemoveModel (see page 195) - removes a model from recognition module
- SERecGetModelCount (see page 189) - returns a number of models set into recognition module

.NET methods:

- AddModel (see page 280) - adds a model to recognition module
- RemoveModel (see page 283) - removes a model from recognition module
- ModelCount (see page 284) - returns a number of models set into recognition module

Functions for object recognition:

- SERecRecognizeImage (see page 191) - compares one image with models stored in recognition module and returns the id of the nearest model
- SERecGetAllRecognitionDetails (see page 188) - provides detailed information about recognition results.

.NET methods for object recognition:

- Recognize (see page 281) - compares one test image or test model with models stored in recognition module and returns the id of the nearest model
- GetAllRecognitionDetails - returns detailed information about recognition results

4.2 SentiSight API

SentiSight API is intended for visual appearance based object recognition. Thus, its main features are object learning and recognition. The first part, object learning, is dedicated to extraction of object model from a sequence of images containing the object. There are various functions which should help developers to achieve this purpose. The second part, recognition, stands for object recognition in a test image.

4.2.1 Object Learning

In order to recognize an object in an image, the appearance of the object should be memorized. The process of memorizing an appearance of the object from images with various poses is called object learning. A set of images containing the object should be provided to the algorithm and the algorithm extracts so called model - a symbolic representation of the object. It is highly recommended to provide information about exact location of the object in the image. This can be done by the image with a mask of the object. Mask explicitly specifies which pixels of the image present the object and which ones background. Thus, only object specific information will be included into model template. Afterwards, this model can be compressed into more compact representation, which contains less redundant information of the object. The model can be stored for later

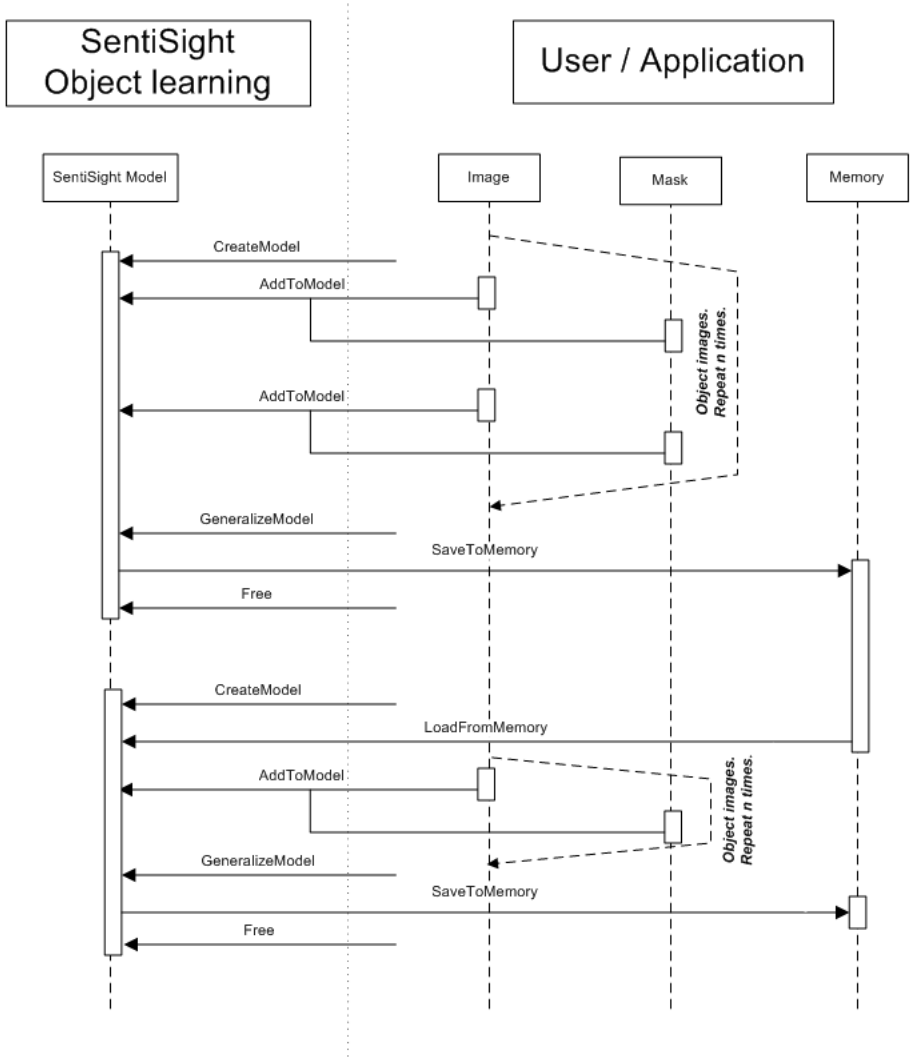
usage and recovered when it is needed.

A model is referenced as SEModel structure in SentiSight API. It should be created using function SECreateModel (see page 173). After creation a model is empty. Model learning is performed sequentially calling SELrnAddToModelEx (see page 175) function with provided handles to model, image and mask.

After object learning is complete, the model can be compressed using function SELrnGeneralizeModelEx (see page 176) into more compact representation. This function removes redundant information from model and thus decreases size of it.

The models can be stored in memory calling SEModelSaveToMemoryEx (see page 181) with provided buffer. Also, this template can be loaded from memory using SEModelLoadFromMemory (see page 180) function with the same buffer.

Figure below shows object learning sequence diagram. At the beginning SentiSight model is created, then several images and their masks are provided to SELrnAddToModelEx (see page 175) function and SentiSight model is filled with data of the object. Afterwards, the SentiSight model is generalized. Finally, it is saved to memory and released.



The quality of object recognition highly depends on model created by object learning part. Thus, a set of images of the object should contain all possible poses of the object - the three dimensional rotations (off plane rotations) are highly recommended. Also, it is recommended that images of the object would be taken under different light conditions or using different light sources in order to improve invariance to diverse light conditions. It is recommended to extract masks which excludes background and reduces the amount of noise and redundant information. Mask images are images of the same height and width as object images. White color areas in mask images indicate object presence. The given mask can be extended in order to eliminate noise and small halls inside the object. The mask extension could be turned on and off using

functions VTSetParameter and VTGetParameter with parameter ID SEP_LRN_ENHANCE_MASK (see page 217) and value true, if one wants to use mask smoothing and false otherwise.

Constraints related to the object and the object learning process could be found in Section Constraints (see page 14).

The practical usage of object learning part could be examined in Section Samples and its Object Learning (see page 7).

4.2.1.1 Foreground\Background Seperation

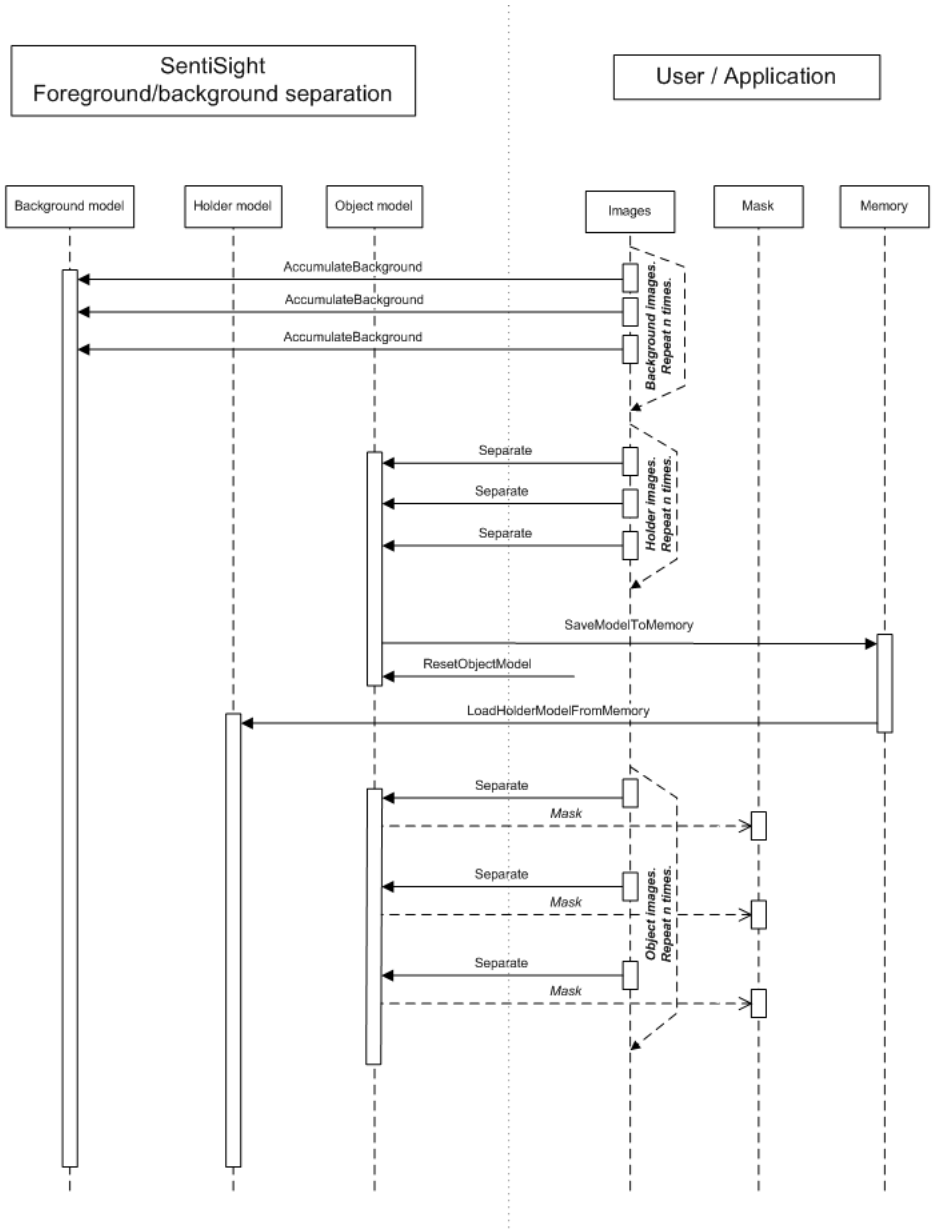
Foreground/Background Separation is a process of automatic extraction of masks of objects from image sequences with static background. The process can be performed using images from a camera on the fly, as well as offline using stored video sequences.

Foreground/Background Separation process consists of three main steps: background accumulation, optional holder learning step and mask separation step. These steps will be described below in this section. First step is background accumulation. Since frames grabbed by camera contain noise, it is necessary to feed several frames of background to the function `SESepAccumulateBackground` (see page 196) to build a model of background. If background has changed it is better to reset it using function `SESepResetBackgroundModel` (see page 199) and then accumulate it again. Background model should be up-to-date, so there is no sense of storing and loading it and that is why there are no functions for it in SentiSight API.

During mask extraction process the object should be moved in front of the camera. Usually, a hand or a stick (it will be referred as holder later) is used to move and rotate the object. If a holder will be used in mask separation process, it is useful to exclude it from object and consider it as background. To achieve this one needs to learn holder model first and then use it during mask extraction process. Function `SESepSeparate` (see page 202) extracts mask from the image and also accumulates internal colour model of the object. So, the holder learning means presenting images of the holder on the same background in various poses to the function `SESepSeparate` (see page 202). After holder learning is complete one can get colour model of the holder by calling `SESepSaveModelToMemory` (see page 201) with provided buffer. The colour model of the holder should be set by calling `SESepLoadHolderFromMemory` with the same buffer as parameter in order to exclude it from the object and consider as background during mask extraction process. Also, the colour model of holder should be removed from internal colour model by calling `SESepResetObjectModel` (see page 200), as the holder was learnt as an object. There is a possibility to remove a set holder model by calling function `SESepResetHolderModel` (see page 200), if one does not want to use holder or wants to use another one. As colour model is sensitive to light conditions, it is recommended to relearn holder model when the visual appearance of the holder has changed because of different light conditions.

The third step is object mask extraction itself. An image of the object on the same background should be passed to `SESepSeparate` (see page 202) function to extract mask. Afterwards the same image together with the extracted mask should be passed to model learning function `SELrnAddToModelEx` (see page 175). The usage of this function and learning process was discussed in previous section Object learning (see page 8). The images of the object should be taken in the same light conditions as a background was accumulated and a camera should face that background. If one wants to use holder to hold and move the object, a colour model of the holder should be set before mask extraction of the object.

Figure shows foreground/background separation sequence diagram. `HSentiSightEngine` (see page 218) handle encapsulates background model, holder model and object model which are used in separation process. First step is background accumulation. Then holder learning takes place. Images of the holder are passed to `SESepSeparate` (see page 202) function which builds internal object model. Colour model of the holder is saved to memory, and then this model is reset. There is no need to accumulate background again, unless it has changed or light conditions are different. Afterwards, memory buffer is set to `HSentiSightEngine` (see page 218) handle as holder model. The third step is object learning and automatic mask extraction. Images of the object are presented to `SESepSeparate` (see page 202) function, which computes masks. As holder is set to the `HSentiSightEngine` (see page 218) handle, it will be excluded from the mask and considered as background.



Separation is used only during acquirement of images of the object. Camera should be placed in front of a static flat background. The light conditions during all the capturing process should be constant. Also, a colour of the object should be distinguishable from a background. Foreground/background separation could be used only if learning environment is arranged so that camera stands still and the object moves in front of it. Otherwise, changes in background would result in inaccurate separation and thus inaccurate mask extraction. More information about limitations of separation process could be found in Section Constraints (see page 14).

The demonstration application showing foreground/background separation process is described in Section Sample.

4.2.2 Recognition

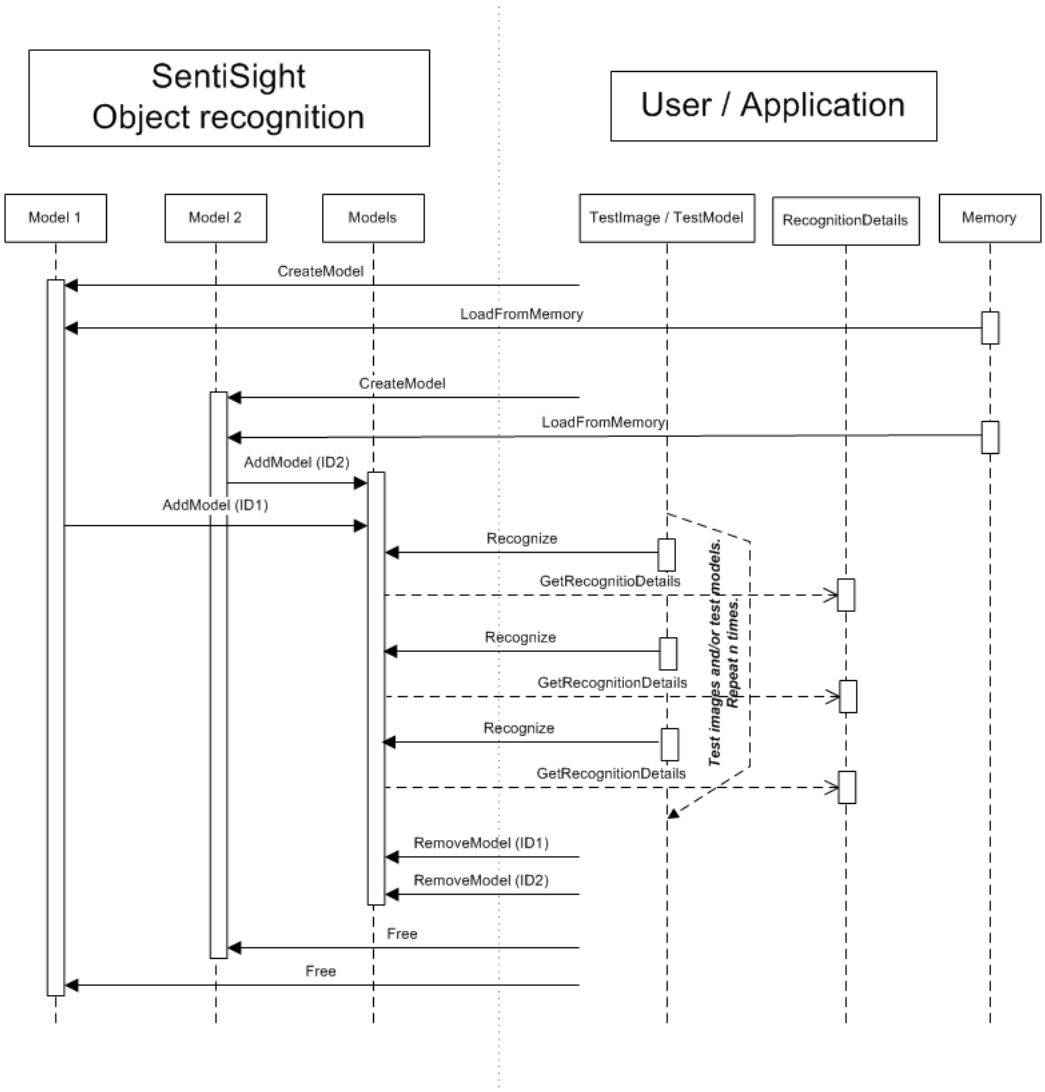
Recognition is a process of identification whether an image (later called a test image) or a model (later referred as test model) contains an object or not. Recognition compares models which were extracted by Object Learning part, with current test image or test model, and returns a comparative score (later referred as similarity score or similarity). A high similarity

score suggests that the test image (or model) contains one of the learnt objects. On the other hand, a low similarity score implies that the test image (or model) contains noise, background or unknown object. The score is different using different recognition speed or different learning mode. See SERecSpeed (see page 216), SELrnMode (see page 215).

Before recognition takes place, one ore more models must be loaded into SentiSight recognition module. Manipulation with models in SentiSight is achieved through functions SERecAddModelEx (see page 183), SERecRemoveAllModels (see page 194) and SERecGetModelCount (see page 189). Recognition of test image is called by function SERecRecognizeImageEx (see page 193). Recognition of test model is called by SERecRecognizeModel. Here mentioned test model is a SentiSight model filled with single one image of one object (refer to object learning process). These recognition functions returns identifier of the nearest model in recognition module. More detailed information (e.g. similarity score, coordinates of the object in test image) about recognition details could be got by calling SERecGetAllRecognitionDetails (see page 188) function. Having this information, a caller of recognition function should decide whether the test image (or model) contains learnt object or not.

There is a possibility to track the object in a sequence of test images. Tracking is called from the object recognition function SERecRecognizeImageEx (see page 193), only a tracking parameter should be turned on before recognition function is called. A parameter SEP_REC_USE_TRACKING (see page 218) with values true or false can be adjusted through functions SESetParameter and SEGetParameter. This parameter is ignored if SERecRecognizeModel function was called.

Figure below shows object recognition process. At the beginning SentiSight model (referred as Model 1 in diagram) is created, then its contents are load from memory. Similarly, another SentiSight model (referred as Model 2 in diagram) is created and its contents are load from memory. These two SentiSight models are added to the HSentiSightEngine (see page 218) handle and they will be used by object recognition function. Test images and test models are presented to SERecRecognizeImage (see page 191) (or SERecRecognizeModel) function to check whether they contain one of these two objects or not. Diagram shows, that recognition functions return RecognitionDetails structure. In fact, recognition functions return best recognized model id which can be used in SERecGetRecognitioDetails function to extract detailed recognition results. When object recognition is finished, the models are removed from HSentiSightEngine (see page 218) handle (by providing ModelID to SERecRemoveModel (see page 195) function). Finally, SentiSight models are released.



Recognition might fail if the model does not contain particular pose of the object, object scale differs significantly from the learnt one, or the object was learnt under different light conditions. This problem could be solved by extending model template with new poses, scales or images taken in new lighting conditions. The other reason of matching failure is blurred test image or too small part of the image containing the object. In such cases the resolution of the test image should be increased or new frame of not blurred object should be captured. More detailed information about constraints and possible inaccuracies can be found in Section Constraints.

False acceptance is the event of confirming object presence in the test image which actually does not contain the object. It can happen due to inaccurate template (object model contains a lot of noise, background, etc.), or the object is simply similar to test image by its appearance. High False Acceptance Rate (FAR) could suggest that one tries to recognize very similar objects. It is the case when two or more models contain similar data and recognition fluctuates between these templates (SERecGetRecognitionDetails (see page 190) contains non constant template ID). False rejection is the event of denial a presence of the object in the test image where it actually is. It can happen if the model of the object differs from the current test image (e.g. different contrast of the object since a light source has changed). Also, the object in the test image could be poorly textured and, as a consequence, the algorithm was unable to locate it in the image. Furthermore, the object will not be localized, if resolution of the object in the test image is too low. Working sample of recognition is demonstrated and analyzed in Section Sample.

Comparison and determination of transformation between two images

1. Add one image to an empty model.

2. Add this model to the recognition engine.
3. Recognize another image.
4. If image was recognized, that means images have common parts, it is possible to get transformations from recognition details. If images have several common parts or an image is ambiguous, several recognitions can occur.

4.2.3 Constraints

SentiSight API is intended for appearance based object recognition, so it performs best with constant exterior objects. The algorithm uses texture to localize the object, so highly discriminatively textured objects are recognized better than poorly textured ones. It is recommended that the object would have texture inside, and not on the border of it, since the appearance of borders changes with even small 3D rotations. Also, recognition of objects with some moving parts could decrease due to local changes of textures. This is also the case of rugged objects, since the appearance of them is sensitive to light changes and 3D rotation.

Transparent objects change their appearance as pose and background changes, so a recognition rate of them decreases. Also, an appearance of objects with shiny parts usually is sensitive to a direction of light, light sources and even to simple affine transformations of the object. As a result of the conditions, it appears some light peaks and shadows on the surface of such objects and outside changes rapidly. As a consequence, recognition decreases rapidly, unless the same light peaks and shadows were present in the model template of the object.

4.2.3.1 Object Learning and Recognition Constraints

Performance of recognition part is highly dependent on the model which was extracted during object learning. The model should contain various poses of the object taken in diverse light conditions. Various poses stands for front side and backside of the object and all possible 3D rotations (off plane rotations) of it. On the other hand, translation, planar rotation (in plane, orthogonal to a camera) and small scaling (till the object does not change its appearance due to resolution change) are fully reconstructed by the library. Various light directions and sources change an appearance of the object, so the model should include images taken under a range of light conditions. Noisy and blurred images could decrease quality of the model, and should be removed from it.

It is highly recommended to use mask images in object learning as they reduce amount of irrelevant information of background. Mask images, if they are accurate, explicitly show position of the object and discard background. As a result, matching part would try to find an appropriate object in the test image, but not a part of background. Oppositely, if mask images were not used or they were imprecise, some part of background will be introduced in the model and probably will be recognized as the object. As a consequence, if few model templates would contain similar background, they could be confused.

4.2.3.2 Foreground/Background Seperation Constraints

Main requirement for foreground/background separation is static background and movable object. A camera for image capturing should face the background and be still while the object is being moved in front of it.

When using foreground/background separation it is important to have constant light conditions during the whole process. Violation of this constraint results in inaccurate masks of images. If background is accumulated in one light condition and the light changes during holder learning or object learning part, then library fails to separate background from the object correctly. During holder learning it is important to have quite a big amount of samples from various poses of holder; otherwise holder will partly be incorporated into object model and possibly downgrade the recognition.

4.3 NDeviceManager API

NDeviceManager (see page 85) library provides functionality for working with cameras. It enables developers to adjust camera parameters automatically, grab single frame and stream frames from camera. Some tutorials of usage of NDeviceMaanger API can be found in Camera Manager tutorial (see page 19) for programming with C/C++, and Camera Manager Tutorial for programing with .NET.

4.4 Image Support

Image support in the *SentiSight 2.1 SDK* can be divided into the following four parts:

- Image (see page 15). The base of all image support. Developers should start using this part and take advantage of other parts if it is required.
- Image Format (see page 16). Declares the supported image formats. Shows how to load and save images in a format-neutral way.
- Image File (see page 17). Should be used if multiple images are stored in one file and more than one image should be loaded from the file.
- Low-Level Image Input-Output (see page 17). Should be used to have more control on how images are loaded and saved in particular format.

4.4.1 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 SentiSight 2.1 SDK is defined by `HNImage` (see page 125) handle in `NImage` module (`NImage` class in .NET). 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` (see page 124), `NImageGetHeight` (see page 121), `NImageGetStride` (see page 123), `NImageGetPixelFormat` (see page 121) and `NImageGetPixels` (see page 122) functions (`Width`, `Height`, `Stride`, `PixelFormat` and `Pixels` properties in .NET) in API Reference for more information.

An image can have horizontal and vertical resolution attributes assigned to it if they are applicable (they are required for fingerprint image, and do not make sense for face image). See `NImageGetHorzResolution` (see page 121) and `NImageGetVertResolution` (see page 124) functions (`HorzResolution` and `VertResolution` properties in .NET) in API Reference for more information.

An image can be created either as empty or from existing memory block. See `NImageCreate` (see page 115), `NImageCreateFromData` (see page 116) and `NImageCreateWrapper` (see page 119) functions (`Create`, `FromData` and `GetWrapper` methods in .NET) for more information.

For each value of `NPixelFormat` (see page 137) (`NPixelFormat` (see page 137) in .NET) exposed via interface a module (subclass of `NImage` in .NET) is provided for managing according type of image (getting and setting

individual pixels, etc.). See NGrayscaleImage, NMonochromeImage and NRgbImage modules (NGrayscaleImage, NMonochromeImage and NRgbImage classes in .NET) for more information.

An image can be converted to different pixel format using NImageCreateFromImage (see page 118) function (FromImage method in .NET).

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

- On Windows BmpSaveImageToHBitmap (see page 142) function (ToHBitmap method in .NET) can be used to receive a standard Win32 HBITMAP for the image. The reverse process is also possible using BmpLoadImageFromHBitmap (see page 140) function (FromHBitmap method in .NET).
- In .NET ToBitmap method can be used to receive a standard .NET Bitmap. The reverse process is also possible using FromBitmap method.
- On Linux there is no easy method implemented. However, a memory block containing pixels of image could be accessed via NImageGetPixelFormat (see page 121) function (PixelFormat method in .NET). 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 (see page 124) function (Save method in .NET).

An image stored in file in any supported image format can be loaded using NImageCreateFromFile (see page 117) function (FromFile method in .NET).

Files containing more than one image are also supported. See Image File (see page 17) and Image Format (see page 16) sections for more information.

4.4.2 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 *SentiSight 2.1 SDK* is defined by HNImageFormat (see page 112) handle in NImageFormat module (NImageFormat class in .NET).

There is a number of image formats supported in the *SentiSight 2.1 SDK*. 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	In .NET only	In .NET only
NIST IHead	Yes	Yes
JPEG	Yes	Yes
Lossless JPEG	Yes	Yes
JPEG 2000	Yes	Yes
PNG	Yes	Yes
TIFF	Yes	In .NET only
WSQ	Yes	Yes

Image formats from the table are accessible using these functions:

- NImageFormatGetBmp (see page 101)
- NImageFormatGetIHead (see page 104)
- NImageFormatGetTiff (see page 106)

- NImageFormatGetWsq (see page 106)

For .NET read-only fields Bmp, Gif, IHead, Jpeg, Png, Tiff and Wsq are used.

To find out which images formats are supported in the SentiSight 2.1 SDK in version-independent way these functions should be used:

- NImageFormatGetFormatCount (see page 103)
- NImageFormatGetFormat (see page 103)

Name, file name pattern (file filter) and default file extension of the image format can be retrieved using NImageFormatGetName (see page 105), NImageFormatGetFileFilter (see page 102) and NImageFormatGetDefaultFileExtension (see page 102) functions (Name, FileFilter and DefaultFileExtension properties in .NET).

To find out which image format should be used to read or write a particular file NImageFormatSelect (see page 112) function (Select method in .NET) should be used.

An image can be loaded and saved from/to file or memory buffer using these functions:

- NImageFormatLoadImageFromFile (see page 107)
- NImageFormatLoadImageFromMemory (see page 107)
- NImageFormatSaveImageToFile (see page 110)
- NImageFormatSaveImageToMemory (see page 111)

(LoadImage and SaveImage methods for .NET). Note that not all image formats support both reading and writing. Use NImageFormatCanRead (see page 100) and/or NImageFormatCanWrite (see page 100) function(s) (CanRead and/or CanWrite property(ies) in .NET) to check if the particular image format does.

If image file contains more than one image then image file can be opened using NImageFormatOpenFile (see page 108) or NImageFormatOpenFileFromMemory (see page 108) function (OpenFile method in .NET). Image file further can be used to read all images from the file.

If multiple images should be saved in one file NImageFormatSaveImagesToFile (see page 110) function (SaveImages method in .NET) should be used. Note that not all image formats support writing of multiple images. Use NImageFormatCanWriteMultiple (see page 101) function (CanWriteMultiple property in .NET) to check if the particular image format does.

4.4.3 Image File

Image file in the *SentiSight 2.1 SDK* is defined by HNImageFile (see page 153) handle in NImageFile module (NImageFile class in .NET). HNImageFile (see page 153) handle is an encapsulation of an opened read-only file containing one or more images.

An image file is opened using NImageFileCreate (see page 151) function (FromFile method in .NET). Also an image file can be opened using image format.

Images are read from image file subsequently calling NImageFileReadImage (see page 153) function (ReadImage method in .NET) until HNImage (see page 125) (NImage in .NET) handle returns NULL (see page 82) (null in .NET).

4.4.4 Low-Level Image Input-Output

Low-level image I/O in the *SentiSight 2.1 SDK* is implemented in Bmp, IHead, Tiff and Wsq modules (Bmp, IHead, Tiff

and `Wsq` classes in .NET).

These modules (classes in .NET) provides functions (static methods in .NET) for loading and saving images in according format (BMP, NIST IHead, TIFF and WSQ).

Those functions (static methods in .NET) can take parameters that precisely control loading and saving of the image in particular formats. For example, bit rate is specified when saving in WSQ format.

4.5 OpenCV License

Intel License Agreement

For Open Source Computer Vision Library

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistribution's of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistribution's in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The name of Intel Corporation may not be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided by the copyright holders and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed.

In no event shall the Intel Corporation or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused

and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

5 Tutorials

SentiSight 2.1 SDK contains these tutorials:

- Object learning (see page 19)
- Object recognition (see page 20)
- Object separation (see page 20)
- Video file reading (see page 20)
- Camera manager (see page 19)

Tutorials are used to demonstrate how to use functionality of SentiSight 2.1 SDK. All tutorials are written in C and C# programming languages. Tutorials are saved under `\tutorials` folder. Tutorials can be built using Microsoft Visual Studio solution (*.sln) file included into distribution.

5.1 DeviceManager

5.1.1 CameraManager

This tutorial explains usage of NDeviceManager (see page 85) library in C programming language and Neurotec.DeviceManager (see page 305) library in .NET programming languages. It demonstrates enumeration of cameras connected to the computer and demonstrates grabbing of frames from cameras.

5.2 SentiSight

SentiSight tutorials are used to demonstrate how to use functionality of SentiSight (see page 168) and Neurotec.SentiSight (see page 272) libraries. SDK includes 3 major tutorials: object learning (see page 19), foreground/background separation (see page 20) and object recognition (see page 20).

5.2.1 Learning

This tutorial describes an object learning process using C++ or C# programming language. Object learning is required for memorizing an appearance of the object from a number of images. A special model is allocated and filled during this process. More information about object learning can be found in Section Object Learning (see page 7).

5.2.2 Recognition

This tutorial describes an object recognition process using C++ and C# programming language. This process is dedicated to find an object in a test image. More information about this process can be found in Section Object Recognition (see page 8).

5.2.3 Separation

Separation tutorial is used to separate foreground (an object) and background. Separation process is intended for automatic mask extraction of the object. More information about this process can be found in Section Foreground/Background Separation (see page 7) from Images.

Foreground and background separation tutorials are written in C and C# programming languages.

5.3 Video

5.3.1 VideoFileReading

Video file reading tutorial explains usage of VideoReader (see page 162) library in C and C# programming languages. Using this tutorial video file can be loaded and a frame grabbed from it. This tutorial requires a video file. Test video files can be downloaded from http://www.neurotechnology.com/ and placed in `/data` folder.

6 PiXORD N606

NDeviceManager (see page 85) library supports Pixord N606 network camera. In order to use this camera, it is necessary to change parameters in `CmmPixordIpCameras.ini` file within `\bin\Win32_x86\Cmm` folder of SDK. Configuration file format:

rtsp://<ip of the camera>/<channel>

<ip of the camera> - either IP of the camera or domain name of the camera

<channel> - channel name that can be set through camera's web interface

Files

Files for this camera are placed in `\bin\Win32_x86\Cmm\Additional\`

- `CmmPixord.dll`
- `CmmPixord.so`

These files must be copied to `\bin\Win32_x86\Cmm\` before using Pixord camera.

7 Samples

SentiSight 2.1 SDK includes two sample applications which demonstrates how to use product and recognize objects. There are two main samples in SentiSight SDK:

- SentiSight Demo
- SentiSight Demo Lite

Both of them demonstrates object learning and recognition, foreground/background separation and other features of SentiSight library. Difference is that Demo Lite application is a wizard which guides through basic features of SentiSight, so it can be a good starting point for SentiSight 2.1 SDK.

SentiSight Demo application is written in C++ programming language. SentiSight Demo Lite application is written in C++ and C# programming languages.

Notes

Documentation for SentiSight 2.1 SDK sample application is provided in separate files within `documentation` folder. `SentiSight Demo.pdf` and `SentiSight Demo Lite.pdf` files contain documentation respectively for `SentiSight Demo` and `SentiSight Demo Lite` applications.

7.1 Running sample (Windows)

Demo applications on Windows can be started using `exe` files which can be found in `bin` folder of distribution.

SentiSight Demo

SentiSight Demo application can be started using `/bin/Win32_x86/SentiSightSampleWX.exe` file. Sample images and videos (data files) for this tutorial can be downloaded from http://www.neurotechnology.com/download.html#sentisight.

Source files for SentiSight Demo are located under `samples\SentiSight\CPP\SentiSightSampleWX` folder.

For more information on this demo application read `SentiSight Demo.pdf` document in `documentation` folder.

SentiSight Demo Lite

Self-explanatory SentiSight Demo Lite application can be started using `/bin/Win32_x86/SentiSightSampleLiteCS.exe` (compiled C# application) or `/bin/Win32_x86/SentiSightSampleLiteWX.exe` (compiled C++ application) files. You should note that these applications have limited capabilities and require web camera. If you need full functionality of SentiSight, use `SentiSightSampleWX.exe` application.

Source files for SentiSight Demo Lite are saved under `samples\SentiSight\CPP\SentiSightSampleLiteWX` and `samples\SentiSight\CS\SentiSightSampleLiteCS` folders.

For more information on this demo application read `SentiSight Demo Lite.pdf` document in `documentation` folder.

Notes

If you need to compile sample applications yourself, then use Microsoft Visual Studio project files (`*.sln`). Also you should note that C++ samples require wxWidgets installed on computer. Read wxWidgets Compilation instructions for more information.

7.2 Running sample (Linux)

Before running the samples, you need to activate your SentiSight installation license. For detailed instructions on license activation please read `Activation.pdf`.

The sample requires gtk+-2.4.4 or newer. Many Linux distributions include gtk+. If your distribution does not include GTK+ you can build it from source.

- `./SentiSightSampleWX`

To compile sample from source it is necessary to have wxWidgets 2.8.8 or newer. Read section wxWidgets Compiliation (see page 23) for more information.

7.3 wxWidgets Compilation

wxWidgets library can be downloaded from http://www.wxwidgets.org/. Before using wxWidgets you should compile it. To compile wxWidgets as a static library do the following steps:

1. Open solution file `C:\wxWidgets-2.8.8\build\msw\wx.dsw` (in case wxWidgets are located in C disk).
2. Set **`#define wxUSE_GRAPHICS_CONTEXT 1`** define in include\wx\msw\setup.h.
3. Select all projects and change C/C++/Code Generation/Runtime library to Multi-threaded.
4. Build Unicode Release and Unicode Debug configurations.

(Compile wxWidgets and your applications using the same Visual Studio that was used for sample compilation (Visual Studio 2005 or later) otherwise it will lead to compilation errors)

Finally, Visual Studio include and library paths have to be setup. Go to Tools->Options->Projects and Solutions->VC++ Directories and include these directories and library file from these directories:

- C:\wxWidgets-2.8.8\include
- C:\wxWidgets-2.8.8\include\msvc

Lib:

- C:\wxWidgets-2.8.8\lib\vc_lib

8 API Reference

Modules

Name	Description
.NET (see page 219)	

8.1 C

8.1.1 NCore Library

Provides infrastructure for Neurotechnology components.

Remarks

Requirements (Windows)

- **Import Library**: NCore.dll.lib.
- **DLL**: NCore.dll.

Requirements (Linux)

- **Shared object**: libNCore.so.

Modules

Name	Description
NCore Module (see page 24)	Provides infrastructure/basic functionality for Neurotechnology components.
NErrors Module (see page 27)	Defines error codes used in Neurotechnology components.
NGeometry Module (see page 33)	Provides definitions of geometrical structures types.
NLibraryInfo Module (see page 37)	Provides definitions of library info structure type.
NMemory Module (see page 39)	Provides memory management for Neurotechnology components.
NObject Module (see page 45)	Provides functionality for retrieving information about specified object.
NParameters Module (see page 50)	Provides functionality for working with parameters for Neurotechnology components.
NProcessorInfo Module (see page 54)	Provides functionality for getting processor information.
NStream Module (see page 60)	Supports internal Neurotechnology libraries infrastructure and should not be used directly in your code.
NTypes Module (see page 62)	Defines types and macros used in Neurotechnology components.

8.1.1.1 NCore Module

Provides infrastructure/basic functionality for Neurotechnology components.

Files

Name	Description
NCore.h (see page 26)	Header file for the NCore (see page 24) module. Provides infrastructure/basic functionality for Neurotechnology components.

Functions

	Name	Description
	NCoreGetInfo (see page 25)	Retrieves information about the library.
	NCoreOnExit (see page 25)	This function should be called on programs exit.
	NCoreOnStart (see page 25)	This function should be called on programs start.
	NCoreOnThreadExit (see page 26)	This function should be called when program's thread finished its task.
	NCoreOnThreadStart (see page 26)	This function should be called when new thread is created.

8.1.1.1.1 Functions

8.1.1.1.1.1 NCoreGetInfo Function

Retrieves information about the library.

C++

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

Parameters

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

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

NCore Module (see page 24)

8.1.1.1.1.2 NCoreOnExit Function

This function should be called on programs exit.

C++

```
void N_API NCoreOnExit();
```

Notes

This function works on Lib-No-Dll and VAR products.

Module

NCore Module (see page 24)

8.1.1.1.1.3 NCoreOnStart Function

This function should be called on programs start.

C++

```
void N_API NCoreOnStart();
```

Notes

This function works on Lib-No-Dll and VAR products.

Module

NCore Module (see page 24)

8.1.1.1.1.4 NCoreOnThreadExit Function

This function should be called when program's thread finished its task.

C++

```
void N_API NCoreOnThreadExit();
```

Notes

This function works on Lib-No-Dll and VAR products.

Module

NCore Module (see page 24)

8.1.1.1.1.5 NCoreOnThreadStart Function

This function should be called when new thread is created.

C++

```
void N_API NCoreOnThreadStart();
```

Notes

This function works on Lib-No-Dll and VAR products.

Module

NCore Module (see page 24)

8.1.1.1.2 Files

8.1.1.1.2.1 NCore.h

Header file for the NCore (see page 24) module. Provides infrastructure/basic functionality for Neurotechnology components.

Functions

	Name	Description
	NCoreGetInfo (see page 25)	Retrieves information about the library.
	NCoreOnExit (see page 25)	This function should be called on programs exit.
	NCoreOnStart (see page 25)	This function should be called on programs start.
	NCoreOnThreadExit (see page 26)	This function should be called when program's thread finished its task.
	NCoreOnThreadStart (see page 26)	This function should be called when new thread is created.

Module

NCore Module (see page 24)

8.1.1.2 NErrors Module

Defines error codes used in Neurotechnology components.

Files

Name	Description
NErrors.h (see page 32)	Header file for NErrors module. Defines error codes used in Neurotechnology components.

Macros

Name	Description
N_E_ARGUMENT (see page 27)	This error occurs when argument value is invalid.
N_E_ARGUMENT_NULL (see page 28)	When argument value is NULL (see page 82) where non-NULL (see page 82) value was expected this error occurs.
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	Argument value is out of range.
N_E_CLR (see page 28)	The Common language runtime (CLR) exception has occurred.
N_E_COM (see page 28)	COM error has occurred.
N_E_CORE (see page 28)	Standard error has occurred (for internal use).
N_E_END_OF_STREAM (see page 29)	This error occurs when is attempted to read file or buffer after its end.
N_E_EXTERNAL (see page 29)	Error in external code has occurred (for internal use).
N_E_FAILED (see page 29)	Unspecified error has occurred.
N_E_FORMAT (see page 29)	This error occurs when format of argument value is invalid.
N_E_INDEX_OUT_OF_RANGE (see page 29)	This error occurs when index is out of range (for internal use).
N_E_INVALID_OPERATION (see page 29)	Attempted to perform invalid operation.
N_E_IO (see page 30)	Input/output error has occurred.
N_E_NOT_ACTIVATED (see page 30)	Product not activated.
N_E_NOT_IMPLEMENTED (see page 30)	This error occurs when trying to use functionality which is not implemented yet.
N_E_NOT_SUPPORTED (see page 30)	This error occurs when trying to use functionality which is not supported.
N_E_NULL_REFERENCE (see page 30)	Null reference has occurred (for internal use).
N_E_OUT_OF_MEMORY (see page 31)	This error occurs when there were not enough memory to proceed task.
N_E_OVERFLOW (see page 31)	Arithmetic overflow has occurred.
N_E_PARAMETER (see page 31)	Parameter id is invalid.
N_E_PARAMETER_READ_ONLY (see page 31)	Attempted to set read only parameter.
N_E_SYS (see page 31)	System error.
N_E_WIN32 (see page 31)	Win32 error has occurred.
N_OK (see page 32)	This value is returned when no error has occurred.
NFailed (see page 32)	Determines whether function result indicates error.
NSucceeded (see page 32)	Determines whether function result indicates success.

8.1.1.2.1 Macros

8.1.1.2.1.1 N_E_ARGUMENT Macro

This error occurs when argument value is invalid.

C++

```
#define N_E_ARGUMENT -10
```

Module

NErrors Module (see page 27)

8.1.1.2.1.2 N_E_ARGUMENT_NULL Macro

When argument value is NULL (see page 82) where non-NULL (see page 82) value was expected this error occurs.

C++

```
#define N_E_ARGUMENT_NULL -11
```

Module

NErrors Module (see page 27)

8.1.1.2.1.3 N_E_ARGUMENT_OUT_OF_RANGE Macro

Argument value is out of range.

C++

```
#define N_E_ARGUMENT_OUT_OF_RANGE -12
```

Module

NErrors Module (see page 27)

8.1.1.2.1.4 N_E_CLR Macro

The Common language runtime (CLR) exception has occurred.

C++

```
#define N_E_CLR -93
```

Module

NErrors Module (see page 27)

8.1.1.2.1.5 N_E_COM Macro

COM error has occurred.

C++

```
#define N_E_COM -92
```

Module

NErrors Module (see page 27)

8.1.1.2.1.6 N_E_CORE Macro

Standard error has occurred (for internal use).

C++

```
#define N_E_CORE -2
```

Module

NErrors Module (see page 27)

8.1.1.2.1.7 N_E_END_OF_STREAM Macro

This error occurs when is attempted to read file or buffer after its end.

C++

```
#define N_E_END_OF_STREAM -15
```

Module

NErrors Module (see page 27)

8.1.1.2.1.8 N_E_EXTERNAL Macro

Error in external code has occurred (for internal use).

C++

```
#define N_E_EXTERNAL -90
```

Module

NErrors Module (see page 27)

8.1.1.2.1.9 N_E_FAILED Macro

Unspecified error has occurred.

C++

```
#define N_E_FAILED -1
```

Module

NErrors Module (see page 27)

8.1.1.2.1.10 N_E_FORMAT Macro

This error occurs when format of argument value is invalid.

C++

```
#define N_E_FORMAT -13
```

Module

NErrors Module (see page 27)

8.1.1.2.1.11 N_E_INDEX_OUT_OF_RANGE Macro

This error occurs when index is out of range (for internal use).

C++

```
#define N_E_INDEX_OUT_OF_RANGE -9
```

Module

NErrors Module (see page 27)

8.1.1.2.1.12 N_E_INVALID_OPERATION Macro

Attempted to perform invalid operation.

C++

```
#define N_E_INVALID_OPERATION -7
```

Module

NErrors Module (see page 27)

8.1.1.2.1.13 N_E_IO Macro

Input/output error has occurred.

C++

```
#define N_E_IO -14
```

Module

NErrors Module (see page 27)

8.1.1.2.1.14 N_E_NOT_ACTIVATED Macro

Product not activated.

C++

```
#define N_E_NOT_ACTIVATED -200
```

See Also

For more information about product activation see `Activation.pdf` within `documentation` folder of SDK.

Module

NErrors Module (see page 27)

8.1.1.2.1.15 N_E_NOT_IMPLEMENTED Macro

This error occurs when trying to use functionality which is not implemented yet.

C++

```
#define N_E_NOT_IMPLEMENTED -5
```

Module

NErrors Module (see page 27)

8.1.1.2.1.16 N_E_NOT_SUPPORTED Macro

This error occurs when trying to use functionality which is not supported.

C++

```
#define N_E_NOT_SUPPORTED -6
```

Module

NErrors Module (see page 27)

8.1.1.2.1.17 N_E_NULL_REFERENCE Macro

Null reference has occurred (for internal use).

C++

```
#define N_E_NULL_REFERENCE -3
```

Module

NErrors Module (see page 27)

8.1.1.2.1.18 N_E_OUT_OF_MEMORY Macro

This error occurs when there were not enough memory to proceed task.

C++

```
#define N_E_OUT_OF_MEMORY -4
```

Module

NErrors Module (see page 27)

8.1.1.2.1.19 N_E_OVERFLOW Macro

Arithmetic overflow has occurred.

C++

```
#define N_E_OVERFLOW -8
```

Module

NErrors Module (see page 27)

8.1.1.2.1.20 N_E_PARAMETER Macro

Parameter id is invalid.

C++

```
#define N_E_PARAMETER -100
```

Module

NErrors Module (see page 27)

8.1.1.2.1.21 N_E_PARAMETER_READ_ONLY Macro

Attempted to set read only parameter.

C++

```
#define N_E_PARAMETER_READ_ONLY -101
```

Module

NErrors Module (see page 27)

8.1.1.2.1.22 N_E_SYS Macro

System error.

C++

```
#define N_E_SYS -94
```

Module

NErrors Module (see page 27)

8.1.1.2.1.23 N_E_WIN32 Macro

Win32 error has occurred.

C++

```
#define N_E_WIN32 -91
```

Module

NErrors Module (see page 27)

8.1.1.2.1.24 N_OK Macro

This value is returned when no error has occurred.

C++

```
#define N_OK 0
```

Module

NErrors Module (see page 27)

8.1.1.2.1.25 NFailed Macro

Determines whether function result indicates error.

C++

```
#define NFailed(result) ((result) < 0)
```

Module

NErrors Module (see page 27)

8.1.1.2.1.26 NSucceeded Macro

Determines whether function result indicates success.

C++

```
#define NSucceeded(result) ((result) >= 0)
```

Module

NErrors Module (see page 27)

8.1.1.2.2 Files

8.1.1.2.2.1 NErrors.h

Header file for NErrors module. Defines error codes used in Neurotechnology components.

Macros

Name	Description
N_E_ARGUMENT (see page 27)	This error occurs when argument value is invalid.
N_E_ARGUMENT_NULL (see page 28)	When argument value is NULL (see page 82) where non-NULL (see page 82) value was expected this error occurs.
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	Argument value is out of range.
N_E_CLR (see page 28)	The Common language runtime (CLR) exception has occurred.
N_E_COM (see page 28)	COM error has occurred.
N_E_CORE (see page 28)	Standard error has occurred (for internal use).
N_E_END_OF_STREAM (see page 29)	This error occurs when is attempted to read file or buffer after its end.
N_E_EXTERNAL (see page 29)	Error in external code has occurred (for internal use).
N_E_FAILED (see page 29)	Unspecified error has occurred.
N_E_FORMAT (see page 29)	This error occurs when format of argument value is invalid.

N_E_INDEX_OUT_OF_RANGE (see page 29)	This error occurs when index is out of range (for internal use).
N_E_INVALID_OPERATION (see page 29)	Attempted to perform invalid operation.
N_E_IO (see page 30)	Input/output error has occurred.
N_E_NOT_ACTIVATED (see page 30)	Product not activated.
N_E_NOT_IMPLEMENTED (see page 30)	This error occurs when trying to use functionality which is not implemented yet.
N_E_NOT_SUPPORTED (see page 30)	This error occurs when trying to use functionality which is not supported.
N_E_NULL_REFERENCE (see page 30)	Null reference has occurred (for internal use).
N_E_OUT_OF_MEMORY (see page 31)	This error occurs when there were not enough memory to proceed task.
N_E_OVERFLOW (see page 31)	Arithmetic overflow has occurred.
N_E_PARAMETER (see page 31)	Parameter id is invalid.
N_E_PARAMETER_READ_ONLY (see page 31)	Attempted to set read only parameter.
N_E_SYS (see page 31)	System error.
N_E_WIN32 (see page 31)	Win32 error has occurred.
N_OK (see page 32)	This value is returned when no error has occurred.
NFailed (see page 32)	Determines whether function result indicates error.
NSucceeded (see page 32)	Determines whether function result indicates success.

Module

NErrors Module (see page 27)

8.1.1.3 NGeometry Module

Provides definitions of geometrical structures types.

Files

Name	Description
NGeometry.h (see page 37)	Header file for the NGeometry module. Provides definitions of geometrical structures types.

Structs, Records, Enums

	Name	Description
	NPoint_ (see page 34)	Structure defining point coordinates in 2D space.
	NPointD_ (see page 34)	Structure defining point coordinates in 2D space.
	NPointF_ (see page 34)	Structure defining point coordinates in 2D space.
	NRect_ (see page 35)	Structure defining a rectangle figure in 2D space.
	NRectD_ (see page 35)	Structure defining a rectangle figure in 2D space.
	NRectF_ (see page 35)	Structure defining a rectangle figure in 2D space.
	NSize_ (see page 36)	Structure defining rectangle size.
	NSizeD_ (see page 36)	Structure defining rectangle size.
	NSizeF_ (see page 36)	Structure defining rectangle size.
	NPoint (see page 34)	Structure defining point coordinates in 2D space.
	NPointD (see page 34)	Structure defining point coordinates in 2D space.
	NPointF (see page 34)	Structure defining point coordinates in 2D space.
	NRect (see page 35)	Structure defining a rectangle figure in 2D space.
	NRectD (see page 35)	Structure defining a rectangle figure in 2D space.
	NRectF (see page 35)	Structure defining a rectangle figure in 2D space.
	NSize (see page 36)	Structure defining rectangle size.

	NSizeD (see page 36)	Structure defining rectangle size.
	NSizeF (see page 36)	Structure defining rectangle size.

8.1.1.3.1 Structs, Records, Enums

8.1.1.3.1.1 NPoint Structure

Structure defining point coordinates in 2D space.

C++

```
typedef struct NPoint_ {
  NInt X;
  NInt Y;
} NPoint;
```

Members

Members	Description
NInt X;	Point coordinate on x axis.
NInt Y;	Point coordinate on y axis.

Module

NGeometry Module (see page 33)

8.1.1.3.1.2 NPointD Structure

Structure defining point coordinates in 2D space.

C++

```
typedef struct NPointD_ {
  NDouble X;
  NDouble Y;
} NPointD;
```

Members

Members	Description
NDouble X;	Point coordinate on x axis.
NDouble Y;	Point coordinate on y axis.

Module

NGeometry Module (see page 33)

8.1.1.3.1.3 NPointF Structure

Structure defining point coordinates in 2D space.

C++

```
typedef struct NPointF_ {
  NFloat X;
  NFloat Y;
} NPointF;
```

Members

Members	Description
NFloat X;	Point coordinate on x axis.
NFloat Y;	Point coordinate on y axis.

Module

NGeometry Module (see page 33)

8.1.1.3.1.4 NRect Structure

Structure defining a rectangle figure in 2D space.

C++

```
typedef struct NRect_ {
  NInt X;
  NInt Y;
  NInt Width;
  NInt Height;
} NRect;
```

Members

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

Module

NGeometry Module (see page 33)

8.1.1.3.1.5 NRectD Structure

Structure defining a rectangle figure in 2D space.

C++

```
typedef struct NRectD_ {
  NDouble X;
  NDouble Y;
  NDouble Width;
  NDouble Height;
} NRectD;
```

Members

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

Module

NGeometry Module (see page 33)

8.1.1.3.1.6 NRectF Structure

Structure defining a rectangle figure in 2D space.

C++

```
typedef struct NRectF_ {
  NFloat X;
  NFloat Y;
  NFloat Width;
  NFloat Height;
```

```
} NRectF;
```

Members

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

Module

NGeometry Module (see page 33)

8.1.1.3.1.7 NSize Structure

Structure defining rectangle size.

C++

```
typedef struct NSize_ {
  NInt Width;
  NInt Height;
} NSize;
```

Members

Members	Description
NInt Width;	Width.
NInt Height;	Height.

Module

NGeometry Module (see page 33)

8.1.1.3.1.8 NSizeD Structure

Structure defining rectangle size.

C++

```
typedef struct NSizeD_ {
  NDouble Width;
  NDouble Height;
} NSizeD;
```

Members

Members	Description
NDouble Width;	Width.
NDouble Height;	Height.

Module

NGeometry Module (see page 33)

8.1.1.3.1.9 NSizeF Structure

Structure defining rectangle size.

C++

```
typedef struct NSizeF_ {
  NFloat Width;
  NFloat Height;
} NSizeF;
```

Members

Members	Description
NFloat Width;	Width.
NFloat Height;	Height.

Module

NGeometry Module (see page 33)

8.1.1.3.2 Files

8.1.1.3.2.1 NGeometry.h

Header file for the NGeometry module. Provides definitions of geometrical structures types.

Module

NGeometry Module (see page 33)

Structures

	Name	Description
	NPoint_ (see page 34)	Structure defining point coordinates in 2D space.
	NPointD_ (see page 34)	Structure defining point coordinates in 2D space.
	NPointF_ (see page 34)	Structure defining point coordinates in 2D space.
	NRect_ (see page 35)	Structure defining a rectangle figure in 2D space.
	NRectD_ (see page 35)	Structure defining a rectangle figure in 2D space.
	NRectF_ (see page 35)	Structure defining a rectangle figure in 2D space.
	NSize_ (see page 36)	Structure defining rectangle size.
	NSizeD_ (see page 36)	Structure defining rectangle size.
	NSizeF_ (see page 36)	Structure defining rectangle size.
	NPoint (see page 34)	Structure defining point coordinates in 2D space.
	NPointD (see page 34)	Structure defining point coordinates in 2D space.
	NPointF (see page 34)	Structure defining point coordinates in 2D space.
	NRect (see page 35)	Structure defining a rectangle figure in 2D space.
	NRectD (see page 35)	Structure defining a rectangle figure in 2D space.
	NRectF (see page 35)	Structure defining a rectangle figure in 2D space.
	NSize (see page 36)	Structure defining rectangle size.
	NSizeD (see page 36)	Structure defining rectangle size.
	NSizeF (see page 36)	Structure defining rectangle size.

8.1.1.4 NLibraryInfo Module

Provides definitions of library info structure type.

Files

Name	Description
NLibraryInfo.h (see page 39)	Header file for the NLibraryInfo module. Provides definitions of library info structure type.

Macros

Name	Description
N_LI_ACTIVATED_MAX_LENGTH (see page 39)	Maximum length of "Activated" field.

Structs, Records, Enums

	Name	Description
	NLibraryInfo_ (see page 38)	Structure defining information about the library as library title, product name, company name, copyright string and library version.
	NLibraryInfoA_ (see page 38)	Structure defining information about the library as library title, product name, company name, copyright string and library version.

8.1.1.4.1 Structs, Records, Enums

8.1.1.4.1.1 NLibraryInfo_ Structure

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

C++

```
struct NLibraryInfo_ {
  NChar Title[N_LI_TITLE_MAX_LENGTH];
  NChar Product[N_LI_PRODUCT_MAX_LENGTH];
  NChar Company[N_LI_COMPANY_MAX_LENGTH];
  NChar Copyright[N_LI_COPYRIGHT_MAX_LENGTH];
  NInt VersionMajor;
  NInt VersionMinor;
  NInt VersionBuild;
  NInt VersionRevision;
  NChar Activated[N_LI_ACTIVATED_MAX_LENGTH];
};
```

Members

Members	Description
NChar Title[N_LI_TITLE_MAX_LENGTH];	Title of the library.
NChar Product[N_LI_PRODUCT_MAX_LENGTH];	Product name.
NChar Company[N_LI_COMPANY_MAX_LENGTH];	Comapny name
NChar Copyright[N_LI_COPYRIGHT_MAX_LENGTH];	Copyright string for the library.
NInt VersionMajor;	Major part of the library version.
NInt VersionMinor;	Minor part of the library version.
NInt VersionBuild;	Build part of the library version.
NInt VersionRevision;	Revision part of the library version.
NChar Activated[N_LI_ACTIVATED_MAX_LENGTH];	Indicates if the library was activated.

Module

NLibraryInfo Module (see page 37)

8.1.1.4.1.2 NLibraryInfoA_ Structure

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

C++

```
struct NLibraryInfoA_ {
};
```

Notes

This structure is used if compiling for platform without ANSI versions of the functions support. If can not choose between functions NLibraryInfoA and NLibraryInfoW, you can use function without suffix A or W.

Module

NLibraryInfo Module (see page 37)

8.1.1.4.2 Macros

8.1.1.4.2.1 N_LI_ACTIVATED_MAX_LENGTH Macro

Maximum length of "Activated" field.

C++

```
#define N_LI_ACTIVATED_MAX_LENGTH 256
```

Module

NLibraryInfo Module (see page 37)

8.1.1.4.3 Files

8.1.1.4.3.1 NLibraryInfo.h

Header file for the NLibraryInfo module. Provides definitions of library info structure type.

Macros

Name	Description
N_LI_ACTIVATED_MAX_LENGTH (see page 39)	Maximum length of "Activated" field.

Module

NLibraryInfo Module (see page 37)

Structures

	Name	Description
	NLibraryInfo_ (see page 38)	Structure defining information about the library as library title, product name, company name, copyright string and library version.
	NLibraryInfoA_ (see page 38)	Structure defining information about the library as library title, product name, company name, copyright string and library version.

8.1.1.5 NMemory Module

Provides memory management for Neurotechnology components.

Files

Name	Description
NMemory.h (see page 44)	Header file for the NMemory module. Provides memory management for Neurotechnology components.

Functions

	Name	Description
	NAlignedFree (see page 40)	Frees an aligned memory resources.
	NAlloc (see page 40)	Allocates memory block.
	NCAlloc (see page 41)	Allocates memory block with all bytes set to zero.
	NCompare (see page 41)	Compares bytes in two memory blocks.
	NCopy (see page 42)	Copies data between memory blocks.
	NFill (see page 42)	Sets bytes of memory block to specified value.
	NFree (see page 43)	Deallocates memory block.
	NMove (see page 43)	Move data from one memory block to another.
	NReAlloc (see page 43)	Reallocates memory block.

Macros

Name	Description
NClear (see page 44)	Clears memory block.

8.1.1.5.1 Functions

8.1.1.5.1.1 NAlignedFree Function

Frees an aligned memory resources.

C++

```
void N_API NAlignedFree(void * pBlock);
```

Parameters

Parameters	Description
void * pBlock	[out] A pointer to previously allocated memory block to free.

Remarks

If passed NULL (see page 82) as a parameter, function does nothing. Attempting to free an invalid pointer, can cause errors.

Module

NMemory Module (see page 39)

8.1.1.5.1.2 NAlloc Function

Allocates memory block.

C++

```
NResult N_API NAlloc(NSizeType size, void * * ppBlock);
```

Parameters

Parameters	Description
NSizeType size	The size in bytes of memory to be allocated.
void * * ppBlock	[out] A pointer to a newly allocated memory block.

Returns

Return value	Description
N_OK (see page 32)	If function succeeds it returns N_OK (see page 32).

N_E_ARGUMENT_NULL (see page 28)	If provided *ppBlock* value is NULL (see page 82), function fails and returns this error.
N_E_OUT_OF_MEMORY (see page 31)	If there was nor enough memory to allocate *size* bytes this error is returned.

Module

NMemory Module (see page 39)

8.1.1.5.1.3 NCAlloc Function

Allocates memory block with all bytes set to zero.

C++

```
NResult N_API NCAlloc(NSizeType size, void * * ppBlock);
```

Parameters

Parameters	Description
NSizeType size	The size in bytes of memory to be allocated.
void * * ppBlock	[out] A pointer to a newly allocated memory block.

Returns

Return value	Description
N_OK (see page 32)	If function succeeds it returns N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If provided *ppBlock* value is NULL (see page 82), function fails and returns this error.
N_E_OUT_OF_MEMORY (see page 31)	If there was nor enough memory to allocate *size* bytes this error is returned.

Module

NMemory Module (see page 39)

8.1.1.5.1.4 NCompare Function

Compares bytes in two memory blocks.

C++

```
NResult N_API NCompare(const void * pBlock1, const void * pBlock2, NSizeType size, NInt *
pResult);
```

Parameters

Parameters	Description
const void * pBlock1	A pointer to the first memory block.
const void * pBlock2	A pointer to the second memory block.
NSizeType size	Number of bytes to be compared.
NInt * pResult	[out] Compare result.

Returns

Return value	Description
N_OK (see page 32)	If function succeeds it returns N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If provided *ppBlock1* or *ppBlock2* value is NULL (see page 82), function fails and returns this error.

Remarks

If *pResult* value is less than zero then a block specified by *pBlock1* is less than a block specified by *pBlock2*.

If *pResult* has a zero value then a block specified by *pBlock1* is identical to a block specified by *pBlock2*.

If *pResult* is greater than zero then A block specified by *pBlock1* is greater than a block specified by *pBlock2*.

Module

NMemory Module (see page 39)

8.1.1.5.1.5 NCopy Function

Copies data between memory blocks.

C++

```
NResult N_API NCopy(void * pDstBlock, const void * pSrcBlock, NSizeType size);
```

Parameters

Parameters	Description
void * pDstBlock	A destination memory block (memory block to copy from).
const void * pSrcBlock	A source memory block (memory block to copy to).
NSizeType size	The size of memory block to be copied (the size is in bytes).

Returns

Return value	Description
N_OK (see page 32)	If function succeeds it returns N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If provided *pDstBlock* or *pDstBlock* value is NULL (see page 82), function fails and returns this error.

Module

NMemory Module (see page 39)

8.1.1.5.1.6 NFill Function

Sets bytes of memory block to specified value.

C++

```
NResult N_API NFill(void * pBlock, NByte value, NSizeType size);
```

Parameters

Parameters	Description
void * pBlock	A pointer to a destination memory block that contains data.
NByte value	A value to set.
NSizeType size	The size in bytes of memory block to be set.

Returns

Return value	Description
N_OK (see page 32)	If function succeeds it returns N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If provided *pBlock* value is NULL (see page 82), function fails and returns this error.

Module

NMemory Module (see page 39)

8.1.1.5.1.7 NFree Function

Deallocates memory block.

C++

```
void N_API NFree(void * pBlock);
```

Parameters

Parameters	Description
void * pBlock	A pointer to memory block to be deallocated.

Returns

If function succeeds the return value is N_OK (see page 32).

Module

NMemory Module (see page 39)

8.1.1.5.1.8 NMove Function

Move data from one memory block to another.

C++

```
NResult N_API NMove(void * pDstBlock, const void * pSrcBlock, NSizeType size);
```

Parameters

Parameters	Description
void * pDstBlock	A pointer to destination memory block.
const void * pSrcBlock	A pointer to source memory block.
NSizeType size	Size in bytes to be copied.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Remarks

Copies *size* bytes from *pDstBlock* to *pSrcBlock*.

Make sure that the destination buffer is the same size or larger than the source buffer.

Module

NMemory Module (see page 39)

8.1.1.5.1.9 NReAlloc Function

Reallocates memory block.

C++

```
NResult N_API NReAlloc(void * * ppBlock, NSizeType size);
```

Parameters

Parameters	Description
void * * ppBlock	Pointer to previously allocated memory block which should be reallocated.

NSizeType size	The size in gytes of a new memory block.

Returns

Return value	Description
N_OK (see page 32)	If function succeeds it returns N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If provided *ppBlock* value is NULL (see page 82), function fails and returns this error.
N_E_OUT_OF_MEMORY (see page 31)	If there was nor enough memory to allocate *size* bytes this error is returned.

Module

NMemory Module (see page 39)

8.1.1.5.2 Macros

8.1.1.5.2.1 NClear Macro

Clears memory block.

C++

```
#define NClear(pBlock, size) NFill(pBlock, 0, size)
```

Module

NMemory Module (see page 39)

8.1.1.5.3 Files

8.1.1.5.3.1 NMemory.h

Header file for the NMemory module. Provides memory management for Neurotechnology components.

Functions

	Name	Description
	NAlignedFree (see page 40)	Frees an aligned memory resources.
	NAlloc (see page 40)	Allocates memory block.
	NCAlloc (see page 41)	Allocates memory block with all bytes set to zero.
	NCompare (see page 41)	Compares bytes in two memory blocks.
	NCopy (see page 42)	Copies data between memory blocks.
	NFill (see page 42)	Sets bytes of memory block to specified value.
	NFree (see page 43)	Deallocates memory block.
	NMove (see page 43)	Move data from one memory block to another.
	NReAlloc (see page 43)	Reallocates memory block.

Macros

Name	Description
NClear (see page 44)	Clears memory block.

Module

NMemory Module (see page 39)

8.1.1.6 NObject Module

Provides functionality for retrieving information about specified object.

Files

Name	Description
NObject.h (see page 45)	Header file for NObject module. Provides functionality for retrieving information about specified object.

Functions

	Name	Description
	NObjectCopyParameters (see page 46)	Copies parameter values from one object to another.
	NObjectFree (see page 46)	Deletes the object specified by handle. After the object is deleted the specified handle is no longer valid.
	NObjectGetOwner (see page 46)	Gets owner of specified object.
	NObjectGetParameter (see page 47)	Retrieves value of the parameter specified by Id.
	NObjectGetParameterWithPart (see page 47)	Retrieves value of the parameter specified by Id.
	NObjectGetType (see page 48)	Retrieves type of specified object.
	NObjectReset (see page 48)	Sets default values for all parameters of the specified object.
	NObjectSetParameter (see page 48)	Sets value of the specified parameter of the specified object.
	NObjectSetParameterWithPart (see page 49)	Sets value of the specified parameter of the specified object.
	NObjectTypeOf (see page 49)	Retrieves object type.
	NTypeGetBaseType (see page 50)	Retrieves base type of specified type.
	NTypeGetName (see page 50)	Retrieves name of specified type.

8.1.1.6.1 Files

8.1.1.6.1.1 NObject.h

Header file for NObject (see page 45) module. Provides functionality for retrieving information about specified object.

Functions

	Name	Description
	NObjectCopyParameters (see page 46)	Copies parameter values from one object to another.
	NObjectFree (see page 46)	Deletes the object specified by handle. After the object is deleted the specified handle is no longer valid.
	NObjectGetOwner (see page 46)	Gets owner of specified object.
	NObjectGetParameter (see page 47)	Retrieves value of the parameter specified by Id.
	NObjectGetParameterWithPart (see page 47)	Retrieves value of the parameter specified by Id.
	NObjectGetType (see page 48)	Retrieves type of specified object.
	NObjectReset (see page 48)	Sets default values for all parameters of the specified object.

	NObjectSetParameter (see page 48)	Sets value of the specified parameter of the specified object.
	NObjectSetParameterWithPart (see page 49)	Sets value of the specified parameter of the specified object.
	NObjectTypeOf (see page 49)	Retrieves object type.
	NTypeGetBaseType (see page 50)	Retrieves base type of specified type.
	NTypeGetName (see page 50)	Retrieves name of specified type.

Module

NObject Module (see page 45)

8.1.1.6.2 Functions

8.1.1.6.2.1 NObjectCopyParameters Function

Copies parameter values from one object to another.

C++

```
NResult N_API NObjectCopyParameters(HNObject hDstObject, HNObject hSrcObject);
```

Parameters

Parameters	Description
HNObject hDstObject	[in] Destination object. Source object will be copied to this object.
HNObject hSrcObject	[in] Source object to copy.

Returns

If the function succeeds, the return value is N_OK (see page 32). Otherwise, one of the following error codes is returned.

Module

NObject Module (see page 45)

8.1.1.6.2.2 NObjectFree Function

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

C++

```
void N_API NObjectFree(HNObject hObject);
```

Parameters

Parameters	Description
HNObject hObject	in] Handle to object to be deleted.

Module

NObject Module (see page 45)

8.1.1.6.2.3 NObjectGetOwner Function

Gets owner of specified object.

C++

```
NResult N_API NObjectGetOwner(HNObject hObject, HNObject * pValue);
```

Parameters

Parameters	Description
HNObject hObject	[in] Handle to object to retrieve it's owner.
HNObject * pValue	[out] Pointer to owner object.

Returns

If the function succeeds, the return value is N_OK (see page 32). Otherwise, one of the following error codes is returned.

Module

NObject Module (see page 45)

8.1.1.6.2.4 NObjectGetParameter Function

Retrieves value of the parameter specified by Id.

C++

```
NResult N_API NObjectGetParameter(HNObject hObject, NUInt parameterId, void * pValue);
```

Parameters

Parameters	Description
HNObject hObject	[in] Handle to object to retrieve it's parameter value.
NUInt parameterId	[in] Identifier of the parameter to retrieve.
void * pValue	[out] Pointer to variable that receives parameter value.

Returns

If the function succeeds and *parameterId* specifies a N_TYPE_STRING (see page 53) type parameter, and *pValue* is NULL (see page 82), the return value is length of the string (not including the NULL (see page 82)-terminator) *pValue* should point to. If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32). Otherwise, one of the following error codes is returned.

Remarks

To learn the type of the parameter pass value obtained with NParameterMakeId (see page 53) macro using N_PC_TYPE_ID (see page 51) code and the parameter id via *parameterId* parameter and pointer to NInt (see page 67) that will receive one of N_TYPE_XXX via *pValue* parameter. *hObject* can be NULL (see page 82) in this case.

Module

NObject Module (see page 45)

8.1.1.6.2.5 NObjectGetParameterWithPart Function

Retrieves value of the parameter specified by Id.

C++

```
NResult N_API NObjectGetParameterWithPart(HNObject hObject, NUShort partId, NUInt
parameterId, void * pValue);
```

Parameters

Parameters	Description
HNObject hObject	[in] Handle to object to retrieve it's parameter value.
NUShort partId	[in] Parameter part Id.
NUInt parameterId	[in] Identifier of the parameter to retrieve.
void * pValue	[out] Pointer to variable that receives parameter value.

Returns

If the function succeeds and *parameterId* specifies a N_TYPE_STRING (see page 53) type parameter, and *pValue* is NULL (see page 82), the return value is length of the string (not including the NULL (see page 82)-terminator) *pValue* should point to. If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32). Otherwise, one of the following error codes is returned.

Remarks

To learn the type of the parameter pass value obtained with NParameterMakeId (see page 53) macro using N_PC_TYPE_ID (see page 51) code and the parameter id via *parameterId* parameter and pointer to NInt (see page 67) that will receive one of N_TYPE_XXX via *pValue* parameter. *hObject* can be NULL (see page 82) in this case.

Module

NObject Module (see page 45)

8.1.1.6.2.6 NObjectGetType Function

Retrieves type of specified object.

C++

```
NResult N_API NObjectGetType(HNObject hObject, HNType * pValue);
```

Parameters

Parameters	Description
HNObject hObject	[in] Handle to object to retrieve it's type.
HNType * pValue	[out] Object's type value.

Returns

If the function succeeds, the return value is N_OK (see page 32). Otherwise, one of the following error codes is returned.

Module

NObject Module (see page 45)

8.1.1.6.2.7 NObjectReset Function

Sets default values for all parameters of the specified object.

C++

```
NResult N_API NObjectReset(HNObject hObject);
```

Parameters

Parameters	Description
HNObject hObject	[in] Handle to object to reset values.

Returns

If the function succeeds, the return value is N_OK (see page 32). Otherwise, one of the following error codes is returned.

Module

NObject Module (see page 45)

8.1.1.6.2.8 NObjectSetParameter Function

Sets value of the specified parameter of the specified object.

C++

```
NResult N_API NObjectSetParameter(HNObject hObject, NUInt parameterId, const void * pValue);
```

Parameters

Parameters	Description
HNObject hObject	[in] Handle to the object. Can be NULL (see page 82) if setting static parameter value.
NUInt parameterId	[in] Identifier of the parameter to set.
const void * pValue	[in] Pointer to the parameter value to set.

Returns

If the function succeeds and *parameterId* specifies a N_TYPE_STRING (see page 53) type parameter, and *pValue* is NULL (see page 82), the return value is length of the string (not including the NULL (see page 82)-terminator) *pValue* should point to. If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32). Otherwise, one of the following error codes is returned.

Remarks

To learn the type of the parameter pass value obtained with NParameterMakeId (see page 53) macro using N_PC_TYPE_ID (see page 51) code and the parameter id via parameterId parameter and pointer to NInt (see page 67) that will receive one of N_TYPE_XXX via pValue parameter. hObject can be NULL (see page 82) in this case.

Module

NObject Module (see page 45)

8.1.1.6.2.9 NObjectSetParameterWithPart Function

Sets value of the specified parameter of the specified object.

C++

```
NResult N_API NObjectSetParameterWithPart(HNObject hObject, NUShort partId, NUInt parameterId, const void * pValue);
```

Parameters

Parameters	Description
HNObject hObject	[in] Handle to the object. Can be NULL (see page 82) if setting static parameter value.
NUShort partId	[in] Identifier of the part of the matcher.
NUInt parameterId	[in] Identifier of the parameter to set.
const void * pValue	[in] Pointer to the parameter value to set.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Remarks

To learn the type of the parameter pass value obtained with NParameterMakeId (see page 53) macro using N_PC_TYPE_ID (see page 51) code and the parameter id via parameterId parameter and pointer to NInt (see page 67) that will receive one of N_TYPE_XXX via pValue parameter. hObject can be NULL (see page 82) in this case.

Module

NObject Module (see page 45)

8.1.1.6.2.10 NObjectTypeOf Function

Retrieves object type.

C++

```
HNType N_API NObjectTypeOf();
```

Returns

If the function succeeds, the return value is N_OK (see page 32). Otherwise, one of the following error codes is returned.

Module

NObject Module (see page 45)

8.1.1.6.2.11 NTypeGetBaseType Function

Retrieves base type of specified type.

C++

```
NResult N_API NTypeGetBaseType(HNType hType, HNType * pValue);
```

Parameters

Parameters	Description
HNType hType	[in] Handle to type to retrieve its base type.
HNType * pValue	[out] Pointer to base type.

Returns

If the function succeeds, the return value is N_OK (see page 32). Otherwise, one of the following error codes is returned.

Module

NObject Module (see page 45)

8.1.1.6.2.12 NTypeGetName Function

Retrieves name of specified type.

C++

```
NResult N_API NTypeGetName(HNType hType, NChar * pValue);
```

Parameters

Parameters	Description
HNType hType	[in] Handle to type to retrieve it's type.
NChar * pValue	[out] Name of specified type.

Returns

If the function succeeds, the return value is N_OK (see page 32). Otherwise, one of the following error codes is returned.

Module

NObject Module (see page 45)

8.1.1.7 NParameters Module

Provides functionality for working with parameters for Neurotechnology components.

Files

Name	Description
NParameters.h (see page 54)	Header file for the NParameters module. Provides functionality for working with parameters for Neurotechnology components.

Macros

Name	Description
N_PC_TYPE_ID (see page 51)	Specifies that type id (NInt (see page 67) value, one of N_TYPE_XXX) of the parameter should be retrieved.
N_TYPE_BOOL (see page 51)	Specifies that parameter type is NBool (see page 66).
N_TYPE_BYTE (see page 51)	Specifies that parameter type is NByte (see page 66).
N_TYPE_CHAR (see page 52)	Specifies that parameter type is NChar (see page 66).
N_TYPE_DOUBLE (see page 52)	Specifies that parameter type is NDouble (see page 66).
N_TYPE_FLOAT (see page 52)	Specifies that parameter type is NFloat (see page 66).
N_TYPE_INT (see page 52)	Specifies that parameter type is NInt (see page 67).
N_TYPE_LONG (see page 52)	Specifies that parameter type is NLong (see page 68).
N_TYPE_SBYTE (see page 52)	Specifies that parameter type is NSByte (see page 68).
N_TYPE_SHORT (see page 53)	Specifies that parameter type is NShort (see page 68).
N_TYPE_STRING (see page 53)	Specifies that parameter type is null-terminated string of NChar (see page 66).
N_TYPE_UINT (see page 53)	Specifies that parameter type is NUInt (see page 69).
N_TYPE_ULONG (see page 53)	Specifies that parameter type is NULong (see page 70).
N_TYPE_USHORT (see page 53)	Specifies that parameter type is NUShort (see page 70).
NParameterMakeId (see page 53)	Makes parameter id.

8.1.1.7.1 Macros

8.1.1.7.1.1 N_PC_TYPE_ID Macro

Specifies that type id (NInt (see page 67) value, one of N_TYPE_XXX) of the parameter should be retrieved.

C++

```
#define N_PC_TYPE_ID 1
```

Module

NParameters Module (see page 50)

8.1.1.7.1.2 N_TYPE_BOOL Macro

Specifies that parameter type is NBool (see page 66).

C++

```
#define N_TYPE_BOOL 10
```

Module

NParameters Module (see page 50)

8.1.1.7.1.3 N_TYPE_BYTE Macro

Specifies that parameter type is NByte (see page 66).

C++

```
#define N_TYPE_BYTE 1
```

Module

NParameters Module (see page 50)

8.1.1.7.1.4 N_TYPE_CHAR Macro

Specifies that parameter type is NChar (see page 66).

C++

```
#define N_TYPE_CHAR 20
```

Module

NParameters Module (see page 50)

8.1.1.7.1.5 N_TYPE_DOUBLE Macro

Specifies that parameter type is NDouble (see page 66).

C++

```
#define N_TYPE_DOUBLE 31
```

Module

NParameters Module (see page 50)

8.1.1.7.1.6 N_TYPE_FLOAT Macro

Specifies that parameter type is NFloat (see page 66).

C++

```
#define N_TYPE_FLOAT 30
```

Module

NParameters Module (see page 50)

8.1.1.7.1.7 N_TYPE_INT Macro

Specifies that parameter type is NInt (see page 67).

C++

```
#define N_TYPE_INT 6
```

Module

NParameters Module (see page 50)

8.1.1.7.1.8 N_TYPE_LONG Macro

Specifies that parameter type is NLong (see page 68).

C++

```
#define N_TYPE_LONG 8
```

Module

NParameters Module (see page 50)

8.1.1.7.1.9 N_TYPE_SBYTE Macro

Specifies that parameter type is NSByte (see page 68).

C++

```
#define N_TYPE_SBYTE 2
```

Module

NParameters Module (see page 50)

8.1.1.7.1.10 N_TYPE_SHORT Macro

Specifies that parameter type is NShort (see page 68).

C++

```
#define N_TYPE_SHORT 4
```

Module

NParameters Module (see page 50)

8.1.1.7.1.11 N_TYPE_STRING Macro

Specifies that parameter type is null-terminated string of NChar (see page 66).

C++

```
#define N_TYPE_STRING 100
```

Module

NParameters Module (see page 50)

8.1.1.7.1.12 N_TYPE_UINT Macro

Specifies that parameter type is NUInt (see page 69).

C++

```
#define N_TYPE_UINT 5
```

Module

NParameters Module (see page 50)

8.1.1.7.1.13 N_TYPE_ULONG Macro

Specifies that parameter type is NULong (see page 70).

C++

```
#define N_TYPE_ULONG 7
```

Module

NParameters Module (see page 50)

8.1.1.7.1.14 N_TYPE_USHORT Macro

Specifies that parameter type is NUShort (see page 70).

C++

```
#define N_TYPE_USHORT 3
```

Module

NParameters Module (see page 50)

8.1.1.7.1.15 NParameterMakeId Macro

Makes parameter id.

C++

```
#define NParameterMakeId(code, index, id) ((NUInt)(((code) << 24) | (((index) & 0xFF) << 16) | ((id) & 0xFFFF)))
```

Parameters

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

Module

NParameters Module (see page 50)

8.1.1.7.2 Files

8.1.1.7.2.1 NParameters.h

Header file for the NParameters module. Provides functionality for working with parameters for Neurotechnology components.

Macros

Name	Description
N_PC_TYPE_ID (see page 51)	Specifies that type id (NInt (see page 67) value, one of N_TYPE_XXX) of the parameter should be retrieved.
N_TYPE_BOOL (see page 51)	Specifies that parameter type is NBool (see page 66).
N_TYPE_BYTE (see page 51)	Specifies that parameter type is NByte (see page 66).
N_TYPE_CHAR (see page 52)	Specifies that parameter type is NChar (see page 66).
N_TYPE_DOUBLE (see page 52)	Specifies that parameter type is NDouble (see page 66).
N_TYPE_FLOAT (see page 52)	Specifies that parameter type is NFloat (see page 66).
N_TYPE_INT (see page 52)	Specifies that parameter type is NInt (see page 67).
N_TYPE_LONG (see page 52)	Specifies that parameter type is NLong (see page 68).
N_TYPE_SBYTE (see page 52)	Specifies that parameter type is NSByte (see page 68).
N_TYPE_SHORT (see page 53)	Specifies that parameter type is NShort (see page 68).
N_TYPE_STRING (see page 53)	Specifies that parameter type is null-terminated string of NChar (see page 66).
N_TYPE_UINT (see page 53)	Specifies that parameter type is NUInt (see page 69).
N_TYPE_ULONG (see page 53)	Specifies that parameter type is NULong (see page 70).
N_TYPE_USHORT (see page 53)	Specifies that parameter type is NUShort (see page 70).
NParameterMakeId (see page 53)	Makes parameter id.

Module

NParameters Module (see page 50)

8.1.1.8 NProcessorInfo Module

Provides functionality for getting processor information.

Files

Name	Description
NProcessorInfo.h (see page 60)	Header file for the NProcessorInfo. Provides functionality for getting processor information.

Functions

	Name	Description
	NProcessorInfoGetModelNameA (see page 55)	Retrieves processor model name.
	NProcessorInfoGetModelNameW (see page 56)	Retrieves processor model name.
	NProcessorInfoGetVendor (see page 56)	Retrieves a pointer to enumeration that contains information about processor's vendor.
	NProcessorInfoGetVendorNameA (see page 57)	Retrieves processor's vendor name.
	NProcessorInfoGetVendorNameW (see page 57)	Retrieves processor's vendor name.
	NProcessorInfoIs3DNowSupported (see page 58)	
	NProcessorInfoIsMmxSupported (see page 58)	
	NProcessorInfoIsSse2Supported (see page 58)	
	NProcessorInfoIsSse3Supported (see page 58)	
	NProcessorInfoIsSseSupported (see page 59)	

Structs, Records, Enums

	Name	Description
	NProcessorVendor_ (see page 59)	Enumerates different processor vendors.
	NProcessorVendor (see page 59)	Enumerates different processor vendors.

8.1.1.8.1 Functions

8.1.1.8.1.1 NProcessorInfoGetModelNameA Function

Retrieves processor model name.

C++

```
NResult N_API NProcessorInfoGetModelNameA(NAChar * szValue);
```

Parameters

Parameters	Description
NAChar * szValue	[out] A pointer to memory block that contains processor model name.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	szValue is NULL (see page 82).

Notes

This function is used if compiling for platform without ANSI versions of the functions support. If can not choose between functions NProcessorInfoGetModelNameA andNProcessorInfoGetModelNameW, you can use function without suffix A or W.

Module

NProcessorInfo Module (see page 54)

8.1.1.8.1.2 NProcessorInfoGetModelNameW Function

Retrieves processor model name.

C++

```
NResult N_API NProcessorInfoGetModelNameW(NWChar * szValue);
```

Parameters

Parameters	Description
NWChar * szValue	[out] A pointer to memory block that contains processor model name.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	szValue is NULL (see page 82).

Notes

This function is used if compiling for platforms without Unicode support. If can not choose between functions NProcessorInfoGetModelNameA (see page 55) andNProcessorInfoGetModelNameW, you can use function without suffix A or W.

Module

NProcessorInfo Module (see page 54)

8.1.1.8.1.3 NProcessorInfoGetVendor Function

Retrieves a pointer to enumeration that contains information about processor's vendor.

C++

```
NResult N_API NProcessorInfoGetVendor(NProcessorVendor * pValue);
```

Parameters

Parameters	Description
NProcessorVendor * pValue	[out] A pointer to memory block that contains NProcessorVendor (see page 59) enumeration.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	pValue is NULL (see page 82).

Module

NProcessorInfo Module (see page 54)

8.1.1.8.1.4 NProcessorInfoGetVendorNameA Function

Retrieves processor's vendor name.

C++

```
NResult N_API NProcessorInfoGetVendorNameA(NAChar * szValue);
```

Parameters

Parameters	Description
NAChar * szValue	[out] A pointer to memory block that contains processor's vendor name.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	szValue is NULL (see page 82).

Notes

This function is used if compiling for platform without ANSI versions of the functions support. If can not choose between functions NProcessorInfoGetVendorNameA NProcessorInfoGetVendorNameW (see page 57), you can use function without suffix A or W.

Module

NProcessorInfo Module (see page 54)

8.1.1.8.1.5 NProcessorInfoGetVendorNameW Function

Retrieves processor's vendor name.

C++

```
NResult N_API NProcessorInfoGetVendorNameW(NWChar * szValue);
```

Parameters

Parameters	Description
NWChar * szValue	[out] A pointer to memory block that contains processor's vendor name.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	szValue is NULL (see page 82).

Notes

This function is used if compiling for platforms without Unicode support. If can not choose between functions NProcessorInfoGetVendorNameA (see page 57) NProcessorInfoGetVendorNameW, you can use function without suffix A or W.

Module

NProcessorInfo Module (see page 54)

8.1.1.8.1.6 NProcessorInfoIs3DNowSupported Function

C++

```
NBool N_API NProcessorInfoIs3DNowSupported();
```

Returns

Returns a boolean value which indicates 3D support for the particular processor model.

Module

NProcessorInfo Module (see page 54)

8.1.1.8.1.7 NProcessorInfoIsMmxSupported Function

C++

```
NBool N_API NProcessorInfoIsMmxSupported();
```

Returns

Returns a boolean value which indicates if a single instruction set (Mmx) is supported for the particular processor model.

Module

NProcessorInfo Module (see page 54)

8.1.1.8.1.8 NProcessorInfoIsSse2Supported Function

C++

```
NBool N_API NProcessorInfoIsSse2Supported();
```

Returns

Returns a boolean value which indicates if a Streaming SIMD Extension 2 (SSE2) instruction set is supported for a particular processor model.

Module

NProcessorInfo Module (see page 54)

8.1.1.8.1.9 NProcessorInfoIsSse3Supported Function

C++

```
NBool N_API NProcessorInfoIsSse3Supported();
```

Returns

Returns a boolean value which indicates if a Streaming SIMD Extension 3 (SSE3) instruction set is supported for a particular processor model.

Module

NProcessorInfo Module (see page 54)

8.1.1.8.1.10 NProcessorInfoIsSseSupported Function

C++

```
NBool N_API NProcessorInfoIsSseSupported();
```

Returns

Returns a boolean value which indicates if a Streaming SIMD (SSE) instruction set is supported for a particular processor model.

Module

NProcessorInfo Module (see page 54)

8.1.1.8.2 Structs, Records, Enums

8.1.1.8.2.1 NProcessorVendor Enumeration

Enumerates different processor vendors.

C++

```
typedef enum NProcessorVendor_ {
  npvUnknown = 0,
  npvAmd = 1,
  npvCentaur = 2,
  npvCyrix = 3,
  npvIntel = 4,
  npvNationalSemiconductor = 5,
  npvNexGen = 6,
  npvRiseTechnology = 7,
  npvSiS = 8,
  npvTransmeta = 9,
  npvUmc = 10
} NProcessorVendor;
```

Members

Members	Description
npvUnknown = 0	Vendor is unknown.
npvAmd = 1	Advanced Micro Devices, Inc. (AMD)
npvCentaur = 2	Centaur Technology, Inc.
npvCyrix = 3	Cyrix, Inc.
npvIntel = 4	Intel Corporation.
npvNationalSemiconductor = 5	National Semiconductor Corporation.
npvNexGen = 6	NexGen Technologies, Inc.
npvRiseTechnology = 7	Rise Technology, Inc.
npvSiS = 8	Silicon Integrated Systems Corp.
npvTransmeta = 9	Transmeta Corp.
npvUmc = 10	United Microelectronics Corporation.

Module

NProcessorInfo Module (see page 54)

8.1.1.8.3 Files

8.1.1.8.3.1 NProcessorInfo.h

Header file for the NProcessorInfo. Provides functionality for getting processor information.

Enumerations

	Name	Description
	NProcessorVendor_ (see page 59)	Enumerates different processor vendors.
	NProcessorVendor (see page 59)	Enumerates different processor vendors.

Functions

	Name	Description
	NProcessorInfoGetModelNameA (see page 55)	Retrieves processor model name.
	NProcessorInfoGetModelNameW (see page 56)	Retrieves processor model name.
	NProcessorInfoGetVendor (see page 56)	Retrieves a pointer to enumeration that contains information about processor's vendor.
	NProcessorInfoGetVendorNameA (see page 57)	Retrieves processor's vendor name.
	NProcessorInfoGetVendorNameW (see page 57)	Retrieves processor's vendor name.
	NProcessorInfoIs3DNowSupported (see page 58)	
	NProcessorInfoIsMmxSupported (see page 58)	
	NProcessorInfoIsSse2Supported (see page 58)	
	NProcessorInfoIsSse3Supported (see page 58)	
	NProcessorInfoIsSseSupported (see page 59)	

Module

NProcessorInfo Module (see page 54)

8.1.1.9 NStream Module

Supports internal Neurotechnology libraries infrastructure and should not be used directly in your code.

Macros

Name	Description
NIsReverseByteOrder (see page 61)	Checks if specified byte order is reverse to system byte order.

Structs, Records, Enums

	Name	Description
	NByteOrder_ (see page 61)	Specifies byte order.
	NFileAccess_ (see page 61)	Specifies access to a file.
	NByteOrder (see page 61)	Specifies byte order.
	NFileAccess (see page 61)	Specifies access to a file.

8.1.1.9.1 Structs, Records, Enums

8.1.1.9.1.1 NByteOrder Enumeration

Specifies byte order.

C++

```
typedef enum NByteOrder_ {
  nboLittleEndian = 0,
  nboBigEndian = 1,
  nboSystem = nboLittleEndian
} NByteOrder;
```

Members

Members	Description
nboLittleEndian = 0	Little-endian byte order.
nboBigEndian = 1	Big-endian byte order.
nboSystem = nboLittleEndian	System-dependent byte order (either little-endian or bigendian).

Module

NStream Module (see page 60)

8.1.1.9.1.2 NFileAccess Enumeration

Specifies access to a file.

C++

```
typedef enum NFileAccess_ {
  nfaRead = 1,
  nfaWrite = 2,
  nfaReadWrite = nfaRead|nfaWrite
} NFileAccess;
```

Members

Members	Description
nfaRead = 1	Read access to the file.
nfaWrite = 2	Write access to the file.
nfaReadWrite = nfaRead|nfaWrite	Read and write access to the file.

Module

NStream Module (see page 60)

8.1.1.9.2 Macros

8.1.1.9.2.1 NIsReverseByteOrder Macro

Checks if specified byte order is reverse to system byte order.

C++

```
#define NIsReverseByteOrder(byteOrder) ((byteOrder) != nboSystem)
```

Module

NStream Module (see page 60)

8.1.1.10 NTypes Module

Defines types and macros used in Neurotechnology components.

Files

Name	Description
NTypes.h (see page 83)	Header file for NTypes module. Defines types and macros used in Neurotechnology components.

Macros

Name	Description
N_64 (see page 70)	Defined if compiling for 64-bit architecture.
N_ANSI_C (see page 71)	Defined if ANSI C language compliance is enabled in compiler.
N_BIG_ENDIAN (see page 71)	Defined if compiling for big-endian processor architecture.
N_BYTE_MAX (see page 71)	Maximum value for NByte (see page 66).
N_BYTE_MIN (see page 71)	Minimum value for NByte (see page 66).
N_CALLBACK_AW (see page 71)	Picks either ANSI or Unicode (if N_UNICODE (see page 81) is defined) version of the callback (with either 'A' or 'W' suffix accordingly).
N_CPP (see page 71)	Defined if compiling as C++ code.
N_DEBUG (see page 72)	Defined if compiling in debug mode.
N_DECLARE_HANDLE (see page 72)	Declares handle with specified name.
N_DOUBLE_EPSILON (see page 72)	Epsilon value for NDouble (see page 66).
N_DOUBLE_MAX (see page 72)	Maximum value for NDouble (see page 66).
N_DOUBLE_MIN (see page 72)	Minimum value for NDouble (see page 66).
N_FAST_FLOAT (see page 72)	Defined if CPU has floating-point instructions.
N_FLOAT_EPSILON (see page 73)	Epsilon value for NFloat (see page 66).
N_FLOAT_MAX (see page 73)	Maximum value for NFloat (see page 66).
N_FLOAT_MIN (see page 73)	Minimum value for NFloat (see page 66).
N_FUNC_AW (see page 73)	Picks either ANSI or Unicode (if N_UNICODE (see page 81) is defined) version of the function (with either 'A' or 'W' suffix accordingly).
N_GCC (see page 73)	Defined if compiling with GCC.
N_INT_MAX (see page 74)	Maximum value for NInt (see page 67).
N_INT_MIN (see page 74)	Minimum value for NInt (see page 67).
N_INT16_MAX (see page 74)	Maximum value for NShort (see page 68).
N_INT16_MIN (see page 74)	Minimum value for NShort (see page 68).
N_INT32_MAX (see page 74)	Maximum value for NInt (see page 67).
N_INT32_MIN (see page 74)	Minimum value for NInt (see page 67).
N_INT64_MAX (see page 75)	Maximum value for NInt64 (see page 67).
N_INT64_MIN (see page 75)	Minimum value for NInt64 (see page 67).
N_INT8_MAX (see page 75)	Maximum value for NSByte (see page 68).
N_INT8_MIN (see page 75)	Minimum value for NSByte (see page 68).
N_LIB (see page 75)	Defined if compiling static library.
N_LINUX (see page 75)	Defined if compiling for Linux.
N_LONG_MAX (see page 76)	Maximum value for NLong (see page 68).
N_LONG_MIN (see page 76)	Minimum value for NLong (see page 68).
N_MAC (see page 76)	Defined if compiling for Mac OS.
N_MSVC (see page 76)	Defined if compiling with Microsoft Visual C++.

N_NO_ANSI_FUNC (see page 76)	Defined if compiling for platform without ANSI versions of the functions support.
N_NO_INT_64 (see page 77)	Defined if compiling for platform without 64-bit integer types support.
N_PACKED (see page 77)	Struct packing/alignment.
N_POS_TYPE_MAX (see page 77)	Maximum value for NPosType (see page 68).
N_POS_TYPE_MIN (see page 77)	Minimum value for NPosType (see page 68).
N_SBYTE_MAX (see page 77)	Maximum value for NSByte (see page 68).
N_SBYTE_MIN (see page 77)	Minimum value for NSByte (see page 68).
N_SHORT_MAX (see page 78)	Maximum value for NShort (see page 68).
N_SHORT_MIN (see page 78)	Minimum value for NShort (see page 68).
N_SINGLE_EPSILON (see page 78)	Epsilon value for NSingle (see page 69).
N_SINGLE_MAX (see page 78)	Maximum value for NSingle (see page 69).
N_SINGLE_MIN (see page 78)	Minimum value for NSingle (see page 69).
N_SIZE_TYPE_MAX (see page 78)	Maximum value for NSizeType (see page 69).
N_SIZE_TYPE_MIN (see page 79)	Minimum value for NSizeType (see page 69).
N_STRUCT_AW (see page 79)	Picks either ANSI or Unicode (if N_UNICODE (see page 81) is defined) version of the struct (with either 'A' or 'W' suffix accordingly)
N_T (see page 79)	Makes either ANSI or Unicode (if N_UNICODE (see page 81) is defined) string or character constant.
N_UINT_MAX (see page 79)	Maximum value for NUInt (see page 69).
N_UINT_MIN (see page 79)	Minimum value for NUInt (see page 69).
N_UINT16_MAX (see page 80)	Maximum value for NUShort (see page 70).
N_UINT16_MIN (see page 80)	Minimum value for NUShort (see page 70).
N_UINT32_MAX (see page 80)	Maximum value for NUInt (see page 69).
N_UINT32_MIN (see page 80)	Minimum value for NUInt (see page 69).
N_UINT64_MAX (see page 80)	Maximum value for NULong (see page 70).
N_UINT64_MIN (see page 80)	Minimum value for NULong (see page 70).
N_UINT8_MAX (see page 81)	Maximum value for NUInt8 (see page 70).
N_UINT8_MIN (see page 81)	Minimum value for NUInt8 (see page 70).
N_ULONG_MAX (see page 81)	Maximum value for NULong (see page 70).
N_ULONG_MIN (see page 81)	Minimum value for NULong (see page 70).
N_UNICODE (see page 81)	Defined if compiling with Unicode character set (affects NChar (see page 66) type).
N_USHORT_MAX (see page 81)	Maximum value for NUShort (see page 70).
N_USHORT_MIN (see page 82)	Minimum value for NUShort (see page 70).
N_WINDOWS (see page 82)	Defined if compiling for Windows.
NFalse (see page 82)	False value for NBoolean (see page 66).
NTrue (see page 82)	True value for NBoolean (see page 66).
NULL (see page 82)	Null value for pointer.

Structs, Records, Enums

	Name	Description
	NIndexPair_ (see page 64)	Represents a pair of indexes.
	NRational_ (see page 65)	Represents a signed rational number.
	NURational_ (see page 65)	Represents an unsigned rational number.
	NIndexPair (see page 64)	Represents a pair of indexes.
	NRational (see page 65)	Represents a signed rational number.
	NURational (see page 65)	Represents an unsigned rational number.

Types

Name	Description
NAChar (see page 65)	ANSI character (8-bit).
NBool (see page 66)	Same as NBoolean (see page 66).
NBoolean (see page 66)	32-bit boolean value. See also NTrue (see page 82) and NFalse (see page 82).
NByte (see page 66)	Same as NUInt8 (see page 70).
NChar (see page 66)	Character type. Either NAChar (see page 65) or NWChar (see page 70) (if N_UNICODE (see page 81) is defined).
NDouble (see page 66)	Double precision floating point number.
NFloat (see page 66)	Same as NSingle (see page 69).
NHandle (see page 67)	Pointer to unspecified data (same as void *).
NInt (see page 67)	Same as NInt32 (see page 67).
NInt16 (see page 67)	16-bit signed integer (short).
NInt32 (see page 67)	32-bit signed integer (int).
NInt64 (see page 67)	64-bit signed integer (long). Not available on some 32-bit platforms.
NInt8 (see page 67)	8-bit signed integer (signed byte).
NLong (see page 68)	Same as NInt64 (see page 67).
NPosType (see page 68)	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 (see page 68)	Result of a function (same as NInt (see page 67)). See also NErrors module.
NSByte (see page 68)	Same as NInt8 (see page 67).
NShort (see page 68)	Same as NInt16 (see page 67).
NSingle (see page 69)	Single precision floating point number.
NSizeType (see page 69)	Platform dependent size type. Unsigned 32-bit integer on 32-bit platform, unsigned 64-bit integer on 64-bit platform.
NUInt (see page 69)	Same as NUInt32 (see page 69).
NUInt16 (see page 69)	16-bit unsigned integer (unsigned short).
NUInt32 (see page 69)	32-bit unsigned integer (unsigned int).
NUInt64 (see page 69)	64-bit unsigned integer (unsigned long). Not available on some 32-bit platforms.
NUInt8 (see page 70)	8-bit unsigned integer (byte).
NULong (see page 70)	Same as NUInt64 (see page 69).
NUShort (see page 70)	Same as NUInt16 (see page 69).
NWChar (see page 70)	Unicode character (16-bit).

8.1.1.10.1 Structs, Records, Enums

8.1.1.10.1.1 NIndexPair Structure

Represents a pair of indexes.

C++

```
typedef struct NIndexPair_ {
  NInt Index1;
  NInt Index2;
} NIndexPair;
```

Members

Members	Description
NInt Index1;	First index of this NIndexPair.
NInt Index2;	Second index of this NIndexPair.

Module

NTypes Module (see page 62)

8.1.1.10.1.2 NRational Structure

Represents a signed rational number.

C++

```
typedef struct NRational_ {
  NInt Numerator;
  NInt Denominator;
} NRational;
```

Members

Members	Description
NInt Numerator;	Numerator of this NRational.
NInt Denominator;	Denominator of this NRational.

Module

NTypes Module (see page 62)

8.1.1.10.1.3 NURational Structure

Represents an unsigned rational number.

C++

```
typedef struct NURational_ {
  NUInt Numerator;
  NUInt Denominator;
} NURational;
```

Members

Members	Description
NUInt Numerator;	Numerator of this NURational.
NUInt Denominator;	Denominator of this NURational.

Module

NTypes Module (see page 62)

8.1.1.10.2 Types

8.1.1.10.2.1 NAChar Type

ANSI character (8-bit).

C++

```
typedef char NAChar;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.2 NBool Type

Same as NBoolean (see page 66).

C++

```
typedef NBoolean NBool;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.3 NBoolean Type

32-bit boolean value. See also NTrue (see page 82) and NFalse (see page 82).

C++

```
typedef int NBoolean;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.4 NByte Type

Same as NUInt8 (see page 70).

C++

```
typedef NUInt8 NByte;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.5 NChar Type

Character type. Either NAChar (see page 65) or NWChar (see page 70) (if N_UNICODE (see page 81) is defined).

C++

```
typedef NAChar NChar;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.6 NDouble Type

Double precision floating point number.

C++

```
typedef double NDouble;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.7 NFloat Type

Same as NSingle (see page 69).

C++

```
typedef NSingle NFloat;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.8 NHandle Type

Pointer to unspecified data (same as void *).

C++

```
typedef void * NHandle;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.9 NInt Type

Same as NInt32 (see page 67).

C++

```
typedef NInt32 NInt;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.10 NInt16 Type

16-bit signed integer (short).

C++

```
typedef signed short NInt16;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.11 NInt32 Type

32-bit signed integer (int).

C++

```
typedef signed int NInt32;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.12 NInt64 Type

64-bit signed integer (long). Not available on some 32-bit platforms.

C++

```
typedef signed long long NInt64;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.13 NInt8 Type

8-bit signed integer (signed byte).

C++

```
typedef signed char NInt8;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.14 NLong Type

Same as NInt64 (see page 67).

C++

```
typedef NInt64 NLong;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.15 NPosType Type

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).

C++

```
typedef NInt64 NPosType;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.16 NResult Type

Result of a function (same as NInt (see page 67)). See also NErrors module.

C++

```
typedef NInt NResult;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.17 NSByte Type

Same as NInt8 (see page 67).

C++

```
typedef NInt8 NSByte;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.18 NShort Type

Same as NInt16 (see page 67).

C++

```
typedef NInt16 NShort;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.19 NSingle Type

Single precision floating point number.

C++

```
typedef float NSingle;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.20 NSizeType Type

Platform dependent size type. Unsigned 32-bit integer on 32-bit platform, unsigned 64-bit integer on 64-bit platform.

C++

```
typedef NUInt32 NSizeType;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.21 NUInt Type

Same as NUInt32 (see page 69).

C++

```
typedef NUInt32 NUInt;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.22 NUInt16 Type

16-bit unsigned integer (unsigned short).

C++

```
typedef unsigned short NUInt16;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.23 NUInt32 Type

32-bit unsigned integer (unsigned int).

C++

```
typedef unsigned int NUInt32;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.24 NUInt64 Type

64-bit unsigned integer (unsigned long). Not available on some 32-bit platforms.

C++

```
typedef unsigned long long NUInt64;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.25 NUInt8 Type

8-bit unsigned integer (byte).

C++

```
typedef unsigned char NUInt8;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.26 NULong Type

Same as NUInt64 (see page 69).

C++

```
typedef NUInt64 NULong;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.27 NUShort Type

Same as NUInt16 (see page 69).

C++

```
typedef NUInt16 NUShort;
```

Module

NTypes Module (see page 62)

8.1.1.10.2.28 NWChar Type

Unicode character (16-bit).

C++

```
typedef int NWChar;
```

Module

NTypes Module (see page 62)

8.1.1.10.3 Macros

8.1.1.10.3.1 N_64 Macro

Defined if compiling for 64-bit architecture.

C++

```
#define N_64
```

Module

NTypes Module (see page 62)

8.1.1.10.3.2 N_ANSI_C Macro

Defined if ANSI C language compliance is enabled in compiler.

C++

```
#define N_ANSI_C
```

Module

NTypes Module (see page 62)

8.1.1.10.3.3 N_BIG_ENDIAN Macro

Defined if compiling for big-endian processor architecture.

C++

```
#define N_BIG_ENDIAN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.4 N_BYTE_MAX Macro

Maximum value for NByte (see page 66).

C++

```
#define N_BYTE_MAX N_UINT8_MAX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.5 N_BYTE_MIN Macro

Minimum value for NByte (see page 66).

C++

```
#define N_BYTE_MIN N_UINT8_MIN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.6 N_CALLBACK_AW Macro

Picks either ANSI or Unicode (if N_UNICODE (see page 81) is defined) version of the callback (with either 'A' or 'W' suffix accordingly).

C++

```
#define N_CALLBACK_AW(name) name##A
```

Module

NTypes Module (see page 62)

8.1.1.10.3.7 N_CPP Macro

Defined if compiling as C++ code.

C++

```
#define N_CPP
```

Module

NTypes Module (see page 62)

8.1.1.10.3.8 N_DEBUG Macro

Defined if compiling in debug mode.

C++

```
#define N_DEBUG
```

Module

NTypes Module (see page 62)

8.1.1.10.3.9 N_DECLARE_HANDLE Macro

Declares handle with specified name.

C++

```
#define N_DECLARE_HANDLE(name) typedef struct name##_ { int unused; } * name;
```

Module

NTypes Module (see page 62)

8.1.1.10.3.10 N_DOUBLE_EPSILON Macro

Epsilon value for NDouble (see page 66).

C++

```
#define N_DOUBLE_EPSILON 2.2204460492503131e-016
```

Module

NTypes Module (see page 62)

8.1.1.10.3.11 N_DOUBLE_MAX Macro

Maximum value for NDouble (see page 66).

C++

```
#define N_DOUBLE_MAX 1.7976931348623158e+308
```

Module

NTypes Module (see page 62)

8.1.1.10.3.12 N_DOUBLE_MIN Macro

Minimum value for NDouble (see page 66).

C++

```
#define N_DOUBLE_MIN 2.2250738585072014e-308
```

Module

NTypes Module (see page 62)

8.1.1.10.3.13 N_FAST_FLOAT Macro

Defined if CPU has floating-point instructions.

C++

```
#define N_FAST_FLOAT
```

Module

NTypes Module (see page 62)

8.1.1.10.3.14 N_FLOAT_EPSILON Macro

Epsilon value for NFloat (see page 66).

C++

```
#define N_FLOAT_EPSILON N_SINGLE_EPSILON
```

Module

NTypes Module (see page 62)

8.1.1.10.3.15 N_FLOAT_MAX Macro

Maximum value for NFloat (see page 66).

C++

```
#define N_FLOAT_MAX N_SINGLE_MAX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.16 N_FLOAT_MIN Macro

Minimum value for NFloat (see page 66).

C++

```
#define N_FLOAT_MIN N_SINGLE_MIN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.17 N_FUNC_AW Macro

Picks either ANSI or Unicode (if N_UNICODE (see page 81) is defined) version of the function (with either 'A' or 'W' suffix accordingly).

C++

```
#define N_FUNC_AW(name) name##A
```

Module

NTypes Module (see page 62)

8.1.1.10.3.18 N_GCC Macro

Defined if compiling with GCC.

C++

```
#define N_GCC
```

Module

NTypes Module (see page 62)

8.1.1.10.3.19 N_INT_MAX Macro

Maximum value for NInt (see page 67).

C++

```
#define N_INT_MAX N_INT32_MAX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.20 N_INT_MIN Macro

Minimum value for NInt (see page 67).

C++

```
#define N_INT_MIN N_INT32_MIN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.21 N_INT16_MAX Macro

Maximum value for NShort (see page 68).

C++

```
#define N_INT16_MAX ((NInt16)0x7FFF)
```

Module

NTypes Module (see page 62)

8.1.1.10.3.22 N_INT16_MIN Macro

Minimum value for NShort (see page 68).

C++

```
#define N_INT16_MIN ((NInt16)0x8000)
```

Module

NTypes Module (see page 62)

8.1.1.10.3.23 N_INT32_MAX Macro

Maximum value for NInt (see page 67).

C++

```
#define N_INT32_MAX 0x7FFFFFFF
```

Module

NTypes Module (see page 62)

8.1.1.10.3.24 N_INT32_MIN Macro

Minimum value for NInt (see page 67).

C++

```
#define N_INT32_MIN 0x80000000
```

Module

NTypes Module (see page 62)

8.1.1.10.3.25 N_INT64_MAX Macro

Maximum value for NInt64 (see page 67).

C++

```
#define N_INT64_MAX 0x7FFFFFFFFFFFFFFFll
```

Module

NTypes Module (see page 62)

8.1.1.10.3.26 N_INT64_MIN Macro

Minimum value for NInt64 (see page 67).

C++

```
#define N_INT64_MIN 0x8000000000000000ll
```

Module

NTypes Module (see page 62)

8.1.1.10.3.27 N_INT8_MAX Macro

Maximum value for NSByte (see page 68).

C++

```
#define N_INT8_MAX ((NInt8)0x7F)
```

Module

NTypes Module (see page 62)

8.1.1.10.3.28 N_INT8_MIN Macro

Minimum value for NSByte (see page 68).

C++

```
#define N_INT8_MIN ((NInt8)0x80)
```

Module

NTypes Module (see page 62)

8.1.1.10.3.29 N_LIB Macro

Defined if compiling static library.

C++

```
#define N_LIB
```

Module

NTypes Module (see page 62)

8.1.1.10.3.30 N_LINUX Macro

Defined if compiling for Linux.

C++

```
#define N_LINUX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.31 N_LONG_MAX Macro

Maximum value for NLong (see page 68).

C++

```
#define N_LONG_MAX N_INT64_MAX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.32 N_LONG_MIN Macro

Minimum value for NLong (see page 68).

C++

```
#define N_LONG_MIN N_INT64_MIN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.33 N_MAC Macro

Defined if compiling for Mac OS.

C++

```
#define N_MAC
```

Module

NTypes Module (see page 62)

8.1.1.10.3.34 N_MSVC Macro

Defined if compiling with Microsoft Visual C++.

C++

```
#define N_MSVC
```

Module

NTypes Module (see page 62)

8.1.1.10.3.35 N_NO_ANSI_FUNC Macro

Defined if compiling for platform without ANSI versions of the functions support.

C++

```
#define N_NO_ANSI_FUNC
```

Module

NTypes Module (see page 62)

8.1.1.10.3.36 N_NO_INT_64 Macro

Defined if compiling for platform without 64-bit integer types support.

C++

```
#define N_NO_INT_64
```

Module

NTypes Module (see page 62)

8.1.1.10.3.37 N_PACKED Macro

Struct packing/alignment.

C++

```
#define N_PACKED
```

Module

NTypes Module (see page 62)

8.1.1.10.3.38 N_POS_TYPE_MAX Macro

Maximum value for NPosType (see page 68).

C++

```
#define N_POS_TYPE_MAX N_INT64_MAX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.39 N_POS_TYPE_MIN Macro

Minimum value for NPosType (see page 68).

C++

```
#define N_POS_TYPE_MIN N_INT64_MIN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.40 N_SBYTE_MAX Macro

Maximum value for NSByte (see page 68).

C++

```
#define N_SBYTE_MAX N_INT8_MAX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.41 N_SBYTE_MIN Macro

Minimum value for NSByte (see page 68).

C++

```
#define N_SBYTE_MIN N_INT8_MIN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.42 N_SHORT_MAX Macro

Maximum value for NShort (see page 68).

C++

```
#define N_SHORT_MAX N_INT16_MAX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.43 N_SHORT_MIN Macro

Minimum value for NShort (see page 68).

C++

```
#define N_SHORT_MIN N_INT16_MIN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.44 N_SINGLE_EPSILON Macro

Epsilon value for NSingle (see page 69).

C++

```
#define N_SINGLE_EPSILON 1.192092896e-07F
```

Module

NTypes Module (see page 62)

8.1.1.10.3.45 N_SINGLE_MAX Macro

Maximum value for NSingle (see page 69).

C++

```
#define N_SINGLE_MAX 3.402823466e+38F
```

Module

NTypes Module (see page 62)

8.1.1.10.3.46 N_SINGLE_MIN Macro

Minimum value for NSingle (see page 69).

C++

```
#define N_SINGLE_MIN 1.175494351e-38F
```

Module

NTypes Module (see page 62)

8.1.1.10.3.47 N_SIZE_TYPE_MAX Macro

Maximum value for NSizeType (see page 69).

C++

```
#define N_SIZE_TYPE_MAX N_UINT32_MAX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.48 N_SIZE_TYPE_MIN Macro

Minimum value for NSizeType (see page 69).

C++

```
#define N_SIZE_TYPE_MIN N_UINT32_MIN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.49 N_STRUCT_AW Macro

Picks either ANSI or Unicode (if N_UNICODE (see page 81) is defined) version of the struct (with either 'A' or 'W' suffix accordingly)

C++

```
#define N_STRUCT_AW(name) name##A
```

Module

NTypes Module (see page 62)

8.1.1.10.3.50 N_T Macro

Makes either ANSI or Unicode (if N_UNICODE (see page 81) is defined) string or character constant.

C++

```
#define N_T(text) N_T_(text)
```

Module

NTypes Module (see page 62)

8.1.1.10.3.51 N_UINT_MAX Macro

Maximum value for NUInt (see page 69).

C++

```
#define N_UINT_MAX N_UINT32_MAX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.52 N_UINT_MIN Macro

Minimum value for NUInt (see page 69).

C++

```
#define N_UINT_MIN N_UINT32_MIN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.53 N_UINT16_MAX Macro

Maximum value for NUShort (see page 70).

C++

```
#define N_UINT16_MAX ((NUInt16)0xFFFFu)
```

Module

NTypes Module (see page 62)

8.1.1.10.3.54 N_UINT16_MIN Macro

Minimum value for NUShort (see page 70).

C++

```
#define N_UINT16_MIN ((NUInt16)0x0000u)
```

Module

NTypes Module (see page 62)

8.1.1.10.3.55 N_UINT32_MAX Macro

Maximum value for NUInt (see page 69).

C++

```
#define N_UINT32_MAX 0xFFFFFFFFu
```

Module

NTypes Module (see page 62)

8.1.1.10.3.56 N_UINT32_MIN Macro

Minimum value for NUInt (see page 69).

C++

```
#define N_UINT32_MIN 0x00000000u
```

Module

NTypes Module (see page 62)

8.1.1.10.3.57 N_UINT64_MAX Macro

Maximum value for NULong (see page 70).

C++

```
#define N_UINT64_MAX 0xFFFFFFFFFFFFFFFFull
```

Module

NTypes Module (see page 62)

8.1.1.10.3.58 N_UINT64_MIN Macro

Minimum value for NULong (see page 70).

C++

```
#define N_UINT64_MIN 0x0000000000000000ull
```

Module

NTypes Module (see page 62)

8.1.1.10.3.59 N_UINT8_MAX Macro

Maximum value for NUInt8 (see page 70).

C++

```
#define N_UINT8_MAX ((NUInt8)0xFFu)
```

Module

NTypes Module (see page 62)

8.1.1.10.3.60 N_UINT8_MIN Macro

Minimum value for NUInt8 (see page 70).

C++

```
#define N_UINT8_MIN ((NUInt8)0x00u)
```

Module

NTypes Module (see page 62)

8.1.1.10.3.61 N_ULONG_MAX Macro

Maximum value for NULong (see page 70).

C++

```
#define N_ULONG_MAX N_UINT64_MAX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.62 N_ULONG_MIN Macro

Minimum value for NULong (see page 70).

C++

```
#define N_ULONG_MIN N_UINT64_MIN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.63 N_UNICODE Macro

Defined if compiling with Unicode character set (affects NChar (see page 66) type).

C++

```
#define N_UNICODE
```

Module

NTypes Module (see page 62)

8.1.1.10.3.64 N_USHORT_MAX Macro

Maximum value for NUShort (see page 70).

C++

```
#define N_USHORT_MAX N_UINT16_MAX
```

Module

NTypes Module (see page 62)

8.1.1.10.3.65 N_USHORT_MIN Macro

Minimum value for NUShort (see page 70).

C++

```
#define N_USHORT_MIN N_UINT16_MIN
```

Module

NTypes Module (see page 62)

8.1.1.10.3.66 N_WINDOWS Macro

Defined if compiling for Windows.

C++

```
#define N_WINDOWS
```

Module

NTypes Module (see page 62)

8.1.1.10.3.67 NFalse Macro

False value for NBoolean (see page 66).

C++

```
#define NFalse 0
```

Module

NTypes Module (see page 62)

8.1.1.10.3.68 NTrue Macro

True value for NBoolean (see page 66).

C++

```
#define NTrue 1
```

Module

NTypes Module (see page 62)

8.1.1.10.3.69 NULL Macro

Null value for pointer.

C++

```
#define NULL 0
```

Module

NTypes Module (see page 62)

8.1.1.10.4 Files

8.1.1.10.4.1 NTypes.h

Header file for NTypes module. Defines types and macros used in Neurotechnology components.

Macros

Name	Description
N_64 (see page 70)	Defined if compiling for 64-bit architecture.
N_ANSI_C (see page 71)	Defined if ANSI C language compliance is enabled in compiler.
N_BIG_ENDIAN (see page 71)	Defined if compiling for big-endian processor architecture.
N_BYTE_MAX (see page 71)	Maximum value for NByte (see page 66).
N_BYTE_MIN (see page 71)	Minimum value for NByte (see page 66).
N_CALLBACK_AW (see page 71)	Picks either ANSI or Unicode (if N_UNICODE (see page 81) is defined) version of the callback (with either 'A' or 'W' suffix accordingly).
N_CPP (see page 71)	Defined if compiling as C++ code.
N_DEBUG (see page 72)	Defined if compiling in debug mode.
N_DECLARE_HANDLE (see page 72)	Declares handle with specified name.
N_DOUBLE_EPSILON (see page 72)	Epsilon value for NDouble (see page 66).
N_DOUBLE_MAX (see page 72)	Maximum value for NDouble (see page 66).
N_DOUBLE_MIN (see page 72)	Minimum value for NDouble (see page 66).
N_FAST_FLOAT (see page 72)	Defined if CPU has floating-point instructions.
N_FLOAT_EPSILON (see page 73)	Epsilon value for NFloat (see page 66).
N_FLOAT_MAX (see page 73)	Maximum value for NFloat (see page 66).
N_FLOAT_MIN (see page 73)	Minimum value for NFloat (see page 66).
N_FUNC_AW (see page 73)	Picks either ANSI or Unicode (if N_UNICODE (see page 81) is defined) version of the function (with either 'A' or 'W' suffix accordingly).
N_GCC (see page 73)	Defined if compiling with GCC.
N_INT_MAX (see page 74)	Maximum value for NInt (see page 67).
N_INT_MIN (see page 74)	Minimum value for NInt (see page 67).
N_INT16_MAX (see page 74)	Maximum value for NShort (see page 68).
N_INT16_MIN (see page 74)	Minimum value for NShort (see page 68).
N_INT32_MAX (see page 74)	Maximum value for NInt (see page 67).
N_INT32_MIN (see page 74)	Minimum value for NInt (see page 67).
N_INT64_MAX (see page 75)	Maximum value for NInt64 (see page 67).
N_INT64_MIN (see page 75)	Minimum value for NInt64 (see page 67).
N_INT8_MAX (see page 75)	Maximum value for NSByte (see page 68).
N_INT8_MIN (see page 75)	Minimum value for NSByte (see page 68).
N_LIB (see page 75)	Defined if compiling static library.
N_LINUX (see page 75)	Defined if compiling for Linux.
N_LONG_MAX (see page 76)	Maximum value for NLong (see page 68).
N_LONG_MIN (see page 76)	Minimum value for NLong (see page 68).
N_MAC (see page 76)	Defined if compiling for Mac OS.
N_MSVC (see page 76)	Defined if compiling with Microsoft Visual C++.
N_NO_ANSI_FUNC (see page 76)	Defined if compiling for platform without ANSI versions of the functions support.
N_NO_INT_64 (see page 77)	Defined if compiling for platform without 64-bit integer types support.
N_PACKED (see page 77)	Struct packing/alignment.

N_POS_TYPE_MAX (see page 77)	Maximum value for NPosType (see page 68).
N_POS_TYPE_MIN (see page 77)	Minimum value for NPosType (see page 68).
N_SBYTE_MAX (see page 77)	Maximum value for NSByte (see page 68).
N_SBYTE_MIN (see page 77)	Minimum value for NSByte (see page 68).
N_SHORT_MAX (see page 78)	Maximum value for NShort (see page 68).
N_SHORT_MIN (see page 78)	Minimum value for NShort (see page 68).
N_SINGLE_EPSILON (see page 78)	Epsilon value for NSingle (see page 69).
N_SINGLE_MAX (see page 78)	Maximum value for NSingle (see page 69).
N_SINGLE_MIN (see page 78)	Minimum value for NSingle (see page 69).
N_SIZE_TYPE_MAX (see page 78)	Maximum value for NSizeType (see page 69).
N_SIZE_TYPE_MIN (see page 79)	Minimum value for NSizeType (see page 69).
N_STRUCT_AW (see page 79)	Picks either ANSI or Unicode (if N_UNICODE (see page 81) is defined) version of the struct (with either 'A' or 'W' suffix accordingly)
N_T (see page 79)	Makes either ANSI or Unicode (if N_UNICODE (see page 81) is defined) string or character constant.
N_UINT_MAX (see page 79)	Maximum value for NUInt (see page 69).
N_UINT_MIN (see page 79)	Minimum value for NUInt (see page 69).
N_UINT16_MAX (see page 80)	Maximum value for NUShort (see page 70).
N_UINT16_MIN (see page 80)	Minimum value for NUShort (see page 70).
N_UINT32_MAX (see page 80)	Maximum value for NUInt (see page 69).
N_UINT32_MIN (see page 80)	Minimum value for NUInt (see page 69).
N_UINT64_MAX (see page 80)	Maximum value for NULong (see page 70).
N_UINT64_MIN (see page 80)	Minimum value for NULong (see page 70).
N_UINT8_MAX (see page 81)	Maximum value for NUInt8 (see page 70).
N_UINT8_MIN (see page 81)	Minimum value for NUInt8 (see page 70).
N_ULONG_MAX (see page 81)	Maximum value for NULong (see page 70).
N_ULONG_MIN (see page 81)	Minimum value for NULong (see page 70).
N_UNICODE (see page 81)	Defined if compiling with Unicode character set (affects NChar (see page 66) type).
N_USHORT_MAX (see page 81)	Maximum value for NUShort (see page 70).
N_USHORT_MIN (see page 82)	Minimum value for NUShort (see page 70).
N_WINDOWS (see page 82)	Defined if compiling for Windows.
NFalse (see page 82)	False value for NBoolean (see page 66).
NTrue (see page 82)	True value for NBoolean (see page 66).
NULL (see page 82)	Null value for pointer.

Module

NTypes Module (see page 62)

Structures

	Name	Description
	NIndexPair_ (see page 64)	Represents a pair of indexes.
	NRational_ (see page 65)	Represents a signed rational number.
	NURational_ (see page 65)	Represents an unsigned rational number.
	NIndexPair (see page 64)	Represents a pair of indexes.
	NRational (see page 65)	Represents a signed rational number.
	NURational (see page 65)	Represents an unsigned rational number.

Types

Name	Description
NAChar (see page 65)	ANSI character (8-bit).
NBool (see page 66)	Same as NBoolean (see page 66).
NBoolean (see page 66)	32-bit boolean value. See also NTrue (see page 82) and NFalse (see page 82).
NByte (see page 66)	Same as NUInt8 (see page 70).
NChar (see page 66)	Character type. Either NAChar (see page 65) or NWChar (see page 70) (if N_UNICODE (see page 81) is defined).
NDouble (see page 66)	Double precision floating point number.
NFloat (see page 66)	Same as NSingle (see page 69).
NHandle (see page 67)	Pointer to unspecified data (same as void *).
NInt (see page 67)	Same as NInt32 (see page 67).
NInt16 (see page 67)	16-bit signed integer (short).
NInt32 (see page 67)	32-bit signed integer (int).
NInt64 (see page 67)	64-bit signed integer (long). Not available on some 32-bit platforms.
NInt8 (see page 67)	8-bit signed integer (signed byte).
NLong (see page 68)	Same as NInt64 (see page 67).
NPosType (see page 68)	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 (see page 68)	Result of a function (same as NInt (see page 67)). See also NErrors module.
NSByte (see page 68)	Same as NInt8 (see page 67).
NShort (see page 68)	Same as NInt16 (see page 67).
NSingle (see page 69)	Single precision floating point number.
NSizeType (see page 69)	Platform dependent size type. Unsigned 32-bit integer on 32-bit platform, unsigned 64-bit integer on 64-bit platform.
NUInt (see page 69)	Same as NUInt32 (see page 69).
NUInt16 (see page 69)	16-bit unsigned integer (unsigned short).
NUInt32 (see page 69)	32-bit unsigned integer (unsigned int).
NUInt64 (see page 69)	64-bit unsigned integer (unsigned long). Not available on some 32-bit platforms.
NUInt8 (see page 70)	8-bit unsigned integer (byte).
NULong (see page 70)	Same as NUInt64 (see page 69).
NUShort (see page 70)	Same as NUInt16 (see page 69).
NWChar (see page 70)	Unicode character (16-bit).

8.1.2 NDeviceManager Library

Provides functionality for managing devices, like cameras or fingerprint scanners.

Remarks

Requirements (Windows)

- **Import Library**: NDeviceManager.dll.lib
- **DLL**: *NDeviceManager.dll*
- **Required DLLs**:

- NCore.dll (see page 24).
- NImages.dll (see page 98)

Requirements (Linux)

- **Shared object**: *libNDeviceManager.so*.
- **Required objects**:
  - libNCore.so.
  - linNImages.so.

Modules

Name	Description
Camera Module (see page 86)	Provides functions for working with cameras.
CameraMan Module (see page 94)	Manages, enumerates and creates Cameras.
NDeviceManager Module (see page 97)	Provides general information about NDeviceManager library.

8.1.2.1 Camera Module

Provides functions for working with cameras.

Files

Name	Description
Camera.h (see page 94)	Header file for Camera module. Provides functionality for managing cameras.

Functions

	Name	Description
	CameraGetCurrentFrame (see page 87)	Gets current frame from the capture device.
	CameraGetId (see page 88)	Gets associated device identifier.
	CameraGetVideoFormat (see page 88)	Gets current video format for the capture device.
	CameraGetVideoFormats (see page 88)	Gets all supported video format for the capture device.
	CameraIsCapturing (see page 89)	Checks camera status.
	CameraSetVideoFormat (see page 89)	Sets video format for the capture device.
	CameraStartCapturing (see page 90)	Starts capturing.
	CameraStopCapturing (see page 90)	Stops capturing.
	HCamera (see page 91)	Handle to Camera object.

Macros

Name	Description
CAMERAP_EXPOSURE (see page 91)	Identifier of type N_TYPE_UINT (see page 53) specifying camera exposure (total amount of light allowed to fall on the camera during taking video frame). The exposure range can be acquired using CAMERAP_EXPOSURE_MIN (see page 92) and CAMERAP_EXPOSURE_MAX (see page 92).
CAMERAP_EXPOSURE_MAX (see page 92)	Read only identifier of type N_TYPE_UINT (see page 53) specifying camera maximum exposure value.

CAMERAP_EXPOSURE_MIN (see page 92)	Read only identifier of type N_TYPE_UINT (see page 53) specifying camera's minimum exposure value.
CAMERAP_GAIN (see page 92)	Identifier of type N_TYPE_UINT (see page 53) specifying camera gain value. The gain range can be acquired using CAMERAP_GAIN_MIN (see page 93) and CAMERAP_GAIN_MAX (see page 92).
CAMERAP_GAIN_MAX (see page 92)	Read only identifier of type N_TYPE_UINT (see page 53) specifying maximum camera gain value.
CAMERAP_GAIN_MIN (see page 93)	Read only identifier of type N_TYPE_UINT (see page 53) specifying minimum camera gain value.
CAMERAP_IP_CHANNEL_ID (see page 93)	Identifier of type N_TYPE_STRING (see page 53) specifying Cisco IP camera streaming Channel Id.
CAMERAP_IP_CHANNEL_NAME (see page 93)	Identifier of type N_TYPE_STRING (see page 53) specifying Cisco IP camera streaming Channel name.
CAMERAP_IP_PASSWORD (see page 93)	Identifier of type N_TYPE_STRING (see page 53) specifying Cisco IP camera password.
CAMERAP_IP_USERNAME (see page 93)	Identifier of type N_TYPE_STRING (see page 53) specifying Cisco IP camera user name.

Structs, Records, Enums

	Name	Description
	CameraVideoFormat_ (see page 91)	Describes camera video format.
	CameraVideoFormat (see page 91)	Describes camera video format.

8.1.2.1.1 Functions

8.1.2.1.1.1 CameraGetCurrentFrame Function

Gets current frame from the capture device.

C++

```
NResult N_API CameraGetCurrentFrame(HCamera hCamera, HNImage * pHImage);
```

Parameters

Parameters	Description
HCamera hCamera	[in] Handle to the Camera object.
HNImage * pHImage	[out] Points to HNImage (see page 125) object which receives current frame from the camera. Return Value List

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*hCamera* or *pHImage* is NULL (see page 82).

Module

Camera Module (see page 86)

8.1.2.1.1.2 CameraGetId Function

Gets associated device identifier.

C++

```
NResult N_API CameraGetId(HCamera hCamera, NChar * pValue);
```

Parameters

Parameters	Description
HCamera hCamera	[in] Handle to the Camera object.
NChar * pValue	[out] Pointer to string that receives camera identifier.

Returns

If the function succeeds and *pValue* is NULL (see page 82), the return value is length of the string (not including the NULL (see page 82)-terminator) *pValue* should point to.

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*hCamera* is NULL (see page 82).

Module

Camera Module (see page 86)

8.1.2.1.1.3 CameraGetVideoFormat Function

Gets current video format for the capture device.

C++

```
NResult N_API CameraGetVideoFormat(HCamera hCamera, CameraVideoFormat * pVideoFormat);
```

Parameters

Parameters	Description
HCamera hCamera	[in] Handle to the Camera object.
CameraVideoFormat * pVideoFormat	[out] Points to CameraVideoFormat (see page 91) structure that describes current camera video format.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*hCamera* or *pVideoFormat* is NULL (see page 82).

Module

Camera Module (see page 86)

8.1.2.1.1.4 CameraGetVideoFormats Function

Gets all supported video format for the capture device.

C++

```
NResult N_API CameraGetVideoFormats(HCamera hCamera, CameraVideoFormat * arVideoFormats);
```

Parameters

Parameters	Description
HCamera hCamera	[in] Handle to the Camera object.
CameraVideoFormat * arVideoFormats	[out] Points to CameraVideoFormat (see page 91) structure that describes all supported video formats for the current capture device.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*hCamera* or *arVideoFormats* is NULL (see page 82).

Module

Camera Module (see page 86)

8.1.2.1.1.5 CameraIsCapturing Function

Checks camera status.

C++

```
NResult N_API CameraIsCapturing(HCamera hCamera, NBool * pValue);
```

Parameters

Parameters	Description
HCamera hCamera	[in] Handle to the Camera object.
NBool * pValue	[out] Pointer to NBool (see page 66) that receives value indicating whether camera is already capturing.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*hCamera* or *pValue* is NULL (see page 82).

Module

Camera Module (see page 86)

8.1.2.1.1.6 CameraSetVideoFormat Function

Sets video format for the capture device.

C++

```
NResult N_API CameraSetVideoFormat(HCamera hCamera, const CameraVideoFormat * pVideoFormat);
```

Parameters

Parameters	Description
HCamera hCamera	[in] Handle to the Camera object.

const CameraVideoFormat * pVideoFormat	[in] Points to CameraVideoFormat (see page 91) structure that describes camera video format.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*hCamera* or *pVideoFormat* is NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	Capturing is already started

Module

Camera Module (see page 86)

8.1.2.1.1.7 CameraStartCapturing Function

Starts capturing.

C++

```
NResult N_API CameraStartCapturing(HCamera hCamera);
```

Parameters

Parameters	Description
HCamera hCamera	[in] Handle to the Camera object.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*hCamera* is NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	The camera is already capturing.
N_E_FAILED (see page 29)	Unspecified error has occurred.

Module

Camera Module (see page 86)

8.1.2.1.1.8 CameraStopCapturing Function

Stops capturing.

C++

```
NResult N_API CameraStopCapturing(HCamera hCamera);
```

Parameters

Parameters	Description
HCamera hCamera	[in] Handle to the Camera object.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*hCamera* is NULL (see page 82).

Module

Camera Module (see page 86)

8.1.2.1.1.9 HCamera

Handle to Camera object.

Remarks

The following parameters can be passed to function with corresponding handle:

- CAMERAP_AUTOMATIC_SETTINGS
- CAMERAP_MIROR_HORIZONTAL
- CAMERAP_MIROR_VERTICAL

Module

Camera Module (see page 86)

8.1.2.1.2 Structs, Records, Enums

8.1.2.1.2.1 CameraVideoFormat Structure

Describes camera video format.

C++

```
typedef struct CameraVideoFormat_ {
  NInt FrameWidth;
  NInt FrameHeight;
  NFloat FrameRate;
} CameraVideoFormat;
```

Members

Members	Description
NInt FrameWidth;	Number of frames captured per second.
NInt FrameHeight;	Width of the frame image.
NFloat FrameRate;	Height of the frame image.

See Also

CameraVideoFormat_

Module

Camera Module (see page 86)

8.1.2.1.3 Macros

8.1.2.1.3.1 CAMERAP_EXPOSURE Macro

Identifier of type N_TYPE_UINT (see page 53) specifying camera exposure (total amount of light allowed to fall on the camera during taking video frame). The exposure range can be acquired using CAMERAP_EXPOSURE_MIN (see page 92) and CAMERAP_EXPOSURE_MAX (see page 92).

C++

```
#define CAMERAP_EXPOSURE 10420
```

Remarks

This parameter is used only when automatic settings is disabled. See CAMERAP_AUTOMATIC_SETTINGS.

Module

Camera Module (see page 86)

8.1.2.1.3.2 CAMERAP_EXPOSURE_MAX Macro

Read only identifier of type N_TYPE_UINT (see page 53) specifying camera maximum exposure value.

C++

```
#define CAMERAP_EXPOSURE_MAX 10422
```

Remarks

This parameter is used only when automatic settings is disabled. See CAMERAP_AUTOMATIC_SETTINGS.

Module

Camera Module (see page 86)

8.1.2.1.3.3 CAMERAP_EXPOSURE_MIN Macro

Read only identifier of type N_TYPE_UINT (see page 53) specifying camera's minimum exposure value.

C++

```
#define CAMERAP_EXPOSURE_MIN 10421
```

Remarks

This parameter is used only when automatic settings is disabled. See CAMERAP_AUTOMATIC_SETTINGS.

Module

Camera Module (see page 86)

8.1.2.1.3.4 CAMERAP_GAIN Macro

Identifier of type N_TYPE_UINT (see page 53) specifying camera gain value. The gain range can be acquired using CAMERAP_GAIN_MIN (see page 93) and CAMERAP_GAIN_MAX (see page 92).

C++

```
#define CAMERAP_GAIN 10410
```

Remarks

This parameter is used only when automatic settings is disabled. See CAMERAP_AUTOMATIC_SETTINGS.

Module

Camera Module (see page 86)

8.1.2.1.3.5 CAMERAP_GAIN_MAX Macro

Read only identifier of type N_TYPE_UINT (see page 53) specifying maximum camera gain value.

C++

```
#define CAMERAP_GAIN_MAX 10412
```

Remarks

This parameter is used only when automatic settings is disabled. See CAMERAP_AUTOMATIC_SETTINGS

Module

Camera Module (see page 86)

8.1.2.1.3.6 CAMERAP_GAIN_MIN Macro

Read only identifier of type N_TYPE_UINT (see page 53) specifying minimum camera gain value.

C++

```
#define CAMERAP_GAIN_MIN 10411
```

Remarks

This parameter is used only when automatic settings is disabled. See CAMERAP_AUTOMATIC_SETTINGS.

Module

Camera Module (see page 86)

8.1.2.1.3.7 CAMERAP_IP_CHANNEL_ID Macro

Identifier of type N_TYPE_STRING (see page 53) specifying Cisco IP camera streaming Channel Id.

C++

```
#define CAMERAP_IP_CHANNEL_ID 10503
```

Module

Camera Module (see page 86)

8.1.2.1.3.8 CAMERAP_IP_CHANNEL_NAME Macro

Identifier of type N_TYPE_STRING (see page 53) specifying Cisco IP camera streaming Channel name.

C++

```
#define CAMERAP_IP_CHANNEL_NAME 10504
```

Module

Camera Module (see page 86)

8.1.2.1.3.9 CAMERAP_IP_PASSWORD Macro

Identifier of type N_TYPE_STRING (see page 53) specifying Cisco IP camera password.

C++

```
#define CAMERAP_IP_PASSWORD 10502
```

Module

Camera Module (see page 86)

8.1.2.1.3.10 CAMERAP_IP_USERNAME Macro

Identifier of type N_TYPE_STRING (see page 53) specifying Cisco IP camera user name.

C++

```
#define CAMERAP_IP_USERNAME 10501
```

Module

Camera Module (see page 86)

8.1.2.1.4 Files

8.1.2.1.4.1 Camera.h

Header file for Camera module. Provides functionality for managing cameras.

Functions

	Name	Description
	CameraGetCurrentFrame (see page 87)	Gets current frame from the capture device.
	CameraGetId (see page 88)	Gets associated device identifier.
	CameraGetVideoFormat (see page 88)	Gets current video format for the capture device.
	CameraGetVideoFormats (see page 88)	Gets all supported video format for the capture device.
	CameraIsCapturing (see page 89)	Checks camera status.
	CameraSetVideoFormat (see page 89)	Sets video format for the capture device.
	CameraStartCapturing (see page 90)	Starts capturing.
	CameraStopCapturing (see page 90)	Stops capturing.

Module

Camera Module (see page 86)

Structures

	Name	Description
	CameraVideoFormat_ (see page 91)	Describes camera video format.
	CameraVideoFormat (see page 91)	Describes camera video format.

8.1.2.2 CameraMan Module

Manages, enumerates and creates Cameras.

Files

Name	Description
CameraMan.h (see page 97)	Header file for CameraMan module. Provides functionality for managing, enumerating and creating Cameras.

Functions

	Name	Description
	CameraManGetCamera (see page 95)	Retrieves the camera at the specified index.
	CameraManGetCameraById (see page 95)	Retrieves the camera by specified identifier.

	CameraManGetCameraCount (see page 96)	Retrieves the number of cameras.
	CameraManInitialize (see page 96)	Initializes CameraMan library.
	CameraManUninitialize (see page 96)	Uninitializes CameraMan library.

8.1.2.2.1 Functions

8.1.2.2.1.1 CameraManGetCamera Function

Retrieves the camera at the specified index.

C++

```
NResult N_API CameraManGetCamera(NInt index, HCamera * pHCamera);
```

Parameters

Parameters	Description
NInt index	[in] Index of camera to retrieve.
HCamera * pHCamera	[out] Points to HCamera (see page 91) that receives handle to Camera.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*pHCamera* is NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	There are no initialized CameraMan objects.

Module

CameraMan Module (see page 94)

8.1.2.2.1.2 CameraManGetCameraById Function

Retrieves the camera by specified identifier.

C++

```
NResult N_API CameraManGetCameraById(const NChar * szId, HCamera * pHCamera);
```

Parameters

Parameters	Description
const NChar * szId	[in] Points to string that specifies the camera.
HCamera * pHCamera	[out] Points to HCamera (see page 91) that receives handle to Camera.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*pHCamera* is NULL (see page 82).

N_E_INVALID_OPERATION (see page 29)	There are no initialized CameraMan objects.

Module

CameraMan Module (see page 94)

8.1.2.2.1.3 CameraManGetCameraCount Function

Retrieves the number of cameras.

C++

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

Parameters

Parameters	Description
NInt * pValue	[out] Pointer to NInt (see page 67) that receives number of cameras.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*pHCamera* is NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	There are no initialized CameraMan objects.

Module

CameraMan Module (see page 94)

8.1.2.2.1.4 CameraManInitialize Function

Initializes CameraMan library.

C++

```
NResult N_API CameraManInitialize();
```

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_FAILED (see page 29)	Unspecified error has occurred.

Module

CameraMan Module (see page 94)

8.1.2.2.1.5 CameraManUninitialize Function

Uninitializes CameraMan library.

C++

```
void N_API CameraManUninitialize();
```

Module

CameraMan Module (see page 94)

8.1.2.2.2 Files

8.1.2.2.2.1 CameraMan.h

Header file for CameraMan module. Provides functionality for managing, enumerating and creating Cameras.

Functions

	Name	Description
	CameraManGetCamera (see page 95)	Retrieves the camera at the specified index.
	CameraManGetCameraById (see page 95)	Retrieves the camera by specified identifier.
	CameraManGetCameraCount (see page 96)	Retrieves the number of cameras.
	CameraManInitialize (see page 96)	Initializes CameraMan library.
	CameraManUninitialize (see page 96)	Uninitializes CameraMan library.

Module

CameraMan Module (see page 94)

8.1.2.3 NDeviceManager Module

Provides general information about NDeviceManager (see page 85) library.

Files

Name	Description
NDeviceManager.h (see page 98)	Header file for NDeviceManager (see page 85) library. Provides general functions for retrieving library information.

Functions

	Name	Description
	NDeviceManagerGetInfo (see page 97)	Retrieves NLibraryInfo structure which contains general information about NDeviceManager (see page 85) library.

8.1.2.3.1 Functions

8.1.2.3.1.1 NDeviceManagerGetInfo Function

Retrieves NLibraryInfo structure which contains general information about NDeviceManager (see page 85) library.

C++

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

Parameters

Parameters	Description
NLibraryInfo * pValue	[out] NLibraryInfo structure which contains information about NDeviceManager (see page 85) library.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

NDeviceManager Module (see page 97)

8.1.2.3.2 Files

8.1.2.3.2.1 NDeviceManager.h

Header file for NDeviceManager (see page 85) library. Provides general functions for retrieving library information.

Functions

	Name	Description
	NDeviceManagerGetInfo (see page 97)	Retrieves NLibraryInfo structure which contains general information about NDeviceManager (see page 85) library.

Module

NDeviceManager Module (see page 97)

8.1.3 NImages Library

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

Remarks

Requirements (Windows)

- **Import Library**: NImages.dll.lib
- **DLLs**:
  - *NImages.dll*
  - NCore.dll (see page 24)

Requirements (Linux)

- **Shared objects**:
  - *libNImages.so*
  - libNcore.so

Modules

Name	Description
NImageFormat Module (see page 99)	Provides functionality for loading and saving images in format-neutral way.
NImage Module (see page 114)	Provides functionality for managing images.
NImages Module (see page 126)	Provides library registration and other additional functionality.
NRgbImage Module (see page 128)	Provides functionality for managing 24-bit RGB images.

NMonochromeImage Module (see page 130)	Creates color wrapper for grayscale image.
NGrayscaleImage Module (see page 132)	Provides functionality for managing 8-bit grayscale images.
Tiff Module (see page 132)	Provides functionality for loading images in TIFF format.
NPixelFormat Module (see page 135)	Provides functionality for working with image pixel format.
Bmp Module (see page 138)	Provides functionality for loading and saving images in BMP format.
Jpeg Module (see page 144)	Provides functionality for loading and saving images in JPEG format.
NImageFile Module (see page 150)	Provides functionality for reading image files in format-neutral way.

8.1.3.1 NImageFormat Module

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

Files

Name	Description
NImageFormat.h (see page 112)	Header file for NImageFormat module. Provides functionality for loading and saving images in format-neutral way.

Functions

	Name	Description
	NImageFormatCanRead (see page 100)	Retrieves a value indicating whether the image format supports reading.
	NImageFormatCanWrite (see page 100)	Retrieves a value indicating whether the image format supports writing.
	NImageFormatCanWriteMultiple (see page 101)	Retrieves a value indicating whether the image format supports writing of multiple images.
	NImageFormatGetBmp (see page 101)	Retrieves BMP image format.
	NImageFormatGetDefaultFileExtension (see page 102)	Retrieves default extension of the image format.
	NImageFormatGetFileFilter (see page 102)	Retrieves file filter of the image format.
	NImageFormatGetFormat (see page 103)	Retrieves supported image format with the specified index.
	NImageFormatGetFormatCount (see page 103)	Retrieves number of supported image formats.
	NImageFormatGetIHead (see page 104)	Retrieves NIST IHead image format.
	NImageFormatGetJpeg (see page 104)	Retrieves JPEG image format.
	NImageFormatGetJpeg2K (see page 104)	Retrieves JPEG2K image format.
	NImageFormatGetName (see page 105)	Retrieves name of the image format.
	NImageFormatGetPng (see page 105)	Retrieves PNG image format.
	NImageFormatGetTiff (see page 106)	Retrieves TIFF image format.
	NImageFormatGetWsq (see page 106)	Retrieves WSQ image format.
	NImageFormatLoadImageFromFile (see page 107)	Loads image from file of specified image format.
	NImageFormatLoadImageFromMemory (see page 107)	Loads image from the memory buffer containing file of the specified format.

	NImageFormatOpenFile (see page 108)	Opens image file with specified file name and image format.
	NImageFormatOpenFileFromMemory (see page 108)	Opens image file from the memory buffer containing file of specified format.
	NImageFormatOpenFileFromStream (see page 109)	Opens image file from the memory stream containing file of specified format.
	NImageFormatSaveImagesToFile (see page 110)	Saves specified number of images to the file in the specified format.
	NImageFormatSaveImageToFile (see page 110)	Saves image to the file in the specified format.
	NImageFormatSaveImageToMemory (see page 111)	Saves image to the memory buffer in the specified format.
	NImageFormatSelect (see page 112)	Retrieves supported image format registered with file extension of specified file name and supporting reading/writing as specified.
	HNImageFormat (see page 112)	Handle to image format.

8.1.3.1.1 Functions

8.1.3.1.1.1 NImageFormatCanRead Function

Retrieves a value indicating whether the image format supports reading.

C++

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

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
NBool * pValue	[out] Pointer to NBool (see page 66) that receives value indicating whether the image format supports reading.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when hImageFormat or pValue is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.2 NImageFormatCanWrite Function

Retrieves a value indicating whether the image format supports writing.

C++

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

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
NBool * pValue	[out] Pointer to NBool (see page 66) that receives value indicating whether the image format supports writing.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* or *pValue* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.3 NImageFormatCanWriteMultiple Function

Retrieves a value indicating whether the image format supports writing of multiple images.

C++

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

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
NBool * pValue	[out] Pointer to NBool (see page 66) that receives value indicating whether the image format supports writing of multiple images.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* or *pValue* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.4 NImageFormatGetBmp Function

Retrieves BMP image format.

C++

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

Parameters

Parameters	Description
HNImageFormat * pValue	[out] Pointer to HNImageFormat (see page 112) that receives handle to image format.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when pValue is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.5 NImageFormatGetDefaultFileExtension Function

Retrieves default extension of the image format.

C++

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

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
NChar * pValue	[out] Pointer to string that receives default file extension of the image format. Can be NULL (see page 82).

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
Length of a string	If the function succeeds and *pValue* is NULL (see page 82), the return value is length of the string (not including the NULL (see page 82)-terminator) *pValue* should point to.
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.6 NImageFormatGetFileFilter Function

Retrieves file filter of the image format.

C++

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

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
NChar * pValue	[out] Pointer to the string that receives file filter of the image format. Can be NULL (see page 82).

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
Length of a string	If the function succeeds and *pValue* is NULL (see page 82), the return value is length of the string (not including the NULL (see page 82)-terminator) *pValue* should point to.
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.7 NImageFormatGetFormat Function

Retrieves supported image format with the specified index.

C++

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

Parameters

Parameters	Description
NInt index	[in] Specifies zero-based supported image format index to retrieve.
HNImageFormat * pValue	[out] Pointer to HNImageFormat (see page 112) that receives image format.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when *index* is less that zero or greater than or equal to supported image format count.
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pValue* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.8 NImageFormatGetFormatCount Function

Retrieves number of supported image formats.

C++

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

Parameters

Parameters	Description
NInt * pValue	[out] Pointer to NInt (see page 67) that receives number of supported image formats.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pValue* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.9 NImageFormatGetIHead Function

Retrieves NIST IHead image format.

C++

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

Parameters

Parameters	Description
HNImageFormat * pValue	[out] Pointer to HNImageFormat (see page 112) that receives handle to image format.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pValue* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.10 NImageFormatGetJpeg Function

Retrieves JPEG image format.

C++

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

Parameters

Parameters	Description
HNImageFormat * pValue	[out] Pointer to HNImageFormat (see page 112) that receives handle to image format.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pValue* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.11 NImageFormatGetJpeg2K Function

Retrieves JPEG2K image format.

C++

```
NResult N_API NImageFormatGetJpeg2K(HNImageFormat * pValue);
```

Parameters

Parameters	Description
HNImageFormat * pValue	[out] Pointer to HNImageFormat (see page 112) that receives handle to image format.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pValue* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.12 NImageFormatGetName Function

Retrieves name of the image format.

C++

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

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
NChar * pValue	[out] Pointer to the string that receives name of the image format. Can be NULL (see page 82).

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
Length of a string	If the function succeeds and *pValue* is NULL (see page 82), the return value is length of the string (not including the NULL (see page 82)-terminator) *pValue* should point to.
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* is NULL (see page 82).
Length of a string	If the function succeeds and *pValue* is NULL (see page 82), the return value is length of the string (not including the NULL (see page 82)-terminator) *pValue* should point to.

Module

NImageFormat Module (see page 99)

8.1.3.1.1.13 NImageFormatGetPng Function

Retrieves PNG image format.

C++

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

Parameters

Parameters	Description
HNImageFormat * pValue	[out] Pointer to HNImageFormat (see page 112) that receives handle to image format.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pValue* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.14 NImageFormatGetTiff Function

Retrieves TIFF image format.

C++

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

Parameters

Parameters	Description
HNImageFormat * pValue	[out] Pointer to HNImageFormat (see page 112) that receives handle to image format.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pValue* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.15 NImageFormatGetWsq Function

Retrieves WSQ image format.

C++

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

Parameters

Parameters	Description
HNImageFormat * pValue	[out] Pointer to HNImageFormat (see page 112) that receives handle to image format.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pValue* is NULL (see page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.16 NImageFormatLoadImageFromFile Function

Loads image from file of specified image format.

C++

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

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
const NChar * szFileName	[in] Points to string that specifies file name.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat*, *szFileName* or *pHImage* is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when format of file specified by *szFileName* is invalid.
N_E_NOT_SUPPORTED (see page 30)	If the function fails, this error can be returned. This error code is returned when image format specified by *hImageFormat* does not support reading.

Module

NImageFormat Module (see page 99)

8.1.3.1.1.17 NImageFormatLoadImageFromMemory Function

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

C++

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

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.

const void * buffer	[out] Pointer to memory buffer.
NSizeType bufferLength	[in] Length of memory buffer.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* or *pHImageFormat* is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when format of the file contained in memory buffer specified by buffer is invalid for the specified image format.
N_E_NOT_SUPPORTED (see page 30)	If the function fails, this error can be returned. This error code is returned when image format specified by *hImageFormat* does not support reading.

Module

NImageFormat Module (see page 99)

8.1.3.1.1.18 NImageFormatOpenFile Function

Opens image file with specified file name and image format.

C++

```
NResult N_API NImageFormatOpenFile(HNImageFormat hImageFormat, const NChar * szFileName,
HNImageFile * pHImageFile);
```

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
const NChar * szFileName	[in] File name to open.
HNImageFile * pHImageFile	[out] Pointer to HNImageFile (see page 153) that receives handle to opened image file.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* or *pHImageFile* is NULL (see page 82).
N_E_NOT_SUPPORTED (see page 30)	If the function fails, this error can be returned. This error code is returned when image format specified by *hImageFormat* does not support reading.

Module

NImageFormat Module (see page 99)

8.1.3.1.1.19 NImageFormatOpenFileFromMemory Function

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

C++

```
NResult N_API NImageFormatOpenFileFromMemory(HNImageFormat hImageFormat, const void *
buffer, NSizeType bufferLength, HNImageFile * pHImageFile);
```

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
const void * buffer	[out] Pointer to memory buffer.
NSizeType bufferLength	[in] Length of memory buffer.
HNImageFile * pHImageFile	[out] Pointer to HNImageFile (see page 153) that receives handle to opened image file.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* or *pHImageFile* is NULL (see page 82) - or - *buffer* is NULL (see page 82) and *bufferLength* is not equal to zero.
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when format of file contained in buffer specified by buffer is invalid for specified image format.
N_E_NOT_SUPPORTED (see page 30)	If the function fails, this error can be returned. This error code is returned when image format specified by *hImageFormat* does not support reading.

Module

NImageFormat Module (see page 99)

8.1.3.1.1.20 NImageFormatOpenFileFromStream Function

Opens image file from the memory stream containing file of specified format.

C++

```
NResult N_API NImageFormatOpenFileFromStream(HNImageFormat hImageFormat, HNStream hStream,
NBool ownsStream, HNImageFile * pHImageFile);
```

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
HNStream hStream	[in] Handle to memory stream.
NBool ownsStream	[in] If true, the stream is closed by the writer when done; otherwise false.
HNImageFile * pHImageFile	[out] Pointer to HNImageFile (see page 153) that receives handle to opened image file.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* or *pHImageFile* is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when format of file contained in buffer specified by buffer is invalid for specified image format.
N_E_NOT_SUPPORTED (see page 30)	If the function fails, this error can be returned. This error code is returned when image format specified by *hImageFormat* does not support reading.

Module

NImageFormat Module (see page 99)

8.1.3.1.1.21 NImageFormatSaveImagesToFile Function

Saves specified number of images to the file in the specified format.

C++

```
NResult N_API NImageFormatSaveImagesToFile(HNImageFormat hImageFormat, NInt imageCount,
HNImage * arHImages, const NAChar * szFileName);
```

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
NInt imageCount	[in] Number of images to save.
HNImage * arHImages	[out] Handles to all images that should be saved.
const NAChar * szFileName	[in] Points to memory buffer that contains file names for all images to save.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* or *arHImages* is NULL (see page 82).
N_E_NOT_SUPPORTED (see page 30)	If the function fails, this error can be returned. This error code is returned when image format specified by *hImageFormat* does not support writing.
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when image *imageCount* value is equal or less than zero.

Module

NImageFormat Module (see page 99)

8.1.3.1.1.22 NImageFormatSaveImageToFile Function

Saves image to the file in the specified format.

C++

```
NResult N_API NImageFormatSaveImageToFile(HNImageFormat hImageFormat, HNImage hImage, const
```

```
NChar * szFileName);
```

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
HNImage hImage	[in] Handle to image.
const NChar * szFileName	[in] Points to string that specifies file name.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat*, *hImage* or *szFileName* is NULL (see page 82).
N_E_NOT_SUPPORTED (see page 30)	If the function fails, this error can be returned. This error code is returned when image format specified by *hImageFormat* does not support writing.

Module

NImageFormat Module (see page 99)

8.1.3.1.1.23 NImageFormatSaveImageToMemory Function

Saves image to the memory buffer in the specified format.

C++

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

Parameters

Parameters	Description
HNImageFormat hImageFormat	[in] Handle to image format.
HNImage hImage	[in] Handle to image.
void * * pBuffer	[out] Pointer to void * that receives pointer to allocated memory buffer.
NSizeType * pBufferLength	[out] Pointer to NSizeType (see page 69) that receives size of allocated memory buffer.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFormat*, *hImage*, *pBuffer* or *pBufferLength* is NULL (see page 82).
N_E_OUT_OF_MEMORY (see page 31)	If the function fails, this error can be returned. This error code is returned when there was not enough memory to allocate memory buffer.
N_E_NOT_SUPPORTED (see page 30)	If the function fails, this error can be returned. This error code is returned when image format specified by *hImageFormat* does not support writing.

Remarks

Memory buffer allocated by this function must be deallocated by NFree (see page 43) function when it is no longer needed.

Module

NImageFormat Module (see page 99)

8.1.3.1.1.24 NImageFormatSelect Function

Retrieves supported image format registered with file extension of specified file name and supporting reading/writing as specified.

C++

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

Parameters

Parameters	Description
const NChar * szFileName	[in] Points to string that specifies file name.
NFileAccess fileAccess	[in] Specifies that image format should support reading, writing or both.
HNImageFormat * pHImageFormat	[out] Pointer to HNImageFormat (see page 112) that receives handle to image format.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when *fileAccess* value is invalid.
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *szFileName* or *pHImageFormat* is NULL (see page 82).

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 page 82).

Module

NImageFormat Module (see page 99)

8.1.3.1.1.25 HNImageFormat

Handle to image format.

Module

NImageFormat Module (see page 99)

8.1.3.1.2 Files

8.1.3.1.2.1 NImageFormat.h

Header file for NImageFormat module. Provides functionality for loading and saving images in format-neutral way.

Functions

	Name	Description
	NImageFormatCanRead (see page 100)	Retrieves a value indicating whether the image format supports reading.
	NImageFormatCanWrite (see page 100)	Retrieves a value indicating whether the image format supports writing.
	NImageFormatCanWriteMultiple (see page 101)	Retrieves a value indicating whether the image format supports writing of multiple images.
	NImageFormatGetBmp (see page 101)	Retrieves BMP image format.
	NImageFormatGetDefaultFileExtension (see page 102)	Retrieves default extension of the image format.
	NImageFormatGetFileFilter (see page 102)	Retrieves file filter of the image format.
	NImageFormatGetFormat (see page 103)	Retrieves supported image format with the specified index.
	NImageFormatGetFormatCount (see page 103)	Retrieves number of supported image formats.
	NImageFormatGetIHead (see page 104)	Retrieves NIST IHead image format.
	NImageFormatGetJpeg (see page 104)	Retrieves JPEG image format.
	NImageFormatGetJpeg2K (see page 104)	Retrieves JPEG2K image format.
	NImageFormatGetName (see page 105)	Retrieves name of the image format.
	NImageFormatGetPng (see page 105)	Retrieves PNG image format.
	NImageFormatGetTiff (see page 106)	Retrieves TIFF image format.
	NImageFormatGetWsq (see page 106)	Retrieves WSQ image format.
	NImageFormatLoadImageFromFile (see page 107)	Loads image from file of specified image format.
	NImageFormatLoadImageFromMemory (see page 107)	Loads image from the memory buffer containing file of the specified format.
	NImageFormatOpenFile (see page 108)	Opens image file with specified file name and image format.
	NImageFormatOpenFileFromMemory (see page 108)	Opens image file from the memory buffer containing file of specified format.
	NImageFormatOpenFileFromStream (see page 109)	Opens image file from the memory stream containing file of specified format.
	NImageFormatSaveImagesToFile (see page 110)	Saves specified number of images to the file in the specified format.
	NImageFormatSaveImageToFile (see page 110)	Saves image to the file in the specified format.
	NImageFormatSaveImageToMemory (see page 111)	Saves image to the memory buffer in the specified format.
	NImageFormatSelect (see page 112)	Retrieves supported image format registered with file extension of specified file name and supporting reading/writing as specified.

Module

NImageFormat Module (see page 99)

8.1.3.2 NImage Module

Provides functionality for managing images.

Files

Name	Description
NImage.h (see page 125)	Header file for NImage module. Provides functionality for managing images.

Functions

	Name	Description
	NImageClone (see page 114)	Creates a new image that is a copy of specified image.
	NImageCreate (see page 115)	Creates an image with specified pixel format, size, stride and resolution.
	NImageCreateFromData (see page 116)	Creates an image with specified pixel format, size, stride and resolution and copies specified pixels to it.
	NImageCreateFromFile (see page 117)	Creates (loads) an image from file with specified format.
	NImageCreateFromImage (see page 118)	Creates an image from specified image with specified pixel format and stride.
	NImageCreateFromImageEx (see page 119)	Creates an image from specified image with specified pixel format, stride and resolution.
	NImageCreateWrapper (see page 119)	Creates an image wrapper for specified image pixels with specified pixel format, size, stride and resolution.
	NImageGetHeight (see page 121)	Retrieves height of the image.
	NImageGetHorzResolution (see page 121)	Retrieves horizontal resolution of the image.
	NImageGetPixelFormat (see page 121)	Retrieves pixel format of the image.
	NImageGetPixels (see page 122)	Retrieves pointer to memory block containing pixels of the image.
	NImageGetSize (see page 122)	Retrieves size of memory block containing pixels of the image.
	NImageGetStride (see page 123)	Retrieves stride (size of one row) of the image.
	NImageGetVertResolution (see page 124)	Retrieves vertical resolution of the image.
	NImageGetWidth (see page 124)	Retrieves width of the image.
	NImageSaveToFile (see page 124)	Saves the image to the file of specified format.
	HNImage (see page 125)	Handle to image.

8.1.3.2.1 Functions

8.1.3.2.1.1 NImageClone Function

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

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to the image.

HNImage * pHClonedImage	[out] Pointer to HNImage (see page 125) that receives handle to created image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pHClonedImage* is NULL (see page 82).

Remarks

Created image must be deleted using NObjectFree (see page 46) function.

Module

NImage Module (see page 114)

8.1.3.2.1.2 NImageCreate Function

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

C++

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

Parameters

Parameters	Description
NPixelFormat pixelFormat	[in] Specifies pixel format of the image.
NUInt width	[in] Specifies width of the image.
NUInt height	[in] Specifies height of the image.
NSizeType stride	[in] Specifies stride of the image. Can be zero.
NFloat horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
NFloat vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to created image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when *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 (see page 28)	If the function fails, this error can be returned. This error code is returned when *pHImage* is NULL (see page 82).

N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when *width* or *height* is zero. - or - *horzResolution* or *vertResolution* is less than zero.
N_E_OUT_OF_MEMORY (see page 31)	If the function fails, this error can be returned. This error code is returned when there was not enough memory.

Remarks

If *stride* is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride (see page 123) function.

Created image must be deleted using NObjectFree (see page 46) function.

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

Module

NImage Module (see page 114)

8.1.3.2.1.3 NImageCreateFromData Function

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

C++

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

Parameters

Parameters	Description
NPixelFormat pixelFormat	[in] Specifies pixel format of the image.
NUInt width	[in] Specifies width of the image.
NUInt height	[in] Specifies height of the image.
NSizeType stride	[in] Specifies stride of the image. Can be zero.
NFloat horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
NFloat vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
NSizeType srcStride	[in] Specifies stride of pixels to be copied to the image.
const void * srcPixels	[in] Points to memory block containing pixels that to be copied to the image.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to created image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).

N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when *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 (see page 28)	If the function fails, this error can be returned. This error code is returned when *srcPixels* or *pHImage* is NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when *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 (see page 123) function.

Format of memory block *srcPixels* points to must be the same as described in NImageGetPixels (see page 122) function, only stride is equal to *srcStride*.

Created image must be deleted using NObjectFree (see page 46) function.

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

Module

NImage Module (see page 114)

8.1.3.2.1.4 NImageCreateFromFile Function

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

C++

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

Parameters

Parameters	Description
const NChar * szFileName	[out] Points to string that specifies file name.
HNImageFormat hImageFormat	[in] Handle to the image format of the file. Can be NULL (see page 82).
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to created image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when *szFileName* or *pHImage* is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when format of file specified by *szFileName* is invalid for specified image format.

N_E_NOT_SUPPORTED (see page 30)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* is NULL (see page 82) and none of supported image formats is registered with file extension of *szFileName*. - or - *hImageFormat* is NULL (see page 82) 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 (see page 82) image format is selected by file extension of *szFileName*.

Created image must be deleted using NObjectFree (see page 46) function.

Module

NImage Module (see page 114)

8.1.3.2.1.5 NImageCreateFromImage Function

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

C++

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

Parameters

Parameters	Description
NPixelFormat pixelFormat	[in] Specifies pixel format of the image.
NSizeType stride	[in] Specifies stride of the image. Can be zero.
HNImage hSrcImage	[in] Handle to image used as source for the image.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to created image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when *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 (see page 28)	If the function fails, this error can be returned. This error code is returned when *hSrcImage* or *pHImage* is NULL (see page 82).

Remarks

If *stride* is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride (see page 123) function.

Created image must be deleted using NObjectFree (see page 46) function.

Module

NImage Module (see page 114)

8.1.3.2.1.6 NImageCreateFromImageEx Function

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

C++

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

Parameters

Parameters	Description
NPixelFormat pixelFormat	[in] Specifies pixel format of the image.
NSizeType stride	[in] Specifies stride of the image. Can be zero.
NFloat horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
NFloat vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
HNImage hSrcImage	[in] Handle to image used as source for the image.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to created image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when *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 (see page 28)	If the function fails, this error can be returned. This error code is returned when *hSrcImage* or *pHImage* is NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when *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 (see page 123) function.

Created image must be deleted using NObjectFree (see page 46) function.

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

Module

NImage Module (see page 114)

8.1.3.2.1.7 NImageCreateWrapper Function

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

C++

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

Parameters

Parameters	Description
NPixelFormat pixelFormat	[in] Specifies pixel format of the image.
NUInt width	[in] Specifies width of the image.
NUInt height	[in] Specifies height of the image.
NSizeType stride	[in] Specifies stride of the image.
NFloat horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
NFloat vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
void * pixels	[out] Points to memory block containing pixels for the image.
NBool ownsPixels	[in] Specifies whether pixels will be automatically deleted with the image (if set to NTrue (see page 82)).
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to created image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when *pixelFormat* has invalid value. - or - *stride* is less than minimal value for specified pixel format and width.
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pixels* or *pHImage* is NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when *width* or *height* is zero. - or - *horzResolution* or *vertResolution* is less than zero.

Remarks

For more information on image stride see NImageGetStride (see page 123) function.

Format of memory block pixels points to must be the same as described in NImageGetPixels (see page 122) function.

Created image must be deleted using NObjectFree (see page 46) function.

pixels must not be deleted during lifetime of the image. If ownsPixels is NTrue (see page 82) then pixels will be automatically deleted with the image.

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

Module

NImage Module (see page 114)

8.1.3.2.1.8 NImageGetHeight Function

Retrieves height of the image.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to the image.
NUInt * pValue	[out] Pointer to NUInt (see page 69) that receives height of the image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pValue* is NULL (see page 82).

Module

NImage Module (see page 114)

8.1.3.2.1.9 NImageGetHorzResolution Function

Retrieves horizontal resolution of the image.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to the image.
NFloat * pValue	[out] Pointer to NFloat (see page 66) that receives horizontal resolution in pixels per inch of the image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pValue* is NULL (see page 82).

Remarks

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

Module

NImage Module (see page 114)

8.1.3.2.1.10 NImageGetPixelFormat Function

Retrieves pixel format of the image.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to the image.
NPixelFormat * pValue	[out] Pointer to NPixelFormat (see page 137) that receives pixel format of the image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pValue* is NULL (see page 82).

Module

NImage Module (see page 114)

8.1.3.2.1.11 NImageGetPixels Function

Retrieves pointer to memory block containing pixels of the image.

C++

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

Parameters

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

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pValue* is NULL (see page 82).

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 (see page 121), NImageGetWidth (see page 124), NImageGetHeight (see page 121), NImageGetStride (see page 123), and NImageGetSize (see page 122) functions.

Module

NImage Module (see page 114)

8.1.3.2.1.12 NImageGetSize Function

Retrieves size of memory block containing pixels of the image.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to the image.
NSizeType * pValue	[out] Pointer to NSizeType (see page 69) that receives size of memory block containing pixels of the image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pValue* is NULL (see page 82).

Remarks

Size of memory block containing image pixels is equal to image height multiplied by image stride. For more information see NImageGetHeight (see page 121) and NImageGetStride (see page 123) functions.

Module

NImage Module (see page 114)

8.1.3.2.1.13 NImageGetStride Function

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

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to the image.
NSizeType * pValue	[out] Pointer to NSizeType (see page 69) that receives stride of the image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pValue* is NULL (see page 82).

Remarks

Stride (size of one row) of the image depends on image pixel format and width. It cannot be less than value obtained with NPixelFormatGetRowSize (see page 137) macro with arguments obtained with NImageGetPixelFormat (see page 121) and NImageGetWidth (see page 124) functions.

Module

NImage Module (see page 114)

8.1.3.2.1.14 NImageGetVertResolution Function

Retrieves vertical resolution of the image.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to the image.
NFloat * pValue	[out] Pointer to NFloat (see page 66) that receives vertical resolution in pixels per inch of the image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pValue* is NULL (see page 82).

Remarks

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

Module

NImage Module (see page 114)

8.1.3.2.1.15 NImageGetWidth Function

Retrieves width of the image.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to the image.
NUInt * pValue	[out] Pointer to NUInt (see page 69) that receives width of the image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pValue* is NULL (see page 82).

Module

NImage Module (see page 114)

8.1.3.2.1.16 NImageSaveToFile Function

Saves the image to the file of specified format.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to NImage object.
const NChar * szFileName	[out] Points to string that specifies file name.
HNImageFormat hImageFormat	[in] Handle to the image format of the file. Can be NULL (see page 82).

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds and *pValue* is not NULL (see page 82), the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *szFileName* is NULL (see page 82).
N_E_NOT_SUPPORTED (see page 30)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* is NULL (see page 82) and none of supported image formats is registered with file extension of *szFileName*. - or - *hImageFormat* is NULL (see page 82) 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 (see page 82) image format is selected by file extension of *szFileName*.

Module

NImage Module (see page 114)

8.1.3.2.1.17 HNImage

Handle to image.

Module

NImage Module (see page 114)

8.1.3.2.2 Files

8.1.3.2.2.1 NImage.h

Header file for NImage module. Provides functionality for managing images.

Functions

	Name	Description
	NImageClone (see page 114)	Creates a new image that is a copy of specified image.
	NImageCreate (see page 115)	Creates an image with specified pixel format, size, stride and resolution.

	NImageCreateFromData (see page 116)	Creates an image with specified pixel format, size, stride and resolution and copies specified pixels to it.
	NImageCreateFromFile (see page 117)	Creates (loads) an image from file with specified format.
	NImageCreateFromImage (see page 118)	Creates an image from specified image with specified pixel format and stride.
	NImageCreateFromImageEx (see page 119)	Creates an image from specified image with specified pixel format, stride and resolution.
	NImageCreateWrapper (see page 119)	Creates an image wrapper for specified image pixels with specified pixel format, size, stride and resolution.
	NImageGetHeight (see page 121)	Retrieves height of the image.
	NImageGetHorzResolution (see page 121)	Retrieves horizontal resolution of the image.
	NImageGetPixelFormat (see page 121)	Retrieves pixel format of the image.
	NImageGetPixels (see page 122)	Retrieves pointer to memory block containing pixels of the image.
	NImageGetSize (see page 122)	Retrieves size of memory block containing pixels of the image.
	NImageGetStride (see page 123)	Retrieves stride (size of one row) of the image.
	NImageGetVertResolution (see page 124)	Retrieves vertical resolution of the image.
	NImageGetWidth (see page 124)	Retrieves width of the image.
	NImageSaveToFile (see page 124)	Saves the image to the file of specified format.

Module

NImage Module (see page 114)

8.1.3.3 NImages Module

Provides library registration and other additional functionality.

Files

Name	Description
NImages.h (see page 128)	Header file for NImages (see page 98) module. Provides library registration and other additional functionality.

Functions

	Name	Description
	NImagesGetGrayscaleColorWrapperEx (see page 126)	Creates color wrapper for a grayscale image.
	NImagesGetInfo (see page 127)	Gets information about the library.

8.1.3.3.1 Functions

8.1.3.3.1.1 NImagesGetGrayscaleColorWrapperEx Function

Creates color wrapper for a grayscale image.

C++

```
NResult N_API NImagesGetGrayscaleColorWrapperEx(HNImage hImage, const NRgb * pMinColor,
const NRgb * pMaxColor, HNImage * pHDstImage);
```

Parameters

Parameters	Description
HNImage hImage	[in] Handle to image.
const NRgb * pMinColor	[out] Specifies color to be used for black color.
const NRgb * pMaxColor	[out] Specifies color to be used for white color.
HNImage * pHDstImage	[out] Pointer to HNImage (see page 125) that receives handle to created image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT (see page 27)	This error code is returned when image specified by *hImage* has non-grayscale pixel format (not *npfGrayscale* or *npfMonochrome*).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pHDstImage* is NULL (see page 82).

Remarks

Created image must be deleted using NObjectFree (see page 46) 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.

Module

NImages Module (see page 126)

8.1.3.3.1.2 NImagesGetInfo Function

Gets information about the library.

C++

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

Parameters

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

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*pValue* is NULL (see page 82).

Module

NImages Module (see page 126)

8.1.3.3.2 Files

8.1.3.3.2.1 NImages.h

Header file for NImages (see page 98) module. Provides library registration and other additional functionality.

Functions

	Name	Description
	NImagesGetGrayscaleColorWrapperEx (see page 126)	Creates color wrapper for a grayscale image.
	NImagesGetInfo (see page 127)	Gets information about the library.

Module

NImages Module (see page 126)

8.1.3.4 NRgbImage Module

Provides functionality for managing 24-bit RGB images.

Files

Name	Description
NRgbImage.h (see page 129)	Header file for NRgbImage module. Provides functionality for managing 24-bit RGB images.

Functions

	Name	Description
	NRgbImageGetPixel (see page 128)	Retrieves value of pixel at the specified coordinates in 24-bit RGB image.
	NRgbImageSetPixel (see page 129)	Sets value of pixel at the specified coordinates in 24-bit RGB image.

8.1.3.4.1 Functions

8.1.3.4.1.1 NRgbImageGetPixel Function

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

C++

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

Parameters

Parameters	Description
HNRgbImage hImage	[in] Handle to image.
NUInt x	[in] Specifies x-coordinate of the pixel.
NUInt y	[in] Specifies y-coordinate of the pixel.
NRgb * pValue	[out] Pointer to NRgb (see page 136) that receives pixel value.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).

N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pValue* is NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when x is greater than or equal to image width. - or - y is greater than or equal to image height.
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when image pixel format is not equal to *npfRgb*.

Module

NRgbImage Module (see page 128)

8.1.3.4.1.2 NRgbImageSetPixel Function

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

C++

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

Parameters

Parameters	Description
HNRgbImage hImage	[in] Handle to image.
NUInt x	[in] Specifies x-coordinate of the pixel.
NUInt y	[in] Specifies y-coordinate of the pixel.
const NRgb * pValue	[out] Pointer to NRgb (see page 136) that specifies new pixel value.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pValue* is NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when x is greater than or equal to image width. - or - y is greater than or equal to image height.
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when image pixel format is not equal to *npfRgb*.

Module

NRgbImage Module (see page 128)

8.1.3.4.2 Files

8.1.3.4.2.1 NRgbImage.h

Header file for NRgbImage module. Provides functionality for managing 24-bit RGB images.

Functions

	Name	Description
	NRgbImageGetPixel (see page 128)	Retrieves value of pixel at the specified coordinates in 24-bit RGB image.
	NRgbImageSetPixel (see page 129)	Sets value of pixel at the specified coordinates in 24-bit RGB image.

Module

NRgbImage Module (see page 128)

8.1.3.5 NMonochromeImage Module

Creates color wrapper for grayscale image.

Files

Name	Description
NMonochromeImage.h (see page 132)	Header file for NMonochromeImage module. Provides functionality for managing 1-bit monochrome images.

Functions

	Name	Description
	NMonochromeImageGetPixel (see page 130)	Retrieves value of pixel at the specified coordinates in 1-bit monochrome image.
	NMonochromeImageSetPixel (see page 131)	Sets value of pixel at the specified coordinates in 1-bit monochrome image.

8.1.3.5.1 Functions

8.1.3.5.1.1 NMonochromeImageGetPixel Function

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

C++

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

Parameters

Parameters	Description
HNMonochromeImage hImage	[in] Handle to image.
NUInt x	[in] Specifies x-coordinate of the pixel.
NUInt y	[in] Specifies y-coordinate of the pixel.
NBool * pValue	[out] Points to NBool (see page 66) that receives pixel value.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *pValue* is NULL (see page 82).

N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	This error code is returned when x is greater than or equal to image width. - or - y is greater than or equal to image height.
N_E_FORMAT (see page 29)	This error code is returned when image pixel format is not equal to *npfMonochrome*.

Remarks

If pixel has black color then a value that *pValue* points to receives NFalse (see page 82) and if it is white then value receives NTrue (see page 82).

Module

NMonochromeImage Module (see page 130)

8.1.3.5.1.2 NMonochromeImageSetPixel Function

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

C++

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

Parameters

Parameters	Description
HNMonochromeImage hImage	[in] Handle to image.
NUInt x	[in] Specifies x-coordinate of the pixel.
NUInt y	[in] Specifies y-coordinate of the pixel.
NBool value	[in] Specifies new pixel value.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	This error code is returned when *hImage* is NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	This error code is returned when x is greater than or equal to image width. - or - y is greater than or equal to image height.
N_E_FORMAT (see page 29)	This error code is returned when image pixel format is not equal to *npfMonochrome*.

Remarks

If value is NFalse (see page 82) then pixel will be black and if it is NTrue (see page 82) then pixel will be white.

Module

NMonochromeImage Module (see page 130)

8.1.3.5.2 Files

8.1.3.5.2.1 NMonochromeImage.h

Header file for NMonochromeImage module. Provides functionality for managing 1-bit monochrome images.

Functions

	Name	Description
	NMonochromeImageGetPixel (see page 130)	Retrieves value of pixel at the specified coordinates in 1-bit monochrome image.
	NMonochromeImageSetPixel (see page 131)	Sets value of pixel at the specified coordinates in 1-bit monochrome image.

Module

NMonochromeImage Module (see page 130)

8.1.3.6 NGrayscaleImage Module

Provides functionality for managing 8-bit grayscale images.

Files

Name	Description
NGrayscaleImage.h (see page 132)	Header file for NGrayscaleImage module. Provides functionality for managing 8-bit grayscale images.

8.1.3.6.1 Files

8.1.3.6.1.1 NGrayscaleImage.h

Header file for NGrayscaleImage module. Provides functionality for managing 8-bit grayscale images.

Module

NGrayscaleImage Module (see page 132)

8.1.3.7 Tiff Module

Provides functionality for loading images in TIFF format.

Files

Name	Description
Tiff.h (see page 134)	Header file for Tiff module. Provides functionality for loading images in TIFF format.

Functions

	Name	Description
	TiffLoadImageFromFile (see page 133)	Loads image from TIFF file.
	TiffLoadImageFromMemory (see page 133)	Loads image from memory buffer containing TIFF file.
	TiffLoadImageFromStream (see page 134)	Loads image from stream containing TIFF file.

8.1.3.7.1 Functions

8.1.3.7.1.1 TiffLoadImageFromFile Function

Loads image from TIFF file.

C++

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

Parameters

Parameters	Description
const NChar * szFileName	[out] Points to string that specifies file name.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	the function fails, this error can be returned. This error code is returned when *szFileName* or *pHImage* is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when 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.

Module

Tiff Module (see page 132)

8.1.3.7.1.2 TiffLoadImageFromMemory Function

Loads image from memory buffer containing TIFF file.

C++

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

Parameters

Parameters	Description
const void * buffer	[in] Pointer to memory buffer.
NSizeType bufferLength	[in] Length of memory buffer.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).

N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *buffer* is NULL (see page 82) and *bufferLength* is not equal to zero. - or - *pHImage* is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when format of file specified by buffer is invalid.

Remarks

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

Module

Tiff Module (see page 132)

8.1.3.7.1.3 TiffLoadImageFromStream Function

Loads image from stream containing TIFF file.

C++

```
NResult N_API TiffLoadImageFromStream(HNStream hStream, HNImage * pHImage);
```

Parameters

Parameters	Description
HNStream hStream	[in] Handle to HNStream containing Tiff image.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).

Module

Tiff Module (see page 132)

8.1.3.7.2 Files

8.1.3.7.2.1 Tiff.h

Header file for Tiff module. Provides functionality for loading images in TIFF format.

Functions

	Name	Description
	TiffLoadImageFromFile (see page 133)	Loads image from TIFF file.
	TiffLoadImageFromMemory (see page 133)	Loads image from memory buffer containing TIFF file.
	TiffLoadImageFromStream (see page 134)	Loads image from stream containing TIFF file.

Module

Tiff Module (see page 132)

8.1.3.8 NPixelFormat Module

Provides functionality for working with image pixel format.

Files

Name	Description
NPixelFormat.h (see page 138)	Header file for NPixelFormat (see page 137) module. Provides functionality for working with image pixel format.

Functions

	Name	Description
	NPixelFormatGetBitsPerPixelFunc (see page 135)	Used internally in NPixelFormatGetBitsPerPixel (see page 137) macro.
	NPixelFormatIsValid (see page 135)	Checks if specified pixel format is valid.

Macros

Name	Description
NCalcRowSize (see page 137)	Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.
NPixelFormatGetBitsPerPixel (see page 137)	Retrieves number of bits used to store a pixel from NPixelFormat (see page 137).
NPixelFormatGetRowSize (see page 137)	Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat (see page 137).
NRgbConst (see page 137)	Makes NRgb (see page 136) constant with field values provided.

Structs, Records, Enums

	Name	Description
	NPixelFormat_ (see page 136)	Specifies pixel format of each pixel in the image.
	NRgb_ (see page 136)	Specifies pixel format of each pixel in the image.
	NRgb (see page 136)	Specifies pixel format of each pixel in the image.

Types

Name	Description
NPixelFormat (see page 137)	Specifies pixel format of each pixel in the image.

8.1.3.8.1 Functions

8.1.3.8.1.1 NPixelFormatGetBitsPerPixelFunc Function

Used internally in NPixelFormatGetBitsPerPixel (see page 137) macro.

C++

```
NUInt N_API NPixelFormatGetBitsPerPixelFunc(NPixelFormat pixelFormat);
```

Module

NPixelFormat Module (see page 135)

8.1.3.8.1.2 NPixelFormatIsValid Function

Checks if specified pixel format is valid.

C++

```
NBool N_API NPixelFormatIsValid(NPixelFormat value);
```

Parameters

Parameters	Description
NPixelFormat value	The NPixelFormat (see page 137) object.

Returns

True, if pixel format is valid; false otherwise.

Module

NPixelFormat Module (see page 135)

8.1.3.8.2 Structs, Records, Enums

8.1.3.8.2.1 NPixelFormat_ Enumeration

Specifies pixel format of each pixel in the image.

C++

```
enum NPixelFormat_ {
  npfMonochrome = 0x00001001,
  npfGrayscale = 0x00301001,
  npfRgb = 0x00303003
};
```

Members

Members	Description
npfMonochrome = 0x00001001	Each pixel value is stored in 1 bit representing either black or white color.
npfGrayscale = 0x00301001	Each pixel value is stored in 8 bits representing 256 shades of gray.
npfRgb = 0x00303003	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.

Module

NPixelFormat Module (see page 135)

8.1.3.8.2.2 NRgb Structure

Specifies pixel format of each pixel in the image.

C++

```
typedef struct NRgb_ {
  NByte Red;
  NByte Green;
  NByte Blue;
} NRgb;
```

Members

Members	Description
NByte Red;	Red component value of this NRgb_.

NByte Green;	Green component value of this NRgb_.
NByte Blue;	Blue component value of this NRgb_.

Module

NPixelFormat Module (see page 135)

8.1.3.8.3 Types

8.1.3.8.3.1 NPixelFormat Type

Specifies pixel format of each pixel in the image.

C++

```
typedef enum NPixelFormat_ NPixelFormat;
```

Module

NPixelFormat Module (see page 135)

8.1.3.8.4 Macros

8.1.3.8.4.1 NCalcRowSize Macro

Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.

C++

```
#define NCalcRowSize(bitCount, length) NCalcRowSizeEx(bitCount, length, 1)
```

Module

NPixelFormat Module (see page 135)

8.1.3.8.4.2 NPixelFormatGetBitsPerPixel Macro

Retrieves number of bits used to store a pixel from NPixelFormat (see page 137).

C++

```
#define NPixelFormatGetBitsPerPixel(pixelFormat)
NPixelFormatGetBitsPerPixelFunc(pixelFormat)
```

Module

NPixelFormat Module (see page 135)

8.1.3.8.4.3 NPixelFormatGetRowSize Macro

Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat (see page 137).

C++

```
#define NPixelFormatGetRowSize(pixelFormat, length) NPixelFormatGetRowSizeEx(pixelFormat,
length, 1)
```

Module

NPixelFormat Module (see page 135)

8.1.3.8.4.4 NRgbConst Macro

Makes NRgb (see page 136) constant with field values provided.

C++

```
#define NRgbConst(red, green, blue) { red, green, blue }
```

Module

NPixelFormat Module (see page 135)

8.1.3.8.5 Files

8.1.3.8.5.1 NPixelFormat.h

Header file for NPixelFormat (see page 137) module. Provides functionality for working with image pixel format.

Enumerations

	Name	Description
	NPixelFormat_ (see page 136)	Specifies pixel format of each pixel in the image.

Functions

	Name	Description
	NPixelFormatGetBitsPerPixelFunc (see page 135)	Used internally in NPixelFormatGetBitsPerPixel (see page 137) macro.
	NPixelFormatIsValid (see page 135)	Checks if specified pixel format is valid.

Macros

Name	Description
NCalcRowSize (see page 137)	Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.
NPixelFormatGetBitsPerPixel (see page 137)	Retrieves number of bits used to store a pixel from NPixelFormat (see page 137).
NPixelFormatGetRowSize (see page 137)	Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat (see page 137).
NRgbConst (see page 137)	Makes NRgb (see page 136) constant with field values provided.

Module

NPixelFormat Module (see page 135)

Structures

	Name	Description
	NRgb_ (see page 136)	Specifies pixel format of each pixel in the image.
	NRgb (see page 136)	Specifies pixel format of each pixel in the image.

Types

Name	Description
NPixelFormat (see page 137)	Specifies pixel format of each pixel in the image.

8.1.3.9 Bmp Module

Provides functionality for loading and saving images in BMP format.

Files

Name	Description
Bmp.h (see page 143)	Header file for Bmp module. Provides functionality for loading and saving images in BMP format.

Functions

	Name	Description
	BmpLoadImageFromFile (see page 139)	Loads image from BMP file.
	BmpLoadImageFromHBitmap (see page 140)	Loads image from Windows HBITMAP.
	BmpLoadImageFromMemory (see page 140)	Loads image from memory buffer containing BMP file.
	BmpLoadImageFromStream (see page 141)	Loads image from stream containing Bmp file.
	BmpSaveImageToFile (see page 141)	Saves image to file in BMP format.
	BmpSaveImageToHBitmap (see page 142)	Saves image to Windows HBITMAP.
	BmpSaveImageToMemory (see page 142)	Saves image to memory buffer in BMP format.
	BmpSaveImageToStream (see page 143)	Saves image to stream in BMP format.

8.1.3.9.1 Functions

8.1.3.9.1.1 BmpLoadImageFromFile Function

Loads image from BMP file.

C++

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

Parameters

Parameters	Description
const NChar * szFileName	[in] Points to string that specifies file name.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. It is returned when szFileName or pHImage is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. It is returned when 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.

Module

Bmp Module (see page 138)

8.1.3.9.1.2 BmpLoadImageFromHBitmap Function

Loads image from Windows HBITMAP.

C++

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

Parameters

Parameters	Description
NHandle handle	[in] Handle that specifies Windows HBITMAP.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. It is returned when handle or pHImage is NULL (see page 82).

Remarks

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

Notes

This function is available only on Windows.

Module

Bmp Module (see page 138)

8.1.3.9.1.3 BmpLoadImageFromMemory Function

Loads image from memory buffer containing BMP file.

C++

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

Parameters

Parameters	Description
const void * buffer	[out] Pointer to memory buffer.
NSizeType bufferLength	[in] Length of memory buffer.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).

N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. It is returned when buffer is NULL (see page 82) and bufferLength is not equal to zero.- or -pHImage is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. It is returned when 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.

Module

Bmp Module (see page 138)

8.1.3.9.1.4 BmpLoadImageFromStream Function

Loads image from stream containing Bmp file.

C++

```
NResult N_API BmpLoadImageFromStream(HNStream hStream, HNImage * pHImage);
```

Parameters

Parameters	Description
HNStream hStream	[in] Handle to HNStream containing Bmp image.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).

Module

Bmp Module (see page 138)

8.1.3.9.1.5 BmpSaveImageToFile Function

Saves image to file in BMP format.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to image.
const NChar * szFileName	[out] Points to string that specifies file name.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. It is returned when hImage or szFileName is NULL (see page 82).

Remarks

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

Module

Bmp Module (see page 138)

8.1.3.9.1.6 BmpSaveImageToHBitmap Function

Saves image to Windows HBITMAP.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to image.
NHandle * pHandle	[out] Pointer to NHandle (see page 67) that receives handle to created Windows HBITMAP.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. It is returned when hImage or pHandle is NULL (see page 82).

Remarks

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

HBITMAP must be freed using GDI function DeleteObject.

Notes

This function is available only on Windows.

Module

Bmp Module (see page 138)

8.1.3.9.1.7 BmpSaveImageToMemory Function

Saves image to memory buffer in BMP format.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to image.
void * * pBuffer	[out] Pointer to void * that receives pointer to allocated memory buffer.
NSizeType * pBufferLength	[out] Pointer to NSizeType (see page 69) that receives size of allocated memory buffer.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).

N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. It is returned when hImage, pBuffer or pBufferLength is NULL (see page 82).
N_E_OUT_OF_MEMORY (see page 31)	If the function fails, this error can be returned. It is returned when 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 (see page 43) function when it is no longer needed.

Module

Bmp Module (see page 138)

8.1.3.9.1.8 BmpSaveImageToStream Function

Saves image to stream in BMP format.

C++

```
NResult N_API BmpSaveImageToStream(HNImage hImage, HNStream hStream);
```

Parameters

Parameters	Description
HNImage hImage	[in] Handle to HNStream containing Bmp image.
HNStream hStream	[out] Handle to HNStream where image will be written.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).

Module

Bmp Module (see page 138)

8.1.3.9.2 Files

8.1.3.9.2.1 Bmp.h

Header file for Bmp module. Provides functionality for loading and saving images in BMP format.

Functions

	Name	Description
	BmpLoadImageFromFile (see page 139)	Loads image from BMP file.
	BmpLoadImageFromHBitmap (see page 140)	Loads image from Windows HBITMAP.
	BmpLoadImageFromMemory (see page 140)	Loads image from memory buffer containing BMP file.
	BmpLoadImageFromStream (see page 141)	Loads image from stream containing Bmp file.
	BmpSaveImageToFile (see page 141)	Saves image to file in BMP format.

	BmpSaveImageToHBitmap (see page 142)	Saves image to Windows HBITMAP.
	BmpSaveImageToMemory (see page 142)	Saves image to memory buffer in BMP format.
	BmpSaveImageToStream (see page 143)	Saves image to stream in BMP format.

Module

Bmp Module (see page 138)

8.1.3.10 Jpeg Module

Provides functionality for loading and saving images in JPEG format.

Files

Name	Description
Jpeg.h (see page 149)	Header file for Jpeg module. Provides functionality for loading and saving images in JPEG format.

Functions

	Name	Description
	JpegLoadImageFromFile (see page 144)	Loads image from JPEG file.
	JpegLoadImageFromMemory (see page 145)	Loads image from the memory buffer containing JPEG file.
	JpegLoadImageFromStream (see page 146)	Loads image from the stream containing JPEG file.
	JpegSaveImageToFile (see page 146)	Saves image to file in JPEG format.
	JpegSaveImageToMemory (see page 147)	Saves image to the memory buffer in JPEG format.
	JpegSaveImageToStream (see page 147)	Saves image to the stream in JPEG format.
	LosslessJpegSaveImageToFile (see page 148)	Saves specified by HNImage (see page 125) handle image to file.
	LosslessJpegSaveImageToMemory (see page 148)	Saves specified lossless JPEG image to memory.
	LosslessJpegSaveImageToStream (see page 149)	Saves specified lossless JPEG image to memory stream.

Macros

Name	Description
JPEG_DEFAULT_QUALITY (see page 149)	Specifies default JPEG quality.

8.1.3.10.1 Functions

8.1.3.10.1.1 JpegLoadImageFromFile Function

Loads image from JPEG file.

C++

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

Parameters

Parameters	Description
const NChar * szFileName	[in] Points to string that specifies file name.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *szFileName* or *pHImage* is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned.This error code is returned when format of file specified by *szFileName* is invalid.

Module

Jpeg Module (see page 144)

8.1.3.10.1.2 JpegLoadImageFromMemory Function

Loads image from the memory buffer containing JPEG file.

C++

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

Parameters

Parameters	Description
const void * buffer	[in] Pointer to memory buffer.
NSizeType bufferLength	[in] Length of memory buffer.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when buffer is NULL (see page 82) and bufferLength is not equal to zero.- or -pHImage is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned.This error code is returned when 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.

Module

Jpeg Module (see page 144)

8.1.3.10.1.3 JpegLoadImageFromStream Function

Loads image from the stream containing JPEG file.

C++

```
NResult N_API JpegLoadImageFromStream(HNStream hStream, HNImage * pHImage);
```

Parameters

Parameters	Description
HNStream hStream	[in] Handle to NStream.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to loaded image.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *buffer* is NULL (see page 82) and *bufferLength* is not equal to zero.
N_E_FORMAT (see page 29)	If the function fails, this error can be returned.This error code is returned when format of file contained in memory buffer specified by buffer is invalid.

Remarks

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

Module

Jpeg Module (see page 144)

8.1.3.10.1.4 JpegSaveImageToFile Function

Saves image to file in JPEG format.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to image.
NInt quality	[in] Specifies quality of JPEG compression.
const NChar * szFileName	[in] Points to string that specifies file name.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage* or *szFileName* is NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned.This error code is returned when quality is less than 1 or greater than 100.

Remarks

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

Module

Jpeg Module (see page 144)

8.1.3.10.1.5 JpegSaveImageToMemory Function

Saves image to the memory buffer in JPEG format.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to image.
NInt quality	[in] Specifies quality of JPEG compression.
void * * pBuffer	[out] Pointer to void * that receives pointer to allocated memory buffer.
NSizeType * pBufferLength	[out] Pointer to NSizeType (see page 69) that receives size of allocated memory buffer.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage*, *pBuffer* or *pBufferLength* is NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned.This error code is returned when quality is less than 1 or greater than 100.
N_E_OUT_OF_MEMORY (see page 31)	If the function fails, this error can be returned.This error code is returned when 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 this function must be deallocated by NFree (see page 43) function when it is no longer needed.

Module

Jpeg Module (see page 144)

8.1.3.10.1.6 JpegSaveImageToStream Function

Saves image to the stream in JPEG format.

C++

```
NResult N_API JpegSaveImageToStream(HNImage hImage, NInt quality, HNStream hStream);
```

Parameters

Parameters	Description
HNImage hImage	[in] Handle to image.
NInt quality	[in] Specifies quality of JPEG compression.

HNStream hStream	[out] Handle to NStream where image will be written.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImage*, *pBuffer* or *pBufferLength* is NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned.This error code is returned when quality is less than 1 or greater than 100.
N_E_OUT_OF_MEMORY (see page 31)	If the function fails, this error can be returned.This error code is returned when 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 this function must be deallocated by NFree (see page 43) function when it is no longer needed.

Module

Jpeg Module (see page 144)

8.1.3.10.1.7 LosslessJpegSaveImageToFile Function

Saves specified by HNImage (see page 125) handle image to file.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to NImage object which contains image to be saved to file.
const NChar * szFileName	[in] Zero terminated string which contains file name where Jpeg image will be saved.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

Jpeg Module (see page 144)

8.1.3.10.1.8 LosslessJpegSaveImageToMemory Function

Saves specified lossless JPEG image to memory.

C++

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

Parameters

Parameters	Description
HNImage hImage	[in] Handle to JPEG image to be saved to memory.
void * * pBuffer	[in] Pointer to memory buffer to save image.
NSizeType * pBufferLength	[in] Length of memory buffer where JPEG image will be saved.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

Jpeg Module (see page 144)

8.1.3.10.1.9 LosslessJpegSaveImageToStream Function

Saves specified lossless JPEG image to memory stream.

C++

```
NResult N_API LosslessJpegSaveImageToStream(HNImage hImage, HNStream hStream);
```

Parameters

Parameters	Description
HNImage hImage	[in] Handle to JPEG image to be saved to memory.
HNStream hStream	[out] Handle to NStream which will be used to save lossless JPEG image.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

Jpeg Module (see page 144)

8.1.3.10.2 Macros

8.1.3.10.2.1 JPEG_DEFAULT_QUALITY Macro

Specifies default JPEG quality.

C++

```
#define JPEG_DEFAULT_QUALITY 75
```

Module

Jpeg Module (see page 144)

8.1.3.10.3 Files

8.1.3.10.3.1 Jpeg.h

Header file for Jpeg module. Provides functionality for loading and saving images in JPEG format.

Functions

	Name	Description
	JpegLoadImageFromFile (see page 144)	Loads image from JPEG file.
	JpegLoadImageFromMemory (see page 145)	Loads image from the memory buffer containing JPEG file.
	JpegLoadImageFromStream (see page 146)	Loads image from the stream containing JPEG file.
	JpegSaveImageToFile (see page 146)	Saves image to file in JPEG format.

	JpegSaveImageToMemory (see page 147)	Saves image to the memory buffer in JPEG format.
	JpegSaveImageToStream (see page 147)	Saves image to the stream in JPEG format.
	LosslessJpegSaveImageToFile (see page 148)	Saves specified by HNImage (see page 125) handle image to file.
	LosslessJpegSaveImageToMemory (see page 148)	Saves specified lossless JPEG image to memory.
	LosslessJpegSaveImageToStream (see page 149)	Saves specified lossless JPEG image to memory stream.

Macros

Name	Description
JPEG_DEFAULT_QUALITY (see page 149)	Specifies default JPEG quality.

Module

Jpeg Module (see page 144)

8.1.3.11 NImageFile Module

Provides functionality for reading image files in format-neutral way.

Files

Name	Description
NImageFile.h (see page 153)	Header file for NImageFile module. Provides functionality for reading image files in format-neutral way.

Functions

	Name	Description
	NImageFileClose (see page 150)	Closes the file associated with the image file.
	NImageFileCreate (see page 151)	Opens image file of specified format.
	NImageFileGetFormat (see page 152)	Retrieves image format of the image file.
	NImageFileIsOpened (see page 152)	Retrieves a value indicating whether the file associated with the image file is opened.
	NImageFileReadImage (see page 153)	Reads image from the image file.
	HNImageFile (see page 153)	Handle to opened read-only image file.

8.1.3.11.1 Functions

8.1.3.11.1.1 NImageFileClose Function

Closes the file associated with the image file.

C++

```
NResult N_API NImageFileClose(HNImageFile hImageFile);
```

Parameters

Parameters	Description
HNImageFile hImageFile	[in] Handle to image file.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFile* is NULL (see page 82).

Remarks

If the file associated with the image file is already closed the function does nothing.

Module

NImageFile Module (see page 150)

8.1.3.11.1.2 NImageFileCreate Function

Opens image file of specified format.

C++

```
NResult N_API NImageFileCreate(const NChar * szFileName, HNImageFormat hImageFormat,
HNImageFile * pHImageFile);
```

Parameters

Parameters	Description
const NChar * szFileName	[in] Points to string that specifies file name.
HNImageFormat hImageFormat	[in] Handle to the image format of the file. Can be NULL (see page 82).
HNImageFile * pHImageFile	[out] Pointer to HNImageFile (see page 153) that receives handle to opened image file.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *szFileName* or *pHImageFile* is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when format of file specified by szFileName is invalid for specified image format.
N_E_NOT_SUPPORTED (see page 30)	If the function fails, this error can be returned. This error code is returned when *hImageFormat* is NULL (see page 82) and none of supported image formats is registered with file extension of *szFileName*. - or - *hImageFormat* is NULL (see page 82) 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 (see page 82) image format is selected by file extension of *szFileName*.

Opened image file must be closed using NObjectFree (see page 46) function.

This function does not check format of the file for all image formats. However format of the file is always checked in NImageFileReadImage (see page 153) function.

Module

NImageFile Module (see page 150)

8.1.3.11.1.3 NImageFileGetFormat Function

Retrieves image format of the image file.

C++

```
NResult N_API NImageFileGetFormat(HNImageFile hImageFile, HNImageFormat * pValue);
```

Parameters

Parameters	Description
HNImageFile hImageFile	[in] Handle to image file.
HNImageFormat * pValue	[out] Pointer to NImageFormat that receives image format of the image file.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFile* or *pValue* is NULL (see page 82).

Module

NImageFile Module (see page 150)

8.1.3.11.1.4 NImageFileIsOpened Function

Retrieves a value indicating whether the file associated with the image file is opened.

C++

```
NResult N_API NImageFileIsOpened(HNImageFile hImageFile, NBool * pValue);
```

Parameters

Parameters	Description
HNImageFile hImageFile	[in] Handle to image file.
NBool * pValue	[out] Pointer to NBool (see page 66) that receives value indicating whether the file associated with the image file is opened.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFile* or *pValue* is NULL (see page 82).

Module

NImageFile Module (see page 150)

8.1.3.11.1.5 NImageFileReadImage Function

Reads image from the image file.

C++

```
NResult N_API NImageFileReadImage(HNImageFile hImageFile, HNImage * pHImage);
```

Parameters

Parameters	Description
HNImageFile hImageFile	[in] Handle to image file.
HNImage * pHImage	[out] Pointer to HNImage (see page 125) that receives handle to image read from the image file.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hImageFile* or *pHImage* is NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when format of file associated with the image file is invalid for the image format of the image file.
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when file associated with the image file is closed.

Remarks

If all images are already read from the image file then handle to image returned via *pHImage* is NULL (see page 82).

Module

NImageFile Module (see page 150)

8.1.3.11.1.6 HNImageFile

Handle to opened read-only image file.

Module

NImageFile Module (see page 150)

8.1.3.11.2 Macros

8.1.3.11.3 Files

8.1.3.11.3.1 NImageFile.h

Header file for NImageFile module. Provides functionality for reading image files in format-neutral way.

Functions

	Name	Description
	NImageFileClose (see page 150)	Closes the file associated with the image file.
	NImageFileCreate (see page 151)	Opens image file of specified format.

	NImageFileGetFormat (see page 152)	Retrieves image format of the image file.
	NImageFileIsOpened (see page 152)	Retrieves a value indicating whether the file associated with the image file is opened.
	NImageFileReadImage (see page 153)	Reads image from the image file.

Module

NImageFile Module (see page 150)

8.1.4 NLicensing Library

Provides functionality for getting, releasing licenses.

Remarks

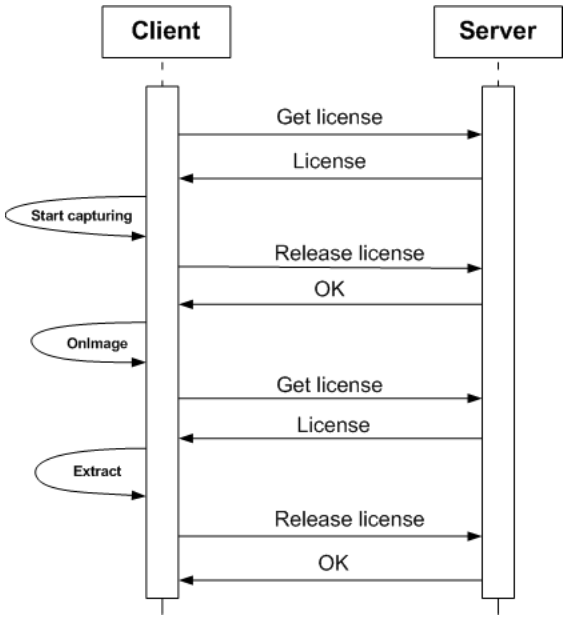
Requirements (Windows)

- **Import Library**: NLicensing.dll.lib
- **DLLs**:
  - *NLicensing.dll*
  - NCore.dll (see page 24)

Requirements (Linux)

- **Shared objects**:
  - *libNLicensing.so*
  - libNcore.so

Concurrent license. Client and License Server scenario.



Modules

Name	Description
NLicensing Module (see page 155)	Provides functionality for getting, releasing licenses.

8.1.4.1 NLicensing Module

Provides functionality for getting, releasing licenses.

Files

Name	Description
NLicensing.h (see page 161)	Header file for the NLicensing (see page 154) library. Provides functionality for getting and releasing licenses.

Functions

	Name	Description
	NLicenseGetInfo (see page 155)	Retrieves license information of specified product.
	NLicenseIsComponentActivated (see page 156)	Checks if component specified by name is activated.
	NLicenseObtain (see page 156)	Obtains product license from the specified server.
	NLicenseRelease (see page 158)	Releases license.
	NLicensingGetInfo (see page 160)	Gets information about the library.

Structs, Records, Enums

	Name	Description
	NLicenseInfo_ (see page 161)	Enumerates values describing license.
	NLicenseInfo (see page 161)	Enumerates values describing license.

8.1.4.1.1 Functions

8.1.4.1.1.1 NLicenseGetInfo Function

Retrieves license information of specified product.

C++

```
NResult N_API NLicenseGetInfo(const NChar * szProduct, NLicenseInfo * pLicenseInfo);
```

Parameters

Parameters	Description
const NChar * szProduct	[in] Zero terminated string which contains product name to retrieve license information.
NLicenseInfo * pLicenseInfo	[out] Pointer to NLicenseInfo (see page 161) structure.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*szProduct* or *pLicenseInfo* is NULL (see page 82).

Module

NLicensing Module (see page 155)

8.1.4.1.1.2 NLicenseIsComponentActivated Function

Checks if component specified by name is activated.

C++

```
NResult N_API NLicenseIsComponentActivated(const NChar * szName, NBool * pValue);
```

Parameters

Parameters	Description
const NChar * szName	[in] Zero terminated string which contains component name.
NBool * pValue	[out] NBool (see page 66) value indicating if the component was activated.

Returns

If the function succeeds, the return value is N_OK (see page 32). If application working folder does not contain required files, N_E_FILE_NOT_FOUND error is thrown.

Remarks

These component names for szName parameter can be used:

- Matchers.Fusion
- Matchers.Fingers
- Matchers.Faces
- Matchers.Irises
- Matchers.Palms
- Matchers.Fingers Pro
- Matchers.Faces Pro
- Matchers.Irises Pro
- Matchers.Palms Pro
- Extractors.Fingers
- Extractors.Faces Detection
- Extractors.Faces Extraction
- Extractors.Irises
- Extractors.Palms

Module

NLicensing Module (see page 155)

8.1.4.1.1.3 NLicenseObtain Function

Obtains product license from the specified server.

C++

```
NResult N_API NLicenseObtain(const NChar * szAddress, const NChar * szPort, const NChar *
szProducts, NBool * pAvailable);
```

Parameters

Parameters	Description
const NChar * szAddress	[in] Server address where license manager is installed as a server. License manager can be in the same or in other computer. **/local** used when license is tied with the computer using serial number.
const NChar * szPort	[in] License manager server port.
const NChar * szProducts	[in] Zero terminated string which contains license name. For a complete list of available values for this parameter see Remarks section.
NBool * pAvailable	[out] Returns NTrue (see page 82) if license available; otherwise, NFalse (see page 82).

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*szAddress*, *szPort*, *szProducts* or *pAvailable* is NULL (see page 82).
N_E_FORMAT (see page 29)	*szProducts* incorrect format.
N_E_FAILED (see page 29)	Unspecified error has occurred.

Remarks

These values are available for *szProducts* parameter (by default concurrent license is obtained):

1. **VeriFinger SDK**:
   1. SingleComputerLicense:VFExtractor or concurrent:VFExtractor
   2. SingleComputerLicense:VFMatcher

VFExtractor - VeriFinger Extractor

VFMatcher - VeriFinger Matcher

2. **VeriLook SDK**:
   1. SingleComputerLicense:VLExtractor or concurrent:VLExtractor
   2. SingleComputerLicense:VLMatcher

VLExtractor - VeriLook Extractor

VLMatcher - VeriLook Matcher

3. **VeriLook Surveillance SDK**:
   1. SingleComputerLicense:VLSurveillance
4. **MegaMatcher SDK**:
   1. SingleComputerLicense:MegaMatcherServer
   2. SingleComputerLicense:MegaMatcherClusterServer
   3. SingleComputerLicense:MegaMatcherClusterNode
   4. SingleComputerLicense:MegaMatcherClient
   5. concurrent:MegaMatcherClient
5. **VeriEye SDK**:

1. SingleComputerLicense:VEExtractor
2. SingleComputerLicense:VEMatcher

VEExtractor - VeriEye Extractor

VEMatcher - VeriEye Matcher

6. **BSS SDK**:
   1. SingleComputerLicense:VFBSS
   2. SingleComputerLicense:VLBSS
   3. SingleComputerLicense:VEBSS
7. **VeriFinger VAR SDK**:
   1. SingleComputerLicense:VFExtractorV
   2. SingleComputerLicense:VFMatcherV

VFExtractorV - VeriFinger Extractor

VFMatcherV - VeriFinger Matcher

8. **VeriLook VAR SDK**:
   1. SingleComputerLicense:VLExtractorV
   2. SingleComputerLicense:VLMatcherV

VLExtractorV - VeriLook Extractor

VLMatcherV - VeriLook Matcher

9. **VeriEye VAR SDK**:
   1. SingleComputerLicense:VEExtractorV
   2. SingleComputerLicense:VEMatcherV

VEExtractorV - VeriEye Extractor

VEMatcherV - VeriEye Matcher

10. **SentiSight SDK:**
    1. SingleComputerLicense:SentiSight

Note: nonconcurrent licenses (SingleComputerLicense) **can not be released** with NLicenseRelease (see page 158) function, but .NLicenseRelease (see page 158) function must be called before program exits.

If concurrent license is in use, then NLicenseRelease (see page 158) function should be called after features extraction.

Notes

Acquired license must be released using NLicenseRelease (see page 158) function.

Module

NLicensing Module (see page 155)

8.1.4.1.1.4 NLicenseRelease Function

Releases license.

C++

```
NResult N_API NLicenseRelease(const NChar * szProducts);
```

Parameters

Parameters	Description
const NChar * szProducts	[in] Product names to release licenses.

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*szProducts* is NULL (see page 82).
N_E_FORMAT (see page 29)	*szProducts* incorrect format.
N_E_FAILED (see page 29)	Unspecified error has occurred.

Remarks

These values are available for *szProducts* parameter:

1. **VeriFinger SDK**:
    1. SingleComputerLicense:VFExtractor or concurrent:VFExtractor
    2. SingleComputerLicense:VFMatcher

VFExtractor - VeriFinger Extractor

VFMatcher - VeriFinger Matcher

2. **VeriLook SDK**:
    1. SingleComputerLicense:VLExtractor or concurrent:VLExtractor
    2. SingleComputerLicense:VLMatcher

VLExtractor - VeriLook Extractor

VLMatcher - VeriLook Matcher

3. **VeriLook Surveillance SDK**:
    1. SingleComputerLicense:VLSurveillance
4. **MegaMatcher SDK**:
    1. SingleComputerLicense:MegaMatcherServer
    2. SingleComputerLicense:MegaMatcherClusterServer
    3. SingleComputerLicense:MegaMatcherClusterNode
    4. SingleComputerLicense:MegaMatcherClient
    5. concurrent:MegaMatcherClient
5. **VeriEye SDK**:
    1. SingleComputerLicense:VEExtractor
    2. SingleComputerLicense:VEMatcher

VEExtractor - VeriEye Extractor

VEMatcher - VeriEye Matcher

6. **BSS SDK**:
    1. SingleComputerLicense:VFBSS
    2. SingleComputerLicense:VLBSS
    3. SingleComputerLicense:VEBSS
7. **VeriFinger VAR SDK**:
    1. SingleComputerLicense:VFExtractorV

2. SingleComputerLicense:VFMatcherV

VFExtractorV - VeriFinger Extractor

VFMatcherV - VeriFinger Matcher

8. **VeriLook VAR SDK**:
   1. SingleComputerLicense:VLExtractorV
   2. SingleComputerLicense:VLMatcherV

VLExtractorV - VeriLook Extractor

VLMatcherV - VeriLook Matcher

9. **VeriEye VAR SDK**:
   1. SingleComputerLicense:VEExtractorV
   2. SingleComputerLicense:VEMatcherV

VEExtractorV - VeriEye Extractor

VEMatcherV - VeriEye Matcher

10. **SentiSight SDK:**
    1. SingleComputerLicense:SentiSight

Note: nonconcurrent licenses (SingleComputerLicense) **can not be released** with NLicenseRelease function, but .NLicenseRelease function must be called before program exits.

If concurrent license is in use, then NLicenseRelease function should be called after features extraction.

Module

NLicensing Module (see page 155)

8.1.4.1.1.5 NLicensingGetInfo Function

Gets information about the library.

C++

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

Parameters

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

Returns

If the function succeeds, the return value is N_OK (see page 32).

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

Error code	Condition
N_E_ARGUMENT_NULL (see page 28)	*pValue* is NULL (see page 82).

Module

NLicensing Module (see page 155)

8.1.4.1.2 Structs, Records, Enums

8.1.4.1.2.1 NLicenseInfo Structure

Enumerates values describing license.

C++

```
typedef struct NLicenseInfo_ {
  NBool IsObtained;
  NInt DistributorId;
  NInt SerialNumber;
} NLicenseInfo;
```

Members

Members	Description
NBool IsObtained;	Indicates if a license obtain was successful.
NInt DistributorId;	Id of a distributor which generated license.
NInt SerialNumber;	License serial number.

Module

NLicensing Module (see page 155)

8.1.4.1.3 Files

8.1.4.1.3.1 NLicensing.h

Header file for the NLicensing (see page 154) library. Provides functionality for getting and releasing licenses.

Functions

	Name	Description
	NLicenseGetInfo (see page 155)	Retrieves license information of specified product.
	NLicenseIsComponentActivated (see page 156)	Checks if component specified by name is activated.
	NLicenseObtain (see page 156)	Obtains product license from the specified server.
	NLicenseRelease (see page 158)	Releases license.
	NLicensingGetInfo (see page 160)	Gets information about the library.

Module

NLicensing Module (see page 155)

Structures

	Name	Description
	NLicenseInfo_ (see page 161)	Enumerates values describing license.
	NLicenseInfo (see page 161)	Enumerates values describing license.

8.1.5 NVideo Library

Provides functionality for reading video files.

Remarks

Requirements for different operating systems:

Windows

- **Import library**: NVideo.dll.lib.
- **DLL**: NVideo.dll.
- **Requirements**: NCore.dll (see page 24), NImages.dll (see page 98)

Linux

- **Shared object**: libNVideo.so.
- **Requirements**: libNCore.so, libNImages.so

Modules

Name	Description
NVideoReader Module (see page 162)	
NVideoWriter Module (see page 166)	Provides functionality for writing to a video file.

8.1.5.1 NVideoReader Module

Files

Name	Description
NVideoReader.h (see page 165)	Header file for NVideoReader module. Provides functionality for reading video files.

Functions

	Name	Description
	NVideoReaderCreateFromFile (see page 162)	Creates a HNVideoReader for the specified video file.
	NVideoReaderGetFrame (see page 163)	Reads one frame from video file.
	NVideoReaderGetFrameCount (see page 164)	Counts number of frames in the video file which was assigned to the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).
	NVideoReaderGetFrameHeight (see page 164)	The function retrieves height of frames in the video file which was assigned to the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).
	NVideoReaderGetFrameRate (see page 164)	The function calculates a frame rate (number of frames presented in one second) of the video file. The file was assigned to the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).
	NVideoReaderGetFrameWidth (see page 165)	The function retrieves width of frames in the video file which was assigned to the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).

8.1.5.1.1 Functions

8.1.5.1.1.1 NVideoReaderCreateFromFile Function

Creates a HNVideoReader for the specified video file.

C++

```
NResult N_API NVideoReaderCreateFromFile(const NChar * szFileName, HNVideoReader *
pHReader);
```

Parameters

Parameters	Description
const NChar * szFileName	[in] A filename of a video file to be opened.
HNVideoReader * pHReader	[out] A handle of HNVideoReader type to be allocated and assigned to the specified video file.

Returns

Return value	Description
N_OK (see page 32)	If function succeeds it returns N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If provided file name is NULL (see page 82), function fails and returns.
N_E_FAILED (see page 29).	If function fails to open a file or assign a handle to the file.

Remarks

When handle is not needed anymore, it must be freed using NVideoFree function.

Module

NVideoReader Module (see page 162)

8.1.5.1.1.2 NVideoReaderGetFrame Function

Reads one frame from video file.

C++

```
NResult N_API NVideoReaderGetFrame(HNVideoReader hReader, NInt frameIndex, HNImage *
pHFrame);
```

Parameters

Parameters	Description
HNVideoReader hReader	[in] A handle of HNVideoReader type. This is the same handle as was created using NVideoReaderCreateFromFile (see page 162) function.
NInt frameIndex	[in] An index of a frame to be read form the video file.
HNImage * pHFrame	[out] An image which will be filled with information available in the frame.

Returns

Return value	Description
N_OK (see page 32)	If function succeeds it returns N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	A HNVideoReader handle or pHFrame are NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	A frame index is larger than the number of frames in the video file.
N_E_FAILED (see page 29)	Failed to grab frame form video file.

Module

NVideoReader Module (see page 162)

8.1.5.1.1.3 NVideoReaderGetFrameCount Function

Counts number of frames in the video file which was assigned to the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).

C++

```
NResult N_API NVideoReaderGetFrameCount(HNVideoReader hReader, NInt * pValue);
```

Parameters

Parameters	Description
HNVideoReader hReader	[in] A handle of HNVideoReader type. This is the same handle as was created using NVideo (see page 161)-ReaderCreateFromFile function.
NInt * pValue	[out] A number of frames in the video file.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT (see page 27) NULL (see page 82)	Function might fail if a handle of HNVideoReader is not created.

Module

NVideoReader Module (see page 162)

8.1.5.1.1.4 NVideoReaderGetFrameHeight Function

The function retrieves height of frames in the video file which was assigned to the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).

C++

```
NResult N_API NVideoReaderGetFrameHeight(HNVideoReader hReader, NUInt * pValue);
```

Parameters

Parameters	Description
HNVideoReader hReader	[in] A handle of HNVideoReader type. This is the same handle as was created using NVideoReaderCreateFromFile (see page 162) function.
NUInt * pValue	[out] A height of frames in the video file.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT (see page 27) NULL (see page 82).	Function might fail if a handle of HNVideoReader is not created.

Module

NVideoReader Module (see page 162)

8.1.5.1.1.5 NVideoReaderGetFrameRate Function

The function calculates a frame rate (number of frames presented in one second) of the video file. The file was assigned to

the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).

C++

```
NResult N_API NVideoReaderGetFrameRate(HNVideoReader hReader, NDouble * pValue);
```

Parameters

Parameters	Description
HNVideoReader hReader	[in] A handle of HNVideoReader type. This is the same handle as was created using NVideoReaderCreateFromFile (see page 162) function.
NDouble * pValue	[out] A number of frames presented in one second when the video file is playing.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	Function might fail if a handle of HNVideoReader is not created.

Module

NVideoReader Module (see page 162)

8.1.5.1.1.6 NVideoReaderGetFrameWidth Function

The function retrieves width of frames in the video file which was assigned to the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).

C++

```
NResult N_API NVideoReaderGetFrameWidth(HNVideoReader hReader, NUInt * pValue);
```

Parameters

Parameters	Description
HNVideoReader hReader	A handle of HNVideoReader type. This is the same handle as was created using NVideoReaderCreateFromFile (see page 162) function.
NUInt * pValue	A width of frames in the video file.

Returns

Return value	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	Function might fail if a handle of HNVideoReader is not created.

Module

NVideoReader Module (see page 162)

8.1.5.1.2 Files

8.1.5.1.2.1 NVideoReader.h

Header file for NVideoReader (see page 162) module. Provides functionality for reading video files.

Functions

	Name	Description
	NVideoReaderCreateFromFile (see page 162)	Creates a HNVideoReader for the specified video file.
	NVideoReaderGetFrame (see page 163)	Reads one frame from video file.
	NVideoReaderGetFrameCount (see page 164)	Counts number of frames in the video file which was assigned to the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).
	NVideoReaderGetFrameHeight (see page 164)	The function retrieves height of frames in the video file which was assigned to the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).
	NVideoReaderGetFrameRate (see page 164)	The function calculates a frame rate (number of frames presented in one second) of the video file. The file was assigned to the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).
	NVideoReaderGetFrameWidth (see page 165)	The function retrieves width of frames in the video file which was assigned to the HNVideoReader handle (using function NVideoReaderCreateFromFile (see page 162)).

Module

NVideoReader Module (see page 162)

8.1.5.2 NVideoWriter Module

Provides functionality for writing to a video file.

Remarks

When using XVID encoder make sure that automatic optimisations are turned off. Otherwise NVideoWriter may crash due to a bug in XVID code.

If writing video files with XVID still crashes, then turn off all optimisations. Optimisation settings can be located in Options->Other Options...->Common starting in a popup dialog box where you choose XVID codec.

Files

Name	Description
NVideoWriter.h (see page 168)	Header file for the NVideoWriter module. Provides functionality for writing to a video file.

Functions

	Name	Description
	NVideoWriterCreateFile (see page 167)	Creates a HNVideoWriter (see page 168) for the specified video file.
	NVideoWriterWriteFrame (see page 167)	Appends a frame to previously created video file.
	NVideoWriterOptionsCreateWithGui (see page 167)	Opens a dialog box allowing to choose a compressor for video file.
	HNVideoWriter (see page 168)	Handle to NVideoWriter object.
	HNVideoWriterOptions (see page 168)	A handle of information about video codec.

8.1.5.2.1 Functions

8.1.5.2.1.1 NVideoWriterCreateFile Function

Creates a HNVideoWriter (see page 168) for the specified video file.

C++

```
NResult N_API NVideoWriterCreateFile(const NChar * szFileName, NUInt Width, NUInt Height,
NDouble FrameRate, HNVideoWriterOptions hOptions, HNVideoWriter * pHWriter);
```

Parameters

Parameters	Description
const NChar * szFileName	A filename of a video file to be opened.
NUInt Width	A width of viedo frame.
NUInt Height	A height of video frame.
NDouble FrameRate	Frame rate.
HNVideoWriterOptions hOptions	A handle to Video writer options structure.
HNVideoWriter * pHWriter	A handle of HNVideoWriter (see page 168) type to be allocated and assigned to the specified video file.

Returns

If function succeeds it returns N_OK (see page 32).

If provided file name is NULL (see page 82), function fails and returns N_E_ARGUMENT_NULL (see page 28).

If function fails to create a file or assign a handle to the file, it returns an error of N_E_FAILED (see page 29).

Remarks

When handle is not needed anymore, it must be freed using NVideoWriterClose function.

Module

NVideoWriter Module (see page 166)

8.1.5.2.1.2 NVideoWriterWriteFrame Function

Appends a frame to previously created video file.

C++

```
NResult N_API NVideoWriterWriteFrame(HNVideoWriter hWriter, HNImage hFrame);
```

Parameters

Parameters	Description
HNVideoWriter hWriter	A handle of HNVideoWriter (see page 168) type. This is the same handle as was created using NVideoWriterCreateFile (see page 167) function.
HNImage hFrame	[out] A handle to HNImage (see page 125).

Module

NVideoWriter Module (see page 166)

8.1.5.2.1.3 NVideoWriterOptionsCreateWithGui Function

Opens a dialog box allowing to choose a compressor for video file.

C++

```
NResult N_API NVideoWriterOptionsCreateWithGui(NHandle hParentWnd, HNVideoWriterOptions *
pHOptions);
```

Parameters

Parameters	Description
NHandle hParentWnd	[in] Windows window handle. This handle will be used as parent window for a dialog box.
HNVideoWriterOptions * pHOptions	[in] Handle to NVideoWriterOptions.

Module

NVideoWriter Module (see page 166)

8.1.5.2.1.4 HNVideoWriter

Handle to NVideoWriter object.

Module

NVideoWriter Module (see page 166)

8.1.5.2.1.5 HNVideoWriterOptions

A handle of information about video codec.

Remarks

This handle is allocated by calling NVideoChooseCompression function with appropriate parameters. This handle can be associated with created video file later on by calling NVideoSetCompression function. When writing to the file is done this handle can be freed by calling NVideoFreeCompression function. If this function is called between NVideoSetCompression and NVideoWriterClose functions expect unexpected behaviour.

Module

NVideoWriter Module (see page 166)

8.1.5.2.2 Files

8.1.5.2.2.1 NVideoWriter.h

Header file for the NVideoWriter module. Provides functionality for writing to a video file.

Functions

	Name	Description
	NVideoWriterCreateFile (see page 167)	Creates a HNVideoWriter (see page 168) for the specified video file.
	NVideoWriterWriteFrame (see page 167)	Appends a frame to previously created video file.

Module

NVideoWriter Module (see page 166)

8.1.6 SentiSight Library

Provides the main functionality of HSentiSightEngine (see page 218) Library.

Remarks

Windows

- **Import library**: SentiSight.dll.lib.
- **DLL**: SentiSight.dll.
- **Requirements**: NCore.dll (see page 24), NImages.dll (see page 98).

Linux

- **Shared object**: libSentiSight.so.
- **Requirements**: libNCore.so, libNImages.so.

Notes

There are three main functionality types of HSentiSightEngine (see page 218). Functions starting with prefix SESep are dedicated for foreground/background separation. Functions starting with prefix SELrn are dedicated for object learning. Functions starting with prefix SERec are dedicated for object recognition.

Functions with no special prefix are general purpose functions.

Modules

Name	Description
SentiSight Module (see page 169)	Provides the main functionality of SentiSight Library.

8.1.6.1 SentiSight Module

Provides the main functionality of SentiSight Library.

Functions

	Name	Description
	SECreate (see page 172)	Allocates memory for HSentiSightEngine (see page 218) handle.
	SECreateModel (see page 173)	Allocates memory for HSEModel (see page 214) model.
	SERecDetailsIsTracked (see page 174)	Checks if the recognition details were tracked.
	SELrnAddToModel (see page 174)	Object learning function which adds a representation of a single image to the HSEModel (see page 214) model.
	SELrnAddToModelEx (see page 175)	Adds one or more (*count*) part (*region*) of the image to the model.
	SELrnGeneralizeModel (see page 176)	Compresses contents of the HSEModel (see page 214) model.
	SELrnGeneralizeModelEx (see page 176)	Compresses contents of the HSEModel (see page 214) model.
	SELrnRemoveFromModel (see page 177)	Removes part of the model represented by *pRefId*.
	SEModelClear (see page 178)	Clears everything what was added to model without freeing the object.
	SEModelClone (see page 178)	Clones specified HSEModel (see page 214) object.
	SEModelGetSize (see page 178)	Returns minimum size of buffer needed to save the model of the object.
	SEModelIsEmpty (see page 179)	Test whether model contains any information (added to model).
	SEModelIsLocked (see page 179)	Checks if the model is locked.
	SEModelLoadFromMemory (see page 180)	Retrieves the model of the object from memory.

	SEModelSaveToMemory (see page 181)	Saves the model of the object to memory.
	SEModelSaveToMemoryEx (see page 181)	Saves the model of the object to memory.
	SentiSightGetInfo (see page 182)	Retrieves information about SentiSight library.
	SERecAddModel (see page 182)	Adds a learnt model of the object to the HSEEngine handle.
	SERecAddModelEx (see page 183)	Adds a learnt model of the object to the HSEEngine handle.
	SERecDetailsGetImageToModelTransform (see page 184)	Fills 9 elements 3X3 row wise matrix representing transform. Matrix should be preallocated by user.
	SERecDetailsGetImageToModelTransformEx (see page 185)	Fills 9 elements 3X3 row wise matrix representing transform. Matrix should be preallocated by user.
	SERecDetailsGetModelId (see page 185)	Returns user specified Id for the model being recognized.
	SERecDetailsGetModelToImageTransform (see page 185)	Fills 9 element 3X3 row wise matrix representing transform. Matrix should be preallocated by user.
	SERecDetailsGetModelToImageTransformEx (see page 186)	Fills 9 element 3X3 row wise matrix representing transform. Matrix should be preallocated by user.
	SERecDetailsGetScore (see page 186)	Returns similarity score.
	SERecDetailsGetShape (see page 187)	Gets shape describing approximate region of the image where particular model is recognized.
	SERecDetailsGetShapeEx (see page 187)	Gets shape describing approximate region of the image where particular model is recognized.
	SERecDetailsGetTransformType (see page 188)	Returns type of transform found by recognition process (shape alignment and transform matrices).
	SERecGetAllRecognitionDetails (see page 188)	Gets all recognition details.
	SERecGetModelCount (see page 189)	Returns a number of models set to the HSEEngine handle and used in recognition functions.
	SERecGetModelIds (see page 189)	Returns user all Ids for the model being recognized.
	SERecGetRecognitionDetails (see page 190)	Returns detailed information about object recognition results.
	SERecGetRecognitionDetailsCount (see page 190)	Returns recognition details count in HSEEngine.
	SERecGetTrackingImageSize (see page 191)	Gets an image size used in object recognition with tracking (SERecRecognizeImage (see page 191)).
	SERecRecognizeImage (see page 191)	Object recognition function compares a test image with models which were set to the HSEEngine handle.
	SERecRecognizeImageEx (see page 193)	Object recognition function compares a test image with models which were set to the HSEEngine handle.
	SERecRemoveAllModels (see page 194)	Removes all models added to the recognition engine.
	SERecRemoveModel (see page 195)	Removes a model from the HSEEngine handle.
	SERecSetTrackingImageSize (see page 195)	Sets an image size which will be used in object recognition with tracking (SERecRecognizeImage (see page 191)).
	SESepAccumulateBackground (see page 196)	Accumulates background for foreground/background separation process.
	SESepGetImageSize (see page 197)	Gets an image size of the HSEEngine handle used in foreground/background separation process.
	SESepGetObjectModelSize (see page 197)	Returns minimum size of buffer needed to save a colour model of the object.
	SESepLoadHolderModelFromMemory (see page 198)	Sets a color model of the holder to be used in foreground/background separation process.
	SESepLoadObjectModelFromMemory (see page 199)	Sets a color model of the object to be used in foreground/background separation process.

	SESepResetBackgroundModel (see page 199)	Resets specified background model.
	SESepResetHolderModel (see page 200)	Resets a holder model used in foreground/background separation process.
	SESepResetObjectModel (see page 200)	Resets a colour model of the object of foreground/background separation process.
	SESepSaveModelToMemory (see page 201)	Writes a colour model of the object into provided buffer.
	SESepSaveObjectModelToMemory (see page 201)	Saves specified object of the model to provided memory buffer.
	SESepSeparate (see page 202)	Separates foreground (an object) from background and extracts image mask which can be used in object learning.
	SESepSetImageSize (see page 203)	Sets an image size of the HSEEngine handle used in foreground/background separation process.
	SEShapeAddPoint (see page 204)	Adds point to the end of the shape.
	SEShapeAddPointEx (see page 204)	Adds point to the end of the shape.
	SEShapeClearPoints (see page 205)	Clears shape's points without freeing the object.
	SEShapeClone (see page 205)	Clones HSEShape (see page 214) object.
	SEShapeCreate (see page 205)	Creates new empty shape.
	SEShapeGetCenter (see page 206)	Returns automatically calculated shape center point.
	SEShapeGetHeading (see page 206)	Gets user specified orientation of the shape in radians 0..2pi.
	SEShapeGetPoint (see page 206)	Gets a value of the shape point at specific index.
	SEShapeGetPointCount (see page 207)	Get the number of points in a shape.
	SEShapeGetPoints (see page 207)	Gets all points from a shape.
	SEShapeInsertPoint (see page 208)	Inserts point at a position just before specific index. If new point results in self crossing shape the point is not inserted.
	SEShapeInsertPointEx (see page 208)	Inserts point at a position just before specific index. If new point results in self crossing shape the point is not inserted.
	SEShapeIsLocked (see page 209)	Checks if the shape is locked.
	SEShapeIsValid (see page 209)	Checks if a shape is valid.
	SEShapeRemovePoint (see page 210)	Removes a point from a shape.
	SEShapeRemovePointEx (see page 210)	Removes a point from a shape.
	SEShapeRotate (see page 211)	Rotates shape by amount of radians around the center of the shape.
	SEShapeScale (see page 211)	Scales shape around the center of the shape, below 1 shrinks, above 1 enlarges the shape.
	SEShapeSetHeading (see page 211)	Sets user specified orientation of the shape in radians 0..2pi.
	SEShapeSetPoint (see page 212)	Changes value of the shape point at specific index.
	SEShapeSetPointEx (see page 212)	Changes value of the shape point at specific index.
	SEShapeTestPoint (see page 213)	Gets test point and shape relative position.
	SEShapeTranslate (see page 213)	Translates the shape. Negative distance is translated to the coordinate origin side, positive - out of the coordinate origin.
	HSEModel (see page 214)	A handle of the object model. It is used in object learning and object recognition functions.
	HSERecognitionDetails (see page 214)	Recognition (see page 11) details handle are temporary and are only valid in image recognition, adding or removing models from recognition engine functions calls.
	HSEShape (see page 214)	Handle to shape.

Macros

Name	Description
SEP_LRN_ENHANCE_MASK (see page 217)	Identifier specifying whether to extend image mask during object learning or not.
SEP_LRN_GENERALIZATION_THRESHOLD (see page 217)	Represents Id for generalization threshold parameter.
SEP_LRN_MODE (see page 217)	Represents Id for learning mode parameter.
SEP_REC_SPEED (see page 217)	Represents Id for SentiSight recognition speed parameter.
SEP_REC_THRESHOLD (see page 218)	Represents Id for recognition threshold parameter.
SEP_REC_TRANSFORM_TYPE (see page 218)	Represents Id for transform type parameter to be used in recognition to align shape.
SEP_REC_USE_TRACKING (see page 218)	Represents Id for use tracking for recognition parameter.
SEP_SEP_USE_ADAPTIVE_ALG (see page 218)	Represents Id for a parameter to use an adaptive algorithm for separation.

Structs, Records, Enums

	Name	Description
	SEStatus_ (see page 215)	Enumerates different values of SentiShight learning and recognition processes.
	SELrnMode_ (see page 215)	Enumerates different type of learning parameters.
	SERecSpeed_ (see page 216)	Enumerates SentiSight recognition speed parameters.
	SERecTransformType_ (see page 216)	Transform types used in recognition process to align shape or transform images.
	SEStatus (see page 215)	Enumerates different values of SentiShight learning and recognition processes.
	SELrnMode (see page 215)	Enumerates different type of learning parameters.
	SERecSpeed (see page 216)	Enumerates SentiSight recognition speed parameters.
	SERecTransformType (see page 216)	Transform types used in recognition process to align shape or transform images.

Types

Name	Description
HSentiSightEngine (see page 218)	The main handle of HSentiSightEngine library.
SEModelUpdateStatus (see page 218)	Enumerates different types of SentiSight model update status parameters.

8.1.6.1.1 Functions

8.1.6.1.1.1 SECreate Function

Allocates memory for HSentiSightEngine (see page 218) handle.

C++

```
NResult N_API SECreate(HSEEngine * pHEngine);
```

Parameters

Parameters	Description
HSEEngine * pHEngine	[out] A pointer to HSEEngine handle to be allocated.

Returns

Returns a result of success or failure of type NResult (see page 68).

If function succeeds the return value is N_OK (see page 32).

HSentiSightEngine (see page 218) library uses SSE2 instructions for processor. Function fails if the processor does not support SSE2 instructions. The returned error code is N_E_NOT_SUPPORTED (see page 30). Function fails if there is not enough memory to allocate all the structures of the handle. The returned code is N_E_OUT_OF_MEMORY (see page 31).

Remarks

It creates all internal structure of HSentiSightEngine (see page 218) handle. This handle will be used in major part of HSentiSightEngine (see page 218) functions. When the handle is not needed anymore, it must be freed using SEFree function.

See Also

SEFree

Module

SentiSight Module (see page 169)

8.1.6.1.1.2 SECreateModel Function

Allocates memory for HSEModel (see page 214) model.

C++

```
NResult N_API SECreateModel(HSEEngine hEngine, HSEModel * pHModel);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type.
HSEModel * pHModel	[out] A model to be allocated.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the HSEEngine handle is NULL (see page 82).

Remarks

This function creates internal structure of HSEModel (see page 214) model. The model later can be used in object learning process (function SELrnAddToModel (see page 174)). When the model is not needed anymore, it should be freed by calling SEModelFree funciton.

See Also

SELrnAddToModel (see page 174)

SEModelFree

SELrnGeneralizeModel (see page 176)

SEModelSaveToMemory (see page 181)

SEModelLoadFromMemory (see page 180)

Module

SentiSight Module (see page 169)

8.1.6.1.1.3 SERecDetailsIsTracked Function

Checks if the recognition details were tracked.

C++

```
NResult N_API SERecDetailsIsTracked(HSERecognitionDetails recognitionDetails, NBool *
pValue);
```

Parameters

Parameters	Description
HSERecognitionDetails recognitionDetails	[in] A handle to recognition details that should be checked.
NBool * pValue	[out] A boolean value indicating whether details were tracked.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

SentiSight Module (see page 169)

8.1.6.1.1.4 SELrnAddToModel Function

Object learning function which adds a representation of a single image to the HSEModel (see page 214) model.

C++

```
NResult N_API SELrnAddToModel(HSEEngine hEngine, HSEModel hModel, HNImage hImage, NUInt *
pRefId, SEStatus * pStatus);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type.
HSEModel hModel	[in] A handle of HSEModel (see page 214) type. The image will be added to this model.
HNImage hImage	[in] An image of the object. This image should be 3 channel RGB image.
NUInt * pRefId	[out] Reference id of added visual information.
SEStatus * pStatus	[out] Add to model status.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when one or more of *hEngine*, *hImage* or *hModel* are NULL (see page 82).
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when a mask image type is not *npfGrayscale*.

Remarks

This is the main object learning function. It should be provided with a valid, previously created (using SEModelCreate) handle of the model. The function SELrnAddToModel should be repeatedly called with images of the same object until a complete object model is learnt. After learning the model may be compressed by calling SELrnGeneralizeModel (see page 176) function. Also, this model may be saved to external buffer (function SEModelSaveToMemory (see page 181)) and loaded (function SEModelLoadFromMemory (see page 180)) from it later.

The function SELrnAddToModel must be provided with a RGB image of the object and, preferably, its mask. The mask may be retrieved using SESepSeparate (see page 202) function with the same RGB image of the object, only some preconditions must be considered. The image and the mask must be the same size. If parameter SEP_LRN_ENHANCE_MASK (see page 217) is set to NTrue (see page 82) (using SESetParameter function) the mask will be extended before applying it to the image. By default, this parameter is NFalse (see page 82). Its value may be obtained by calling SEGetParameter with an appropriate parameter identifier.

In the current implementation this function adds content of entire image to the model.

Module

SentiSight Module (see page 169)

8.1.6.1.1.5 SELrnAddToModelEx Function

Adds one or more (*count*) part (*region*) of the image to the model.

C++

```
NResult N_API SELrnAddToModelEx(HSEEngine hEngine, HSEModel hModel, HNImage hImage, NInt count, HNImage * arHMasks, HSEShape * arHShapes, NUInt * arRefIds, SEStatus * arStatuses);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type.
HSEModel hModel	[in] A handle of HSEModel (see page 214) type. The image will be added to this model.
HNImage hImage	[in] An image of the object.
NInt count	[in] Number of image parts.
HNImage * arHMasks	[in] Grayscale 8bpp image array of length count. 0 represents background, 1..255 represents meaningful part of the image. Mask is set by user. Meaningful part of the mask image determinate what part of the input image to add to the model.
HSEShape * arHShapes	[in] HSEShape (see page 214) array of length count. Shape is specified by user and is used to find approximate region of occupation by this model in the image to be recognized.
NUInt * arRefIds	[out] void * array of length count. See SELrnAddToModel (see page 174).
arUpdateStatuses	[out] SEStatus (see page 215) array of length count. For each region reports whether algorithm was able to add to model this region.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when one or more of *hEngine*, *hImage* or *hModel* are NULL (see page 82).

Remarks

Combination of position arHShapes [i] and arHMasks [i] defines one region of the image and shape to be added to the model. One of them can be NULL (see page 82), but not both, in this case it is calculated by the other. In case mask is NULL (see page 82) mask is calculated by shape, in case shape is NULL (see page 82), shape is calculated by mask. In extreme case entire masks or shapes parameter can be NULL (see page 82). In this case all of it is calculated by other parameter.

Module

SentiSight Module (see page 169)

8.1.6.1.1.6 SELrnGeneralizeModel Function

Compresses contents of the HSEModel (see page 214) model.

C++

```
NResult N_API SELrnGeneralizeModel(HSEEngine hEngine, HSEModel hModel, NDouble threshold);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. This is [in] A handle of HSEEngine type. This is [in] A handle of HSEEngine type. This is the same handle used in object learning function.
HSEModel hModel	[in, out] A handle of HSEModel (see page 214) type. This model should contain object model aggregated in object learning part. The function changes contents of this model.
NDouble threshold	[in] A generalization threshold. Its values are discussed below. If a recommended value "0" was used, the most suitable threshold will be calculated and used in the function.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the HSentiSightEngine (see page 218) handle or *hModel* are NULL (see page 82).

Remarks

Object learning function SELrnAddToModel (see page 174) aggregates a model of the object into HSEModel (see page 214) handle. After object learning is complete, the model may be compressed by calling SELrnGeneralizeModel with the same HSEModel (see page 214) model provided as was used in object learning function. Generalization function alters contents of HSEModel (see page 214) model; and as a consequence it contains less redundant information.

The amount of redundant information and the size of the model after generalization may be controlled through a threshold provided to the function. If this value is "0", the most suitable threshold for this model will be calculated and used in generalization process. If value is not zero, this threshold means maximum similarity score of two images that they both still would be kept in the model. Generalization process compares all the images in learnt model with each other and removes those images which are "too similar" to some others. This "too similar" concept means that their similarity score is above a certain threshold.

Notes

Function is depreciated. Use SELrnGeneralizeModelEx (see page 176) instead.

See Also

SELrnAddToModel (see page 174)

Module

SentiSight Module (see page 169)

8.1.6.1.1.7 SELrnGeneralizeModelEx Function

Compresses contents of the HSEModel (see page 214) model.

C++

```
NResult N_API SELrnGeneralizeModelEx(HSEEngine hEngine, HSEModel hModel, SEStatus *
pStatus, HSEModel * pHGenModel);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. This is [in] A handle of HSEEngine type. This is [in] A handle of HSEEngine type. This is the same handle used in object learning function.
HSEModel hModel	[in] A handle of HSEModel (see page 214) type. This model should contain object model aggregated in object learning part. The function changes contents of this model.
SEStatus * pStatus	[out] Generalization status.
HSEModel * pHGenModel	[out] Pointer to generalized model.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Remarks

Object learning function SELrnAddToModel (see page 174) aggregates a model of the object into HSEModel (see page 214) handle. After object learning is complete, the model may be compressed by calling SELrnGeneralizeModelEx with the same HSEModel (see page 214) model provided as was used in object learning function. Generalization function alters contents of HSEModel (see page 214) model; and as a consequence it contains less redundant information.

The amount of redundant information and the size of the model after generalization may be controlled through a threshold provided to the function. If this value is "0", the most suitable threshold for this model will be calculated and used in generalization process. If value is not zero, this threshold means maximum similarity score of two images that they both still would be kept in the model. Generalization process compares all the images in learnt model with each other and removes those images which are "too similar" to some others. This "too similar" concept means that their similarity score is above a certain threshold.

See Also

SELrnAddToModel (see page 174)

Module

SentiSight Module (see page 169)

8.1.6.1.1.8 SELrnRemoveFromModel Function

Removes part of the model represented by *pRefId*.

C++

```
NResult N_API SELrnRemoveFromModel(HSEEngine hEngine, HSEModel hModel, NUInt refId);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. This is the same handle used in object learning function.
HSEModel hModel	[in, out] A handle of HSEModel (see page 214) type. This model should contain object model aggregated in object learning part. The function changes contents of this model.
pRefId	[in] Reference id of added visual information.

Remarks

pRefId is not valid after model is generalized and function behavior is not defined in this case (function will operate incorrectly or even my crash).

Module

SentiSight Module (see page 169)

8.1.6.1.1.9 SEModelClear Function

Clears everything what was added to model without freeing the object.

C++

```
NResult N_API SEModelClear(HSEModel hModel);
```

Parameters

Parameters	Description
HSEModel hModel	[in] A handle of HSEModel (see page 214) type. This model should contain object model aggregated in object learning part. The function changes contents of this model.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hModel* is NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.10 SEModelClone Function

Clones specified HSEModel (see page 214) object.

C++

```
NResult N_API SEModelClone(HSEModel hModel, HSEModel * pHClonedModel);
```

Parameters

Parameters	Description
HSEModel hModel	[in] Handle to HSEModel (see page 214) object to be cloned.
HSEModel * pHClonedModel	[out] A handle to destination source.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

SentiSight Module (see page 169)

8.1.6.1.1.11 SEModelGetSize Function

Returns minimum size of buffer needed to save the model of the object.

C++

```
NResult N_API SEModelGetSize(HSEModel hModel, NSizeType * pSize);
```

Parameters

Parameters	Description
HSEModel hModel	[in] A handle of HSEModel (see page 214) type. The minimum size will be calculated for this model.
NSizeType * pSize	[out] A minimum size of a buffer.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the *hModel* is NULL (see page 82).

Remarks

This function may be called after creation of the model (SECreateModel (see page 173)) and object learning (SELrnAddToModel (see page 174)). The function SEModelGetSize returns a minimum size of a buffer needed to save the model using function SEModelSaveToMemory (see page 181).

See Also

SEModelSaveToMemory (see page 181)

Module

SentiSight Module (see page 169)

8.1.6.1.1.12 SEModelIsEmpty Function

Test whether model contains any information (added to model).

C++

```
NResult N_API SEModelIsEmpty(HSEModel hModel, NBool * pValue);
```

Parameters

Parameters	Description
HSEModel hModel	[in] A handle of HSEModel (see page 214) type.
NBool * pValue	[out] A boolean value indicating whether HSEModel (see page 214) is empty.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the *hModel* is NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.13 SEModelIsLocked Function

Checks if the model is locked.

C++

```
NResult N_API SEModelIsLocked(HSEModel hModel, NBool * pValue);
```

Parameters

Parameters	Description
HSEModel hModel	[in] A handle of HSEModel (see page 214) type.
NBool * pValue	[out] A boolean value indicating whether HSEModel (see page 214) is locked.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the *hModel* is NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.14 SEModelLoadFromMemory Function

Retrieves the model of the object from memory.

C++

```
NResult N_API SEModelLoadFromMemory(HSEModel hModel, const void * pBuffer, NSizeType
bufferSize);
```

Parameters

Parameters	Description
HSEModel hModel	[in] A handle of HSEModel (see page 214) type. This model should be allocated (SEModelCreate) before calling the function.
const void * pBuffer	[in] A data buffer to be read.
NSizeType bufferSize	[in] Size of the data buffer.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the *hModel* is NULL (see page 82).
N_E_OUT_OF_MEMORY (see page 31)	If the function fails, this error can be returned. This error code is returned when a provided buffer is invalid.
N_E_CORE (see page 28)	If the function fails, this error can be returned. This error code is returned when other error occur.

Remarks

A model of HSEModel (see page 214) type must be allocated (using function SECreateModel (see page 173)) before calling SEModelLoadFromMemory. A valid data buffer containing information about the model must be provided to the function. A data buffer may be filled using SEModelSaveToMemory (see page 181) function after object learning is completed.

Module

SentiSight Module (see page 169)

8.1.6.1.1.15 SEModelSaveToMemory Function

Saves the model of the object to memory.

C++

```
NResult N_API SEModelSaveToMemory(HSEModel hModel, void * pBuffer, NSizeType bufferSize);
```

Parameters

Parameters	Description
HSEModel hModel	[in] A handle the model of HSEModel (see page 214) type. It should contain the learnt object data.
void * pBuffer	[out] A data buffer where data of the model will be written. A data buffer must be allocated before calling this function.
NSizeType bufferSize	[in] A size of the data buffer. It should be at least the size returned by SEModelGetSize (see page 178) function.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a HSEModel (see page 214) model is NULL (see page 82).
N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when a buffer is too small to write the data of the model.
N_E_OUT_OF_MEMORY (see page 31)	If the function fails, this error can be returned. This error code is returned when a data corruption error occurred.

Remarks

This function can be called after object learning is complete. Firstly, a required size of a data buffer should be retrieved using SEModelGetSize (see page 178) function. Then a data buffer should be allocated or used previously allocated one. After that, the function SEModelSaveToMemory can be called with the model and the data buffer as parameters. If data buffer is larger than minimum required size, then only first (the number is returned by SEModelGetSize (see page 178) function) bytes of it will be occupied.

Notes

Function is depreciated. Use SEModelSaveToMemoryEx (see page 181) instead.

Module

SentiSight Module (see page 169)

8.1.6.1.1.16 SEModelSaveToMemoryEx Function

Saves the model of the object to memory.

C++

```
NResult N_API SEModelSaveToMemoryEx(HSEModel hModel, void * pBuffer, NSizeType bufferSize,
NSizeType * pSize);
```

Parameters

Parameters	Description
HSEModel hModel	[in] A handle the model of HSEModel (see page 214) type. It should contain the learnt object data.
void * pBuffer	[out] A data buffer where data of the model will be written. A data buffer must be allocated before calling this function.
NSizeType bufferSize	[in] A size of the data buffer. It should be at least the size returned by SEModelGetSize (see page 178) function.
NSizeType * pSize	[out] Pointer to NSizeType (see page 69) that receives size of saved HSEModel (see page 214).

Returns

If the function succeeds, the return value is N_OK (see page 32).

Remarks

This function can be called after object learning is complete. Firstly, a required size of a data buffer should be retrieved using SEModelGetSize (see page 178) function. Then a data buffer should be allocated or used previously allocated one. After that, the function SEModelSaveToMemoryEx can be called with the model and the data buffer as parameters. If data buffer is larger than minimum required size, then only first (the number is returned by SEModelGetSize (see page 178) function) bytes of it will be occupied.

Module

SentiSight Module (see page 169)

8.1.6.1.1.17 SentiSightGetInfo Function

Retrieves information about SentiSight library.

C++

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

Parameters

Parameters	Description
NLibraryInfo * pValue	[out] Pointer to NLibraryInfo structure which contains information about the library.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

SentiSight Module (see page 169)

8.1.6.1.1.18 SERecAddModel Function

Adds a learnt model of the object to the HSEEngine handle.

C++

```
NResult N_API SERecAddModel(HSEEngine hEngine, HSEModel hModel, const void * modelId);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. The same handle will be used in SERecRecognizeImage (see page 191) and SERecRecognizeModel functions.

HSEModel hModel	[in] A handle of HSEModel (see page 214) type. It must contain a learnt model of the object.
modelID	[in] A specified model identifier. Later it will be referred in functions SERecRecognizeImage (see page 191) and SERecRecognizeModel as pBestRecModelID.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a HSentiSightEngine (see page 218) handle or *hModel* are NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when a provided *modelID* is already occupied.
N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when a provided model is empty.

Remarks

Recognition (see page 11) functions SERecRecognizeImage (see page 191) and SERecRecognizeModel compare a test image and a test model with models which were aggregated using SERecAddModel. Each model is of type HSEModel (see page 214); it can be created and filled using object learning functions respectively SECreateModel (see page 173) and SELrnAddToModel (see page 174).

A caller of SERecAddModel must provide an identifier of that model. The identifier will be used in object recognition process. It will be referred as pBestRecModelID in recognition functions and as modelID in SERecGetRecognitionDetails (see page 190) function. Also, this identifier may be used to remove the model by calling SERecRemoveModel (see page 195).

At least one model must be added to the HSentiSightEngine (see page 218) handle before functions SERecRecognizeImage (see page 191) and SERecRecognizeModel are called.

Notes

This function is depreciated. Use SERecAddModelEx (see page 183) instead.

See Also

SERecRecognizeImage (see page 191)

SERecRemoveModel (see page 195)

SERecGetModelCount (see page 189)

Module

SentiSight Module (see page 169)

8.1.6.1.1.19 SERecAddModelEx Function

Adds a learnt model of the object to the HSEEngine handle.

C++

```
NResult N_API SERecAddModelEx(HSEEngine hEngine, HSEModel hModel, const void * modelId,
SEStatus * pStatus);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. The same handle will be used in SERecRecognizeImage (see page 191) and SERecRecognizeModel functions.

HSEModel hModel	[in] A handle of HSEModel (see page 214) type. It must contain a learnt model of the object.
SEStatus * pStatus	[out] Add model status.
modelID	[in] A specified model identifier. Later it will be referred in functions SERecRecognizeImage (see page 191) and SERecRecognizeModelEx as pBestRecModelID.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Remarks

Recognition (see page 11) functions SERecRecognizeImageEx (see page 193) and SERecRecognizeModel compare a test image and a test model with models which were aggregated using SERecAddModelEx. Each model is of type HSEModel (see page 214); it can be created and filled using object learning functions respectively SECreateModel (see page 173) and SELrnAddToModel (see page 174).

A caller of SERecAddModelEx must provide an identifier of that model. The identifier will be used in object recognition process. It will be referred as pBestRecModelID in recognition functions and as modelID in SERecGetRecognitionDetails (see page 190) function. Also, this identifier may be used to remove the model by calling SERecRemoveModel (see page 195).

At least one model must be added to the HSentiSightEngine (see page 218) handle before functions SERecRecognizeImageEx (see page 193) and SERecRecognizeModel are called.

See Also

SERecRecognizeImage (see page 191)

SERecRemoveModel (see page 195)

SERecGetModelCount (see page 189)

Module

SentiSight Module (see page 169)

8.1.6.1.1.20 SERecDetailsGetImageToModelTransform Function

Fills 9 elements 3X3 row wise matrix representing transform. Matrix should be preallocated by user.

C++

```
NResult N_API SERecDetailsGetImageToModelTransform(HSERecognitionDetails
hRecognitionDetails, NDouble * pTransMat);
```

Parameters

Parameters	Description
HSERecognitionDetails hRecognitionDetails	[in] A handle to image recogntion details.
NDouble * pTransMat	[out] Pointer to 3X3 row wise matrix representing transform.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a *hRecognitionDetails* or *pTransMat* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.21 SERecDetailsGetImageToModelTransformEx Function

Fills 9 elements 3X3 row wise matrix representing transform. Matrix should be preallocated by user.

C++

```
NResult N_API SERecDetailsGetImageToModelTransformEx(HSERecognitionDetails
hRecognitionDetails, SERecTransformType transformType, NDouble * pTransMat);
```

Parameters

Parameters	Description
HSERecognitionDetails hRecognitionDetails	[in] A handle to image recogntion details.
SERecTransformType transformType	[in] A recognition transform type.
NDouble * pTransMat	[out] Pointer to 3X3 row wise matrix representing transform.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a *hRecognitionDetails* or *pTransMat* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.22 SERecDetailsGetModelId Function

Returns user specified Id for the model being recognized.

C++

```
NResult N_API SERecDetailsGetModelId(HSERecognitionDetails hRecognitionDetails, const void
** pModelId);
```

Parameters

Parameters	Description
HSERecognitionDetails hRecognitionDetails	[in] A handle to recognition details object.
const void ** pModelId	[in] A pointer to recognition details object.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a *hRecognitionDetails* or *pModeId* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.23 SERecDetailsGetModelToImageTransform Function

Fills 9 element 3X3 row wise matrix representing transform. Matrix should be preallocated by user.

C++

```
NResult N_API SERecDetailsGetModelToImageTransform(HSERecognitionDetails
```

```
hRecognitionDetails, NDouble * pTransMat);
```

Parameters

Parameters	Description
HSERecognitionDetails hRecognitionDetails	[in] A handle to recognition details object.
NDouble * pTransMat	[out] Pointer to 3X3 row wise matrix representing transform.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a *hRecognitionDetails* or *pTransMat* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.24 SERecDetailsGetModelToImageTransformEx Function

Fills 9 element 3X3 row wise matrix representing transform. Matrix should be preallocated by user.

C++

```
NResult N_API SERecDetailsGetModelToImageTransformEx(HSERecognitionDetails
hRecognitionDetails, SERecTransformType transformType, NDouble * pTransMat);
```

Parameters

Parameters	Description
HSERecognitionDetails hRecognitionDetails	[in] A handle to recognition details object.
SERecTransformType transformType	[in] A recognition transform type.
NDouble * pTransMat	[out] Pointer to 3X3 row wise matrix representing transform.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a *hRecognitionDetails* or *pTransMat* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.25 SERecDetailsGetScore Function

Returns similarity score.

C++

```
NResult N_API SERecDetailsGetScore(HSERecognitionDetails hRecognitionDetails, NDouble *
pScore);
```

Parameters

Parameters	Description
HSERecognitionDetails hRecognitionDetails	[in] Recognition (see page 11) details.
NDouble * pScore	[out] Recognition (see page 11) score.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a *hRecognitionDetails* or *pScore* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.26 SERecDetailsGetShape Function

Gets shape describing approximate region of the image where particular model is recognized.

C++

```
NResult N_API SERecDetailsGetShape(HSERecognitionDetails hRecognitionDetails, HSEShape *
pHShape);
```

Parameters

Parameters	Description
HSERecognitionDetails hRecognitionDetails	[in] A handle to approximate region of the image where particular model is recognized.
HSEShape * pHShape	[out] A handle to a shape.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a *hRecognitionDetails* or *pHShape* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.27 SERecDetailsGetShapeEx Function

Gets shape describing approximate region of the image where particular model is recognized.

C++

```
NResult N_API SERecDetailsGetShapeEx(HSERecognitionDetails hRecognitionDetails,
SERecTransformType transformType, HSEShape * pHShape);
```

Parameters

Parameters	Description
HSERecognitionDetails hRecognitionDetails	[in] A handle to approximate region of the image where particular model is recognized.
SERecTransformType transformType	[in] A recognition transform type.
HSEShape * pHShape	[out] A handle to a shape.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a *hRecognitionDetails* or *pHShape* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.28 SERecDetailsGetTransformType Function

Returns type of transform found by recognition process (shape alignment and transform matrices).

C++

```
NResult N_API SERecDetailsGetTransformType(HSERecognitionDetails hRecognitionDetails,
SERecTransformType * pTransformType);
```

Parameters

Parameters	Description
HSERecognitionDetails hRecognitionDetails	[in] A handle to approximate region of the image where particular model is recognized.
SERecTransformType * pTransformType	[out] A pointer to recognition transform type.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a *hRecognitionDetails* or *pTransformType* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.29 SERecGetAllRecognitionDetails Function

Gets all recognition details.

C++

```
NResult N_API SERecGetAllRecognitionDetails(HSEEngine hEngine, HSERecognitionDetails *
arHRecognitionDetails);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle to SentiSight engine type.
HSERecognitionDetails * arHRecognitionDetails	[out] A handle to recognition details type. Must be preallocated and shoud have the size at least SERecGetRecognitionDetailsCount (see page 190).

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a *hEngine* or *arHRecognitionDetails* are NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when the length of shape details is 0.

Module

SentiSight Module (see page 169)

8.1.6.1.1.30 SERecGetModelCount Function

Returns a number of models set to the HSEEngine handle and used in recognition functions.

C++

```
NResult N_API SERecGetModelCount(HSEEngine hEngine, NInt * pCount);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type.
NInt * pCount	[out] Number of models set.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the HSentiSightEngine (see page 218) handle is NULL (see page 82)

Remarks

Recognition (see page 11) functions SERecRecognizeImage (see page 191) and SERecRecognizeModel compare a test image or test model with models which were aggregated using SERecAddModel (see page 182). Function SERecGetModelCount returns a number of aggregated models used in recognition functions.

See Also

SERecAddModel (see page 182)

SERecRemoveModel (see page 195)

SERecRecognizeImage (see page 191)

Module

SentiSight Module (see page 169)

8.1.6.1.1.31 SERecGetModelIds Function

Returns user all Ids for the model being recognized.

C++

```
NResult N_API SERecGetModelIds(HSEEngine hEngine, const void * * arModelIds);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type.
const void * * arModelIds	[out] Memory block which contains models Ids.

Returns

If the function fails an error code is returned. To check if the function failed NFailed (see page 32) macro can be used.

If the function succeeds length of *arModelIds* array is returned. Function succeeds when NSucceeded (see page 32) macro returns result equal or greater than zero.

Module

SentiSight Module (see page 169)

8.1.6.1.1.32 SERecGetRecognitionDetails Function

Returns detailed information about object recognition results.

C++

```
NResult N_API SERecGetRecognitionDetails(HSEEngine hEngine, NInt index,
HSERecognitionDetails * pHRecognitionDetails);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] The HSEEngine handle used in recognition function (SERecRecognizeImage (see page 191) or SERecRecognizeModel).
NInt index	[]in A number from interval [0; SERecGetRecognitionDetailsCount (see page 190)].
HSERecognitionDetails * pHRecognitionDetails	[out] A pointer to recognition details type.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when a *hEngine* or *pHRecognitionDetails* are NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when the length of shape details is 0.
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when an index is less than zero or more than the length of shape details.

Remarks

Recognition (see page 11) details handle is temporary and is valid until image recognition, adding or removing models from recognition engine functions call. Call to this function make sense only just after the call to SERecRecognizeImage (see page 191) with positive result.

Module

SentiSight Module (see page 169)

8.1.6.1.1.33 SERecGetRecognitionDetailsCount Function

Returns recognition details count in HSEEngine.

C++

```
NResult N_API SERecGetRecognitionDetailsCount(HSEEngine hEngine, NInt * pCount);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type.
NInt * pCount	[out] Recognition (see page 11) details count

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

SentiSight Module (see page 169)

8.1.6.1.1.34 SERecGetTrackingImageSize Function

Gets an image size used in object recognition with tracking (SERecRecognizeImage (see page 191)).

C++

```
NResult N_API SERecGetTrackingImageSize(HSEEngine hEngine, NSize * pValue);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A HSEEngine handle which will be used in object recognition process.
NSize * pValue	[out] An image size of type NSize (see page 36).

Returns

If the function succeeds the return value is N_OK (see page 32)

Remarks

The image size of the HSEEngine handle is used in object recognition if SEP_REC_USE_TRACKING (see page 218) tracking is enabled. All the test images used in tracking must be the same size.

The image size can be set using SERecSetTrackingImageSize (see page 195) function.

See Also

SERecSetTrackingImageSize (see page 195)

SERecRecognizeImage (see page 191)

SEP_REC_USE_TRACKING (see page 218)

Module

SentiSight Module (see page 169)

8.1.6.1.1.35 SERecRecognizeImage Function

Object recognition function compares a test image with models which were set to the HSEEngine handle.

C++

```
NResult N_API SERecRecognizeImage(HSEEngine hEngine, HNImage hImage, NDouble
similarityThreshold, NBool * pIsRecognized);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. This is the same handle used in SERecAddModel (see page 182) function.
HNImage hImage	[in] A test image to be recognized. It should be 3 channels RGB image. If tracking is used, the image size must be the same as set to the HSEEngine handle.
NDouble similarityThreshold	[in] A minimum similarity score for object recognition.
NBool * pIsRecognized	[out] Indicates whether something is recognized on this image. To get the number of recognition instances call SERecGetRecognitionDetailsCount (see page 190).

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the *HSentiSightEngine (see page 218)* handle is NULL (see page 82).
N_E_PARAMETER (see page 31)	If the function fails, this error can be returned. This error code is returned when No models are set to the *HSentiSightEngine (see page 218)* handle.
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when tracking is used but image size is not set to the *HSentiSightEngine (see page 218)* handle.
N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when tracking is used and the image size set to the *HSentiSightEngine (see page 218)* handle does not agree with image size provided to the function
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when a templateID of recognized model is larger than maximum number of images in that model.
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when an *hImage* format is invalid.
N_E_FAILED (see page 29)	If the function fails, this error can be returned. This error code is returned when other failure occurred.

Remarks

Object recognition function may be called after one or more models of learnt objects were set to the HSentiSightEngine (see page 218) handle (using SERecAddModel (see page 182)). The models were learnt using object learning function SELrnAddToModel (see page 174).

Function SERecRecognizeImage compares a test image (provided as hImage) with all the models and returns identifier of best recognized model. This id can be passed to SERecGetRecognitionDetails (see page 190) function which provides detailed information about recognition results.

A provided similarityThreshold is used to discard irrelevant recognition information. If actual similarity score of the test image and its nearest model is higher than similarityThreshold, then function SERecGetRecognitionDetails (see page 190) will return meaningful information. If actual similarity score is less than similarityThreshold, the parameter pRecognized of function SERecGetRecognitionDetails (see page 190) will return NFalse (see page 82) and other data is invalid. A value of "0" of similarityThreshold will result in successful recognition in most cases. The indicator of unsuccessful recognition is a parameter pRecognized returned as NFalse (see page 82) value in function SERecGetRecognitionDetails (see page 190).

If parameter SEP_REC_USE_TRACKING (see page 218) is set to NTrue (see page 82) in the HSentiSightEngine (

see page 218) handle (using SESetParameter function), the SERecRecognizeImage function will keep track of the object in a sequence of test images. The parameter SEP_REC_USE_TRACKING (see page 218) should be used only with a sequence of test images where neighbouring images differ only slightly. Object tracking performs better in constant light conditions and constant background. When SERecRecognizeImage function is called for the first time (with tracking switched on) the SERecGetRecognitionDetails (see page 190) function will indicate a successful recognition, i.e. SERecGetRecognitionDetails (see page 190) function will return a parameter pRecognized with value NTrue (see page 82) and other meaningful data. When SERecRecognizeImage function will be called second time and later on (after a successful first time), it will return information only about successful tracking (pBestRecModelID will be the same as the first time and a parameter pTracked in SERecGetRecognitionDetails (see page 190) function returns NTrue (see page 82); other parameters are invalid except of pCenterX and pCenterY). If SERecRecognizeImage function was unable to recognize the object for the first time (pRecognized is NFalse (see page 82)) it will try to do it again in the successive calls of the function.

The tracking is used only when SERecSpeed (see page 216) set to high and HSentiSightEngine (see page 218) has only one learnt model. See: SERecAddModel (see page 182).

During object tracking a parameter similarityThreshold is meaningless. Before calling SERecRecognizeImage function with tracking, it is neccessary to set the size of the test images. The size must be set to the HSentiSightEngine (see page 218) handle using SERecSetTrackingImageSize (see page 195) function.

Notes

Function is depreciated. Use SELrnGeneralizeModelEx (see page 176) instead.

See Also

SERecAddModel (see page 182)

SERecGetRecognitionDetails (see page 190)

SEP_REC_USE_TRACKING (see page 218)

Module

SentiSight Module (see page 169)

8.1.6.1.1.36 SERecRecognizeImageEx Function

Object recognition function compares a test image with models which were set to the HSEEngine handle.

C++

```
NResult N_API SERecRecognizeImageEx(HSEEngine hEngine, HNImage hImage, NBool *
pIsRecognized);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. This is the same handle used in SERecAddModel (see page 182) function.
HNImage hImage	[in] A test image to be recognized. It should be 3 channels RGB image. If tracking is used, the image size must be the same as set to the HSEEngine handle.
NBool * pIsRecognized	[out] Indicates whether something is recognized on this image. To get the number of recognition instances call SERecGetRecognitionDetailsCount (see page 190).

Returns

If the function succeeds, the return value is N_OK (see page 32).

Remarks

Object recognition function may be called after one or more models of learnt objects were set to the HSentiSightEngine (see page 218) handle (using SERecAddModel (see page 182)). The models were learnt using object learning function

SELrnAddToModelEx (see page 175).

Function SERecRecognizeImage (see page 191) compares a test image (provided as hImage) with all the models and returns identifier of best recognized model. This id can be passed to SERecGetRecognitionDetails (see page 190) function which provides detailed information about recognition results.

If parameter SEP_REC_USE_TRACKING (see page 218) is set to NTrue (see page 82) in the HSentiSightEngine (see page 218) handle (using SESetParameter function), the SERecRecognizeImageEx function will keep track of the object in a sequence of test images. The parameter SEP_REC_USE_TRACKING (see page 218) should be used only with a sequence of test images where neighbouring images differ only slightly. Object tracking performs better in constant light conditions and constant background. When SERecRecognizeImage (see page 191) function is called for the first time (with tracking switched on) the SERecGetRecognitionDetails (see page 190) function will indicate a successful recognition, i.e. SERecGetRecognitionDetails (see page 190) function will return a parameter pRecognized with value NTrue (see page 82) and other meaningful data. When SERecRecognizeImageEx function will be called second time and later on (after a successful first time), it will return information only about successful tracking (pBestRecModelID will be the same as the first time and a parameter pTracked in SERecGetRecognitionDetails (see page 190) function returns NTrue (see page 82); other parameters are invalid except of pCenterX and pCenterY). If SERecRecognizeImageEx function was unable to recognize the object for the first time (pRecognized is NFalse (see page 82)) it will try to do it again in the successive calls of the function.

The tracking is used only when SERecSpeed (see page 216) set to high and HSentiSightEngine (see page 218) has only one learnt model. See: SERecAddModelEx (see page 183).

See Also

SERecAddModel (see page 182)

SERecGetRecognitionDetails (see page 190)

SEP_REC_USE_TRACKING (see page 218)

Module

SentiSight Module (see page 169)

8.1.6.1.1.37 SERecRemoveAllModels Function

Removes all models added to the recognition engine.

C++

```
NResult N_API SERecRemoveAllModels(HSEEngine hEngine);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle to SentiSight engine.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the *hEngine* is NULL (see page 82)

Module

SentiSight Module (see page 169)

8.1.6.1.1.38 SERecRemoveModel Function

Removes a model from the HSEEngine handle.

C++

```
NResult N_API SERecRemoveModel(HSEEngine hEngine, const void * modelId);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type.
modelID	[in] A model identifier. It should be the same as provided to SERecAddModel (see page 182) function.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the *hEngine* is NULL (see page 82)
N_E_FAILED (see page 29)	If the function fails, this error can be returned. This error code is returned when a HSEEngine handle is corrupted. Or provided ModelID is not valid.
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when a provided ModelID is not valid.

Remarks

Recognition (see page 11) functions SERecRecognizeImage (see page 191) and SERecRecognizeModel compare a test image or test model with models which were aggregated using SERecAddModel (see page 182). SERecAddModel (see page 182) accepts a modelID identifier from a caller. Some aggregated models can be removed by calling SERecRemoveModel with the same modelID as parameter.

See Also

SERecAddModel (see page 182)

Module

SentiSight Module (see page 169)

8.1.6.1.1.39 SERecSetTrackingImageSize Function

Sets an image size which will be used in object recognition with tracking (SERecRecognizeImage (see page 191)).

C++

```
NResult N_API SERecSetTrackingImageSize(HSEEngine hEngine, const NSize * pValue);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A HSEEngine handle which will be used in object recognition process.
value	[in] An image size of type NSize (see page 36). It must be the same as will be used in object recognition function (SERecRecognizeImage (see page 191)).

Returns

If function succeeds it returns N_OK (see page 32).

Remarks

The image size of the HSEEngine handle is used in object recognition (SERecRecognizeImage (see page 191)) if SEP_REC_USE_TRACKING (see page 218) tracking is enabled. The image size must be set before the process begins, and all the test images must be of that same size.

The image size can be retrieved using SERecGetTrackingImageSize (see page 191) function.

See Also

SERecGetTrackingImageSize (see page 191)

SERecRecognizeImage (see page 191)

SEP_REC_USE_TRACKING (see page 218)

Module

SentiSight Module (see page 169)

8.1.6.1.1.40 SESepAccumulateBackground Function

Accumulates background for foreground/background separation process.

C++

```
NResult N_API SESepAccumulateBackground(HSEEngine hEngine, HNImage hImage, HNImage *
pHBackground, NBool * pNeedMoreFrames);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A HSEEngine handle which stores accumulated background.
HNImage hImage	[in] A handle of HNImage (see page 125) type. It is a current image of background for accumulation. The image must be 3 channel RGB colour image and must have the same size as it was set to the HSEEngine handle (using function SESepSetImageSize (see page 203)).
HNImage * pHBackground	[out] Currently accumulated background.
NBool * pNeedMoreFrames	[out] Specifies, whether more images of background are needed to build accurate background model or not. When its value is NTrue (see page 82), the background accumulation may be stoped.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the HSEEngine handle or provided *hImage* are NULL (see page 82).
N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when the image size of the HSEEngine handle and of hImage do not agree. Or *hImage* data is corrupted.
N_E_FORMAT (see page 29)	If the function fails, this error can be returned. This error code is returned when the *hImage* format is unsupported.
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when the image size is not set to the HSEEngine handle. Or extraction of data from the *hImage* failed.

N_E_FAILED (see page 29)	If the function fails, this error code is returned when occurs other failure.

Remarks

Background images contain some noise, so background accumulation is intended for elimination of that noise. Some images of the same background under the same light conditions should be fed to this function in order to build accurate inner background model. This model will be used later in foreground/background separation (function SESepSeparate (see page 202)). Background can be reset using SESepResetBackgroundModel (see page 199) function.

The image size of the HSEEngine handle must be set (SESepSetImageSize (see page 203)) before calling this function. Furthermore, the image size must be the same as images provided to background accumulation and object separation functions.

See Also

SESepSeparate (see page 202)

SESepResetBackgroundModel (see page 199)

Module

SentiSight Module (see page 169)

8.1.6.1.1.41 SESepGetImageSize Function

Gets an image size of the HSEEngine handle used in foreground/background separation process.

C++

```
NResult N_API SESepGetImageSize(HSEEngine hEngine, NSize * pValue);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A HSEEngine handle which will be used in foreground/background separation process.
NSize * pValue	[out] An image size of type NSize (see page 36).

Returns

If function succeds it returns N_OK (see page 32).

Remarks

The image size of the HSEEngine handle is used in foreground/background separation process (functions starting with SESep). It must be the same in background accumulation (SESepAccumulateBackground (see page 196)), holder learning and object separation (SESepSeparate (see page 202)).

The image size can be set using SESepSetImageSize (see page 203) function.

See Also

SESepSetImageSize (see page 203)

SESepSeparate (see page 202)

SESepAccumulateBackground (see page 196)

Module

SentiSight Module (see page 169)

8.1.6.1.1.42 SESepGetObjectModelSize Function

Returns minimum size of buffer needed to save a colour model of the object.

C++

```
NResult N_API SESepGetObjectModelSize(HSEEngine hEngine, NSizeType * pSize);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. This is the same handle used in foreground/background separation process.
NSizeType * pSize	[out] A minimum size of a buffer to write a colour model.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the HSEEngine handle is NULL (see page 82).
N_E_FAILED (see page 29)	If the function fails, this error can be returned. This error code is returned when memory corruption occurs.

Remarks

Foreground/background separation (function SESepSeparate (see page 202)) builds a colour model of the object. This model can be retrieved using function SESepSaveModelToMemory (see page 201) with provided buffer. Function SESepGetModelSize returns a minimum size of that buffer.

See Also

SESepSaveModelToMemory (see page 201)

Module

SentiSight Module (see page 169)

8.1.6.1.1.43 SESepLoadHolderModelFromMemory Function

Sets a color model of the holder to be used in foreground/background separation process.

C++

```
NResult N_API SESepLoadHolderModelFromMemory(HSEEngine hEngine, const void * pBuffer,
NSizeType bufferSize);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type.
const void * pBuffer	[in] A data buffer containing a color model of the holder.
NSizeType bufferSize	[in] A size of the data buffer.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the HSEEngine handle or *pBuffer* are NULL (see page 82).
N_E_PARAMETER (see page 31)	If the function fails, this error can be returned. This error code is returned when a buffer size is invalid.

N_E_FAILED (see page 29)	If the function fails, this error can be returned. This error code is returned when a buffer data is invalid.

Remarks

Foreground/background separation function SESepSeparate (see page 202) is able to exclude a holder (a hand or a stick which holds the object) from the object and consider it as background.

At the beginning, a colour model of the holder must be learnt. Therefore, a number of images of the holder must be fed to SESepSeparate (see page 202) function which separates the holder from background and builds the colour model of it. The colour model of the holder will be saved in the HSEEngine handle. It can be retrieved from the HSEEngine handle using SESepSaveModelToMemory (see page 201) function with provided data buffer. The colour model can be set as holder using SESepLoadHolderModelFromMemory function with the same data buffer as a parameter.

Notes

Model holder must occupy 50 percent less memory than image.

See Also

SESepSaveModelToMemory (see page 201)

SESepSeparate (see page 202)

SESepResetHolderModel (see page 200)

Module

SentiSight Module (see page 169)

8.1.6.1.1.44 SESepLoadObjectModelFromMemory Function

Sets a color model of the object to be used in foreground/background separation process.

C++

```
NResult N_API SESepLoadObjectModelFromMemory(HSEEngine hEngine, const void * pBuffer,
NSizeType bufferSize);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type.
const void * pBuffer	[in] A data buffer containing a color model of the object.
NSizeType bufferSize	[in] A size of the data buffer.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

SentiSight Module (see page 169)

8.1.6.1.1.45 SESepResetBackgroundModel Function

Resets specified background model.

C++

```
NResult N_API SESepResetBackgroundModel(HSEEngine hEngine);
```

Parameters

Parameters	Description
HSEEngine hEngine	_nt_

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

SentiSight Module (see page 169)

8.1.6.1.1.46 SESepResetHolderModel Function

Resets a holder model used in foreground/background separation process.

C++

```
NResult N_API SESepResetHolderModel(HSEEngine hEngine);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type.

Returns

If function succeeds it returns N_OK (see page 32).

If the HSEEngine handle is NULL (see page 82), function fails and returns N_E_ARGUMENT_NULL (see page 28).

Remarks

Foreground/background separation function SESepSeparate (see page 202) is able to exclude a holder (a hand or a stick which holds the object) from the object and consider it as background. A holder model can be set using SESepLoadHolderModelFromMemory (see page 198) function. If one decided not to use holder, a holder can be removed from the HSEEngine handle by calling SESepResetHolderModel. By default, no holder is set to the HSEEngine handle since the holder must be learnt on the same background and in the same light conditions as the object will be separated.

See Also

SESepSeparate (see page 202)

SESepLoadHolderModelFromMemory (see page 198)

Module

SentiSight Module (see page 169)

8.1.6.1.1.47 SESepResetObjectModel Function

Resets a colour model of the object of foreground/background separation process.

C++

```
NResult N_API SESepResetObjectModel(HSEEngine hEngine);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. This is the same handle used in foreground/background separation process.

Returns

If function succeeds it returns N_OK (see page 32).

If the HSEEngine handle is NULL (see page 82), function fails and returns N_E_ARGUMENT_NULL (see page 28).

Remarks

Foreground/background separation function SESepSeparate (see page 202) builds a colour model of the object. Function SESepResetObjectModel resets the colour model and it becomes empty.

See Also

SESepSeparate (see page 202)

SESepLoadModelFromMemory

Module

SentiSight Module (see page 169)

8.1.6.1.1.48 SESepSaveModelToMemory Function

Writes a colour model of the object into provided buffer.

C++

```
NResult N_API SESepSaveModelToMemory(HSEEngine hEngine, void * pBuffer, NSizeType bufferSize);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. This is the same handle used in foreground/background separation process.
void * pBuffer	[out] A buffer where data of the colour model will be written. A buffer must be allocated before calling the function.
NSizeType bufferSize	[in] A size of the buffer. It should be at least as returned by function SESepGetModelSize.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the HSentiSightEngine (see page 218) handle is NULL (see page 82).
N_E_FAILED (see page 29)	If the function fails, this error can be returned. This error code is returned when the *pBuffer* is too small buffer to save the colour model.

Remarks

Foreground/background separation function SESepSeparate (see page 202) builds a colour model of the object. That model can be saved (using function SESepSaveModelToMemory) and can be loaded (SESepLoadModelFromMemory or SESepLoadHolderModelFromMemory (see page 198) functions) later. A buffer provided to the function SESepSaveModelToMemory should be allocated by the caller. A size of the buffer must be equal or greater than that returned by function SESepGetModelSize. If buffer size is greater than minimum required, then only first (minimum size) bytes will be filled by the function.

See Also

SESepLoadModelFromMemory

SESepLoadHolderModelFromMemory (see page 198)

Module

SentiSight Module (see page 169)

8.1.6.1.1.49 SESepSaveObjectModelToMemory Function

Saves specified object of the model to provided memory buffer.

C++

```
NResult N_API SESepSaveObjectModelToMemory(HSEEngine hEngine, void * pBuffer, NSizeType bufferSize, NSizeType * pSize);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. This is the same handle used in foreground/background separation process.
void * pBuffer	[out] A buffer where data of the object will be written. A buffer must be allocated before calling the function.
NSizeType bufferSize	[in] A size of the buffer. It should be at least as returned by function SESepGetModelSize.
NSizeType * pSize	[out] Value of NSizeType (see page 69).

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

SentiSight Module (see page 169)

8.1.6.1.1.50 SESepSeparate Function

Separates foreground (an object) from background and extracts image mask which can be used in object learning.

C++

```
NResult N_API SESepSeparate(HSEEngine hEngine, HNImage hImage, HNImage * pHMask, HNImage * pHBackground, HNImage * pHSegmented, NBool * pIsForeground);
```

Parameters

Parameters	Description
HSEEngine hEngine	[in] A handle of HSEEngine type. This is the same handle as used in background accumulation.
HNImage hImage	An image of the object to be separated. The image must be 3 channel RGB colour image and must have the same size as it was set to the HSEEngine handle (using functions SESepSetImageSize (see page 203)).
HNImage * pHMask	A grayscale image of the same size as hImage, and containg "0" values in the grid places of background and "255" values in the grid places of the object.
HNImage * pHBackground	An image showing current background (the accumulated one). This image is 3 channel RGB image of the same size as the *hImage* image.
HNImage * pHSegmented	An image with applied mask. The grid places of background are gray; the places of the object present the object itself. This is 3 channel RGB image of the same size as *hImage*.
NBool * pIsForeground	The value NTrue (see page 82) indicates a valid mask, i.e. it contains at least one pixel of the object. The value NFalse (see page 82) indicates that all pixels in the mask are black, i.e. there was no object separated.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).

N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the HSEEngine handle or hImage are NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when the image size is not set to the HSEEngine handle.
N_E_ARGUMENT (see page 27)	If the function fails, this error can be returned. This error code is returned when the image size of the HSEEngine handle does not agree with image size provided to the function. Or *hImage* data is corrupted.
N_E_FAILED (see page 29)	If the function fails, this error can be returned. This error code is returned when memory corruption occurs.

Remarks

This function can be called only after background accumulation (function SESepAccumulateBackground (see page 196)) is finished.

This function separates the object from the background. If a holder model was set (using SESepLoadHolderModelFromMemory (see page 198)) it will be excluded from the object and considered as background. This function also builds a colour model of the object. Later this model can be retrieved calling SESepSaveModelToMemory (see page 201) with provided buffer. This colour model can be reset by calling SESepResetObjectModel (see page 200) or loaded from memory by calling SESepLoadModelFromMemory for appendance.

The image size of the HSEEngine handle must be set (SESepSetImageSize (see page 203)) before calling this function. Furthermore, the image size must be the same as images provided to background accumulation function.

See Also

SESepAccumulateBackground (see page 196)

SESepLoadHolderModelFromMemory (see page 198)

SESepSaveModelToMemory (see page 201)

SESepResetObjectModel (see page 200)

SESepResetHolderModel (see page 200)

SELrnAddToModel (see page 174)

Module

SentiSight Module (see page 169)

8.1.6.1.1.51 SESepSetImageSize Function

Sets an image size of the HSEEngine handle used in foreground/background separation process.

C++

```
NResult N_API SESepSetImageSize(HSEEngine hEngine, const NSize * pValue);
```

Parameters

Parameters	Description
HSEEngine hEngine	A HSEEngine handle which will be used in foreground/background separation process.
value	An image size of type NSize (see page 36). It must be the same as will be used in foreground/background separation.

Returns

If function succeeds it returns N_OK (see page 32).

Function might fail if there is no enough memory to allocate internal structures of the HSEEngine handle. Returned error code is N_E_OUT_OF_MEMORY (see page 31).

Remarks

The image size of the HSEEngine handle is used in foreground/background separation process (functions starting with SESep). The image size must be set before starting this process and images of background, holder and object must be the same size.

The image size can be retrieved using SESepGetImageSize (see page 197) function.

See Also

SESepGetImageSize (see page 197)

SESepAccumulateBackground (see page 196)

SESepSeparate (see page 202)

Module

SentiSight Module (see page 169)

8.1.6.1.1.52 SEShapeAddPoint Function

Adds point to the end of the shape.

C++

```
NResult N_API SEShapeAddPoint(HSEShape hShape, NPointD * pPoint, NBool * pIsPointAdded);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to SEShape type.
NPointD * pPoint	A pointer to a point position.
NBool * pIsPointAdded	A boolean value indicating whether a point was added.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when the *hShape* or *pValue* are NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when a shape is locked.

Notes

Function is depreciated. Use SEShapeAddPointEx (see page 204) instead.

Module

SentiSight Module (see page 169)

8.1.6.1.1.53 SEShapeAddPointEx Function

Adds point to the end of the shape.

C++

```
NResult N_API SEShapeAddPointEx(HSEShape hShape, const NPointD * pPoint);
```

Parameters

Parameters	Description
HSEShape hShape	[in] A handle to SEShape type.

const NPointD * pPoint	[out] A pointer to a point position.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

SentiSight Module (see page 169)

8.1.6.1.1.54 SEShapeClearPoints Function

Clears shape's points without freeing the object.

C++

```
NResult N_API SEShapeClearPoints(HSEShape hShape);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to HSEShape (see page 214) type.

Module

SentiSight Module (see page 169)

8.1.6.1.1.55 SEShapeClone Function

Clones HSEShape (see page 214) object.

C++

```
NResult N_API SEShapeClone(HSEShape hShape, HSEShape * pHClonedShape);
```

Parameters

Parameters	Description
hSrc	A handle to shape object.
pHDst	A handle to destination source.

Module

SentiSight Module (see page 169)

8.1.6.1.1.56 SEShapeCreate Function

Creates new empty shape.

C++

```
NResult N_API SEShapeCreate(HSEShape * pHShape);
```

Parameters

Parameters	Description
HSEShape * pHShape	[out] A handle to created HSEShape (see page 214).

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_OUT_OF_MEMORY (see page 31)	If the function fails, this error can be returned. This error code is returned when *pHShape* NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.57 SEShapeGetCenter Function

Returns automatically calculated shape center point.

C++

```
NResult N_API SEShapeGetCenter(HSEShape hShape, NPointD * pCenter);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape.
NPointD * pCenter	A pointer to shape's center point.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pShape* or *pCenter* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.58 SEShapeGetHeading Function

Gets user specified orientation of the shape in radians 0..2pi.

C++

```
NResult N_API SEShapeGetHeading(HSEShape hShape, NDouble * pHeading);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape.
NDouble * pHeading	[out] A pointer to heading that specifies an orientation of the shape.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pShape* or *pHeading* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.59 SEShapeGetPoint Function

Gets a value of the shape point at specific index.

C++

```
NResult N_API SEShapeGetPoint(HSEShape hShape, NInt index, NPointD * pPoint);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape.
NInt index	An index of a point to be set.
NPointD * pPoint	A pointer to a point to be set.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hShape* is NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when *hShape* has NFalse (see page 82) value.

Module

SentiSight Module (see page 169)

8.1.6.1.1.60 SEShapeGetPointCount Function

Get the number of points in a shape.

C++

```
NResult N_API SEShapeGetPointCount(HSEShape hShape, NInt * pCount);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape where points should be counted.
NInt * pCount	[out] The number of points.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hShape* or *pCount* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.61 SEShapeGetPoints Function

Gets all points from a shape.

C++

```
NResult N_API SEShapeGetPoints(HSEShape hShape, NPointD * arPoints);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to a shape.
NPointD * arPoints	[out] A point to memory buffer that contains all points.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hShape* or *arPoints* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.62 SEShapeInsertPoint Function

Inserts point at a position just before specific index. If new point results in self crossing shape the point is not inserted.

C++

```
NResult N_API SEShapeInsertPoint(HSEShape hShape, NInt index, NPointD * pPoint, NBool *
pIsPointInserted);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape.
NInt index	A specified index where point will be inserted.
NPointD * pPoint	A pointer to NPointD (see page 34) type.
NBool * pIsPointInserted	A pointer to boolean value indicating whether a point was inserted.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pShape* or *pPoint* are NULL (see page 82).
N_E_ARGUMENT_OUT_OF_RANGE (see page 28)	If the function fails, this error can be returned. This error code is returned when *hShape* has NFalse (see page 82) value.
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when index is less than zero.

Notes

This function is depreciated. Use SEShapeInsertPointEx (see page 208) instead.

Module

SentiSight Module (see page 169)

8.1.6.1.1.63 SEShapeInsertPointEx Function

Inserts point at a position just before specific index. If new point results in self crossing shape the point is not inserted.

C++

```
NResult N_API SEShapeInsertPointEx(HSEShape hShape, NInt index, const NPointD * pPoint);
```

Parameters

Parameters	Description
HSEShape hShape	[in] A handle to shape.
NInt index	[in] A specified index where point will be inserted.
const NPointD * pPoint	[out] A pointer to NPointD (see page 34) type.

Returns

If the function succeeds, the return value is N_OK (see page 32).

Module

SentiSight Module (see page 169)

8.1.6.1.1.64 SEShapeIsLocked Function

Checks if the shape is locked.

C++

```
NResult N_API SEShapeIsLocked(HSEShape hShape, NBool * pValue);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to a shape.
NBool * pValue	[out] A boolean value indicating whether a shape is locked.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *pShape* or *pValue* are NULL (see page 82).

Remarks

In this version of SentiSight, shape can be locked only the returned from the SERecDetailsGetShape (see page 187) function. User can not free or modify such shape.

Module

SentiSight Module (see page 169)

8.1.6.1.1.65 SEShapeIsValid Function

Checks if a shape is valid.

C++

```
NResult N_API SEShapeIsValid(HSEShape hShape, NBool * pValue);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape.
NBool * pValue	[out] A boolean value indicating whether a shape is valid.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hShape* is NULL (see page 82).

Remarks

Shape is invalid if it is empty or closing edge (the edge connecting first and last point) is crossing other edges.

Module

SentiSight Module (see page 169)

8.1.6.1.1.66 SEShapeRemovePoint Function

Removes a point from a shape.

C++

```
NResult N_API SEShapeRemovePoint(HSEShape hShape, NInt index, NBool * pIsPointRemoved);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape.
NInt index	Index of point to remove.
NBool * pIsPointRemoved	[out] A boolean value which indicates whether the point was removed.

Returns

If function succeeds it returns N_OK (see page 32).

Remarks

If operation results in a self crossing shape the point is not removed.

Notes

This function is depreciated. Use SEShapeRemovePointEx (see page 210) instead.

Module

SentiSight Module (see page 169)

8.1.6.1.1.67 SEShapeRemovePointEx Function

Removes a point from a shape.

C++

```
NResult N_API SEShapeRemovePointEx(HSEShape hShape, NInt index);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape.
NInt index	Index of point to remove.

Returns

If function succeeds it returns N_OK (see page 32).

Remarks

If operation results in a self crossing shape the point is not removed.

Module

SentiSight Module (see page 169)

8.1.6.1.1.68 SEShapeRotate Function

Rotates shape by amount of radians around the center of the shape.

C++

```
NResult N_API SEShapeRotate(HSEShape hShape, NDouble angle);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to a shape which should be rotated.
NDouble angle	An angle in radians to rotate a shape.

Returns

If function succeeds it returns N_OK (see page 32).

Module

SentiSight Module (see page 169)

8.1.6.1.1.69 SEShapeScale Function

Scales shape around the center of the shape, below 1 shrinks, above 1 enlarges the shape.

C++

```
NResult N_API SEShapeScale(HSEShape hShape, NDouble scale);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape which should be scaled.
NDouble scale	Shape scale.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hShape* is NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when *hShape* has NFalse (see page 82) value.

Module

SentiSight Module (see page 169)

8.1.6.1.1.70 SEShapeSetHeading Function

Sets user specified orientation of the shape in radians 0..2pi.

C++

```
NResult N_API SEShapeSetHeading(HSEShape hShape, NDouble heading);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape.
NDouble heading	A heading to be set.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hShape* is NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when *hShape* has NFalse (see page 82) value.

Module

SentiSight Module (see page 169)

8.1.6.1.1.71 SEShapeSetPoint Function

Changes value of the shape point at specific index.

C++

```
NResult N_API SEShapeSetPoint(HSEShape hShape, NInt index, NPointD * pPoint, NBool *
pIsPointSet);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape.
NInt index	An index of a point to be set.
NPointD * pPoint	A pointer to a point to be set.
NBool * pIsPointSet	[out] A boolean value indicating whether a point was set.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hShape* is NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when *hShape* has NFalse (see page 82) value.

Notes

This function is depreciated. Use SEShapeSetPointEx (see page 212) instead.

Module

SentiSight Module (see page 169)

8.1.6.1.1.72 SEShapeSetPointEx Function

Changes value of the shape point at specific index.

C++

```
NResult N_API SEShapeSetPointEx(HSEShape hShape, NInt index, const NPointD * pPoint);
```

Parameters

Parameters	Description
HSEShape hShape	[in] A handle to shape.
NInt index	[in] An index of a point to be set.
const NPointD * pPoint	[out] A pointer to a point to be set.

Returns

If function succeeds it returns N_OK (see page 32).

Module

SentiSight Module (see page 169)

8.1.6.1.1.73 SEShapeTestPoint Function

Gets test point and shape relative position.

C++

```
NResult N_API SEShapeTestPoint(HSEShape hShape, const NPointD * pPoint, NBool
measureDistance, NDouble * pDistance);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to a shape.
const NPointD * pPoint	A pointer to a point which should be tested.
NBool measureDistance	If measureDistance is false distance is -1, 0, 1. If measureDistance is true distance is negative, zero, positive Euclidean distance, for point being outside, on the edge and inside of the shape respectively.
NDouble * pDistance	[out] A measured Euclidean distance.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hShape*, *pPoint* or *pDistance* are NULL (see page 82).

Module

SentiSight Module (see page 169)

8.1.6.1.1.74 SEShapeTranslate Function

Translates the shape. Negative distance is translated to the coordinate origin side, positive - out of the coordinate origin.

C++

```
NResult N_API SEShapeTranslate(HSEShape hShape, const NPointD * pDistance);
```

Parameters

Parameters	Description
HSEShape hShape	A handle to shape that should be translated.

const NPointD * pDistance	[out] A distance to translated shape.

Returns

Return values	Description
N_OK (see page 32)	If the function succeeds, the return value is N_OK (see page 32).
N_E_ARGUMENT_NULL (see page 28)	If the function fails, this error can be returned. This error code is returned when *hShape* or *pDistance* are NULL (see page 82).
N_E_INVALID_OPERATION (see page 29)	If the function fails, this error can be returned. This error code is returned when *hShape* has NFalse (see page 82) value.

Module

SentiSight Module (see page 169)

8.1.6.1.1.75 HSEModel

A handle of the object model. It is used in object learning and object recognition functions.

Remarks

This handle should be created (using SECreateModel (see page 173)) first. Then, during object learning part (function SELrnAddToModelEx (see page 175)) the information of the object is added to this model. After object learning finished, the model may be compressed (using SELrnGeneralizeModelEx (see page 176)), saved to memory (function SEModelSaveToMemoryEx (see page 181)) and loaded from memory later (function SEModelLoadFromMemory (see page 180)). When the model is not needed anymore it must be released by calling SEModelFree function.

This SentiSight engine model will be used in object recognition functions as one of reference model set to HSentiSightEngine (see page 218) handle (using SERecAddModel (see page 182) function). Also this HSentiSightEngine (see page 218) model can be used as a test model in SERecRecognizeModel function.

Module

SentiSight Module (see page 169)

8.1.6.1.1.76 HSERecognitionDetails

Recognition (see page 11) details handle are temporary and are only valid in image recognition, adding or removing models from recognition engine functions calls.

See Also

SERecDetailsIsTracked (see page 174)

SERecDetailsGetImageToModelTransformEx (see page 185)

SERecDetailsGetModelId (see page 185)

Module

SentiSight Module (see page 169)

8.1.6.1.1.77 HSEShape

Handle to shape.

Remarks

Shapes are created using SEShapeCreate (see page 205) function. Also created shape can be cloned using SEShapeClone (see page 205) function. For more information about shapes read section Object Learning (see page 7).

Module

SentiSight Module (see page 169)

8.1.6.1.2 Structs, Records, Enums

8.1.6.1.2.1 SEStatus Enumeration

Enumerates different values of SentiShight learning and recognition processes.

C++

```
typedef enum SEStatus_ {
  sesSucceeded = 1,
  sesMaskIsEmpty = 2,
  sesShapeNotValid = 3,
  sesNothingToLearn = 4,
  sesModelIsEmpty = 5
} SEStatus;
```

Members

Members	Description
sesSucceeded = 1	Indicates that operation succeeded.
sesMaskIsEmpty = 2	Indicates that model was not updated with new information because passed mask is empty, all zero.
sesShapeNotValid = 3	Indicates that model was not updated with new information because passed shape is not valid.
sesNothingToLearn = 4	Indicates that no data to learned was indicated.
sesModelIsEmpty = 5	Indicates that model contain no objects.

Module

SentiSight Module (see page 169)

8.1.6.1.2.2 SELrnMode Enumeration

Enumerates different type of learning parameters.

C++

```
typedef enum SELrnMode_ {
  selmLowProfile = 0,
  selmHighProfile = 1,
  selmHighProfileEx = 2
} SELrnMode;
```

Members

Members	Description
selmLowProfile = 0	Adds additional not rotation invariant information, improves recognition quality for not rotated objects (has no impact on rotated object recognition). About 5%-10% slower and has about 20%-60% bigger template size (this information is used with low recognition speed only).
selmHighProfile = 1	Fastest and has the smallest template size, suitable in most situations.
selmHighProfileEx = 2	Adds additional information, improves recognition for all types of objects. About 50%-100% slower and has about 5%-10% bigger template size (this information is used with low recognition speed only).

Module

SentiSight Module (see page 169)

8.1.6.1.2.3 SERecSpeed Enumeration

Enumerates SentiSight recognition speed parameters.

C++

```
typedef enum SERecSpeed_ {
  sersLow = 0,
  sersHigh = 256
} SERecSpeed;
```

Members

Members	Description
sersLow = 0	The lowest recognition speed.
sersHigh = 256	The highest recognition speed.

Module

SentiSight Module (see page 169)

8.1.6.1.2.4 SERecTransformType Enumeration

Transform types used in recognition process to align shape or transform images.

C++

```
typedef enum SERecTransformType_ {
  serttAuto = 0,
  serttPespective = 1,
  serttAffine = 2,
  serttSimilarity = 3
} SERecTransformType;
```

Members

Members	Description
serttAuto = 0	The slowest transformation type used in a recognition process.
serttPespective = 1	The slowest transformation type used in a recognition process.
serttAffine = 2	This type in the current version of SentiSight means Similarity.
serttSimilarity = 3	The fastest transformation type used in a recognition process.

Remarks

If auto, selects best posible transform, if particular transform is selected but current condition do not allow to perform it simple transorm is selected. Transform complexity: Similarity-\>Affine-\>Perspective.

Module

SentiSight Module (see page 169)

8.1.6.1.3 Macros

To get a parameter value, use NObjectGetParameter (see page 47) function. To set parameter value - NObjectSetParameter (see page 48) function.

Macros

Name	Description
SEP_LRN_ENHANCE_MASK (see page 217)	Identifier specifying whether to extend image mask during object learning or not.
SEP_LRN_GENERALIZATION_THRESHOLD (see page 217)	Represents Id for generalization threshold parameter.
SEP_LRN_MODE (see page 217)	Represents Id for learning mode parameter.
SEP_REC_SPEED (see page 217)	Represents Id for SentiSight recognition speed parameter.
SEP_REC_THRESHOLD (see page 218)	Represents Id for recognition threshold parameter.
SEP_REC_TRANSFORM_TYPE (see page 218)	Represents Id for transform type parameter to be used in recognition to align shape.
SEP_REC_USE_TRACKING (see page 218)	Represents Id for use tracking for recognition parameter.
SEP_SEP_USE_ADAPTIVE_ALG (see page 218)	Represents Id for a parameter to use an adaptive algorithm for separation.

Module

SentiSight Module (see page 169)

8.1.6.1.3.1 SEP_LRN_ENHANCE_MASK Macro

Identifier specifying whether to extend image mask during object learning or not.

C++

```
#define SEP_LRN_ENHANCE_MASK 50100
```

Remarks

This parameter is used during object learning part by calling SELrnAddToModel (see page 174) function. Its values are of type NBool (see page 66). It is valid only if an image mask is provided to the function. If the parameter is set to NTrue (see page 82), the image mask is extended to eliminate noise, sharp corners and accidental halls in it. If parameter is set to NFalse (see page 82), the image mask is used as it is provided to the function SELrnAddToModel (see page 174).

The default value for this parameter is NFalse (see page 82), thus the image mask is used as it is provided to the object learning function.

See Also

SELrnAddToModel (see page 174)

8.1.6.1.3.2 SEP_LRN_GENERALIZATION_THRESHOLD Macro

Represents Id for generalization threshold parameter.

C++

```
#define SEP_LRN_GENERALIZATION_THRESHOLD 50102
```

8.1.6.1.3.3 SEP_LRN_MODE Macro

Represents Id for learning mode parameter.

C++

```
#define SEP_LRN_MODE 50101
```

8.1.6.1.3.4 SEP_REC_SPEED Macro

Represents Id for SentiSight recognition speed parameter.

C++

```
#define SEP_REC_SPEED 50201
```

8.1.6.1.3.5 SEP_REC_THRESHOLD Macro

Represents Id for recognition threshold parameter.

C++

```
#define SEP_REC_THRESHOLD 50203
```

8.1.6.1.3.6 SEP_REC_TRANSFORM_TYPE Macro

Represents Id for transform type parameter to be used in recognition to align shape.

C++

```
#define SEP_REC_TRANSFORM_TYPE 50202
```

8.1.6.1.3.7 SEP_REC_USE_TRACKING Macro

Represents Id for use tracking for recognition parameter.

C++

```
#define SEP_REC_USE_TRACKING 50200
```

8.1.6.1.3.8 SEP_SEP_USE_ADAPTIVE_ALG Macro

Represents Id for a parameter to use an adaptive algorithm for separation.

C++

```
#define SEP_SEP_USE_ADAPTIVE_ALG 50000
```

8.1.6.1.4 Types

8.1.6.1.4.1 HSentiSightEngine Type

The main handle of HSentiSightEngine library.

C++

```
typedef HSEEngine HSentiSightEngine;
```

Remarks

This handle is used in major part of HSentiSightEngine library functions. It should be allocated once by calling SECreate (see page 172) function and used the whole application. It keeps information about foreground/background separation, object learning and object recognition functions. At the end of the application the handle must be released using SEFree function.

This HSentiSightEngine model will be used in object recognition functions as one of reference model set to HSentiSightEngine handle (using SERecAddModel (see page 182) function). Also this HSentiSightEngine model can be used as a test model in SERecRecognizeModel function.

Module

SentiSight Module (see page 169)

8.1.6.1.4.2 SEModelUpdateStatus Type

Enumerates different types of SentiSight model update status parameters.

C++

```
typedef SEStatus SEModelUpdateStatus;
```

Module

SentiSight Module (see page 169)

8.2 .NET

Namespaces

Name	Description
Neurotec (see page 219)	Contains classes that provide infrastructure for Neurotechnology components.
Neurotec.Images (see page 241)	Classes under this namespace provides functionality that enable loading, saving and converting images in various formats.
Neurotec.SentiSight (see page 272)	General namespace of SentiSight (see page 296) library classes.
Neurotec.DeviceManager (see page 305)	Classes under this namespace provides functionality for working with cameras.
Neurotec.Video (see page 317)	Provides functionality for reading video files.
Neurotec.Licensing (see page 322)	Provides functionality for getting, releasing licenses.
Neurotec.IO (see page 330)	Classes under this namespace provides infrastructure for Neurotechnology components.

8.2.1 Neurotec Namespace

Contains classes that provide infrastructure for Neurotechnology components.

Module

.NET (see page 219)

Classes

	Name	Description
S	NCore (see page 220)	This class supports internal Neurotechnology libraries infrastructure and should not be used directly in your code.
A	NDisposable (see page 227)	Provides a method to release allocated resources.
	NeurotecException (see page 227)	The exception that is thrown when unknown error occurred in one of Neurotechnology libraries.
A	NeurotecExceptionBase (see page 228)	The exception that is thrown when unknown error occurred in one of Neurotechnology libraries.
	NLibraryInfo (see page 229)	Provides definitions of library info class.
A	NObject (see page 231)	Provides functionality for retrieving information about specified object.
	NotActivatedException (see page 233)	The exception that is thrown when one of required Neurotechnology libraries is not activated.
S	NResult (see page 234)	Provides functionality for unmanaged functions error handling.

Interfaces

	Name	Description
	INeurotecException (see page 240)	The interface that provides information about Neurotec exceptions.

Structs, Records, Enums

	Name	Description
	NProcessorVendor (see page 240)	Specifies the processor's vendor name.

Types

Name	Description
NLibraryGetInfo (see page 241)	This delegate supports internal Neurotechnology libraries infrastructure and should not be used directly in your code.

8.2.1.1 Classes

8.2.1.1.1 NCore Class

This class supports internal Neurotechnology libraries infrastructure and should not be used directly in your code.

C#

```
public static class NCore;
```

NCore Fields

	Name	Description
	DllName (see page 220)	NCore (see page 24) library name.

NCore Methods

	Name	Description
S	Alloc (see page 221)	Allocates a block of memory.
S	CAlloc (see page 221)	Allocates an array in memory with elements initialized to 0.
S	Clear (see page 222)	Clears all elements in the specified array.
S	Compare (see page 222)	Compares two memory blocks.
S	Copy (see page 223)	Copies data from one memory block to other.
S	Fill (see page 223)	Sets memory block to a specified value.
S	Free (see page 224)	Frees a specified memory block.
S	GetInfo (see page 224)	Retrieves an information about NCore (see page 24) library.
S	Move (see page 224)	Moves one memory block to another.
S	PtrToArray (see page 225)	Allocates a memory block for an array specified by pointer.
S	PtrToStructureArray (see page 226)	Allocates a memory block for a array structure specified by pointer.
S	ReAlloc (see page 226)	Reallocate memory blocks.
S	WriteBufferToStream (see page 227)	Writes memory buffer to stream.

8.2.1.1.1.1 NCore Fields

8.2.1.1.1.1.1 NCore.DllName Field

NCore (see page 24) library name.

C#

```
public const string DllName = "NCore";
```

8.2.1.1.1.2 NCore Methods

8.2.1.1.1.2.1 Alloc Method

8.2.1.1.1.2.1.1 NCore.Alloc Method (uint)

Allocates a block of memory.

C#

```
public static IntPtr Alloc(uint size);
```

Parameters

Parameters	Description
uint size	Size in bytes of memory to be allocated.

Returns

A representation of a pointer (IntPtr type) to the allocated memory block.

8.2.1.1.1.2.1.2 NCore.Alloc Method (ulong)

Allocates a block of memory.

C#

```
public static IntPtr Alloc(ulong size);
```

Parameters

Parameters	Description
ulong size	Size in bytes of memory to be allocated.

Returns

A representation of a pointer (IntPtr type) to the allocated memory block.

8.2.1.1.1.2.2 CAlloc Method

8.2.1.1.1.2.2.1 NCore.CAlloc Method (uint)

Allocates an array in memory with elements initialized to 0.

C#

```
public static IntPtr CAlloc(uint size);
```

Parameters

Parameters	Description
uint size	Size in bytes of memory to be allocated.

Returns

Pointer to allocated memory block.

8.2.1.1.1.2.2.2 NCore.CAlloc Method (ulong)

Allocates an array in memory with elements initialized to 0.

C#

```
public static IntPtr CAlloc(ulong size);
```

Parameters

Parameters	Description
ulong size	Size in bytes of memory to be allocated.

Returns

Pointer to allocated memory block.

8.2.1.1.1.2.3 Clear Method

8.2.1.1.1.2.3.1 NCore.Clear Method (IntPtr, uint)

Clears all elements in the specified array.

C#

```
public static void Clear(IntPtr pBlock, uint size);
```

Parameters

Parameters	Description
IntPtr pBlock	A pointer to memory block that contains an array.
uint size	The size in bytes of an array.

8.2.1.1.1.2.3.2 NCore.Clear Method (IntPtr, ulong)

Clears all elements in the specified array.

C#

```
public static void Clear(IntPtr pBlock, ulong size);
```

Parameters

Parameters	Description
IntPtr pBlock	A pointer to memory block that contains an array.
ulong size	The size in bytes of an array.

8.2.1.1.1.2.4 Compare Method

8.2.1.1.1.2.4.1 NCore.Compare Method (IntPtr, IntPtr, uint)

Compares two memory blocks.

C#

```
public static int Compare(IntPtr pBlock1, IntPtr pBlock2, uint size);
```

Parameters

Parameters	Description
IntPtr pBlock1	A pointer to the first memory block.
IntPtr pBlock2	A pointer to the second memory block.
uint size	Number of characters to be compared.

Returns

The value indicating the relationship between the memory blocks.

If return value is less than zero a block specified by pBlock1 is less than a block specified by pBlock2.

if return value is zero a block specified by pBlock1 is identical to a block specified by pBlock2.

If return value is greater than zero a block specified by pBlock1 is greater than a block specified by pBlock2.

8.2.1.1.1.2.4.2 NCore.Compare Method (IntPtr, IntPtr, ulong)

Compares two memory blocks.

C#

```
public static int Compare(IntPtr pBlock1, IntPtr pBlock2, ulong size);
```

Parameters

Parameters	Description
IntPtr pBlock1	A pointer to the first memory block.
IntPtr pBlock2	A pointer to the second memory block.
ulong size	Number of characters to be compared.

Returns

The value indicating the relationship between the memory blocks.

If return value is less than zero a block specified by pBlock1 is less than a block specified by pBlock2.

if return value is zero a block specified by pBlock1 is identical to a block specified by pBlock2.

If return value is greater than zero a block specified by pBlock1 is greater than a block specified by pBlock2.

8.2.1.1.1.2.5 Copy Method

8.2.1.1.1.2.5.1 NCore.Copy Method (IntPtr, IntPtr, uint)

Copies data from one memory block to other.

C#

```
public static void Copy(IntPtr pDstBlock, IntPtr pSrcBlock, uint size);
```

Parameters

Parameters	Description
IntPtr pDstBlock	A pointer to destination memory block (a memory block where the data from source memory block will be copied).
IntPtr pSrcBlock	A pointer to memory block to be copied to another memory block.
uint size	The size of memory block to be copied (the size is in bytes).

8.2.1.1.1.2.5.2 NCore.Copy Method (IntPtr, IntPtr, ulong)

Copies data from one memory block to other.

C#

```
public static void Copy(IntPtr pDstBlock, IntPtr pSrcBlock, ulong size);
```

Parameters

Parameters	Description
IntPtr pDstBlock	A pointer to destination memory block (a memory block where the data from source memory block will be copied).
IntPtr pSrcBlock	A pointer to memory block to be copied to another memory block.
ulong size	The size of memory block to be copied (the size is in bytes).

8.2.1.1.1.2.6 Fill Method

8.2.1.1.1.2.6.1 NCore.Fill Method (IntPtr, byte, uint)

Sets memory block to a specified value.

C#

```
public static void Fill(IntPtr pBlock, byte value, uint size);
```

Parameters

Parameters	Description
IntPtr pBlock	A pointer to a destination memory block that contains data.
byte value	A value to set.
uint size	The size in bytes of memory block to be set.

8.2.1.1.1.2.6.2 NCore.Fill Method (IntPtr, byte, ulong)

Sets memory block to a specified value.

C#

```
public static void Fill(IntPtr pBlock, byte value, ulong size);
```

Parameters

Parameters	Description
IntPtr pBlock	A pointer to a destination memory block that contains data.
byte value	A value to set.
ulong size	The size in bytes of memory block to be set.

8.2.1.1.1.2.7 NCore.Free Method

Frees a specified memory block.

C#

```
public static void Free(IntPtr pBlock);
```

Parameters

Parameters	Description
IntPtr pBlock	A pointer to previously allocated memory block to be freed.

Remarks

The Free method frees a memory block that was previously allocated by a call to Alloc (see page 221), CAlloc (see page 221) or ReAlloc (see page 226) methods. The number of freed bytes is equivalent to the number of bytes requested when the block was allocated (or reallocated, in the case of ReAlloc (see page 226)).

8.2.1.1.1.2.8 NCore.GetInfo Method

Retrieves an information about NCore (see page 24) library.

C#

```
public static NLibraryInfo GetInfo();
```

Returns

NLibraryInfo (see page 229) object.

8.2.1.1.1.2.9 Move Method

8.2.1.1.1.2.9.1 NCore.Move Method (IntPtr, IntPtr, uint)

Moves one memory block to another.

C#

```
public static void Move(IntPtr pDstBlock, IntPtr pSrcBlock, uint size);
```

Parameters

Parameters	Description
IntPtr pDstBlock	A pointer to destination memory block.
IntPtr pSrcBlock	A pointer to source memory block.
uint size	Size in bytes to be copied.

Remarks

Copies size bytes from pDstBlock to pSrcBlock.

Make sure that the destination buffer is the same size or larger than the source buffer.

8.2.1.1.1.2.9.2 NCore.Move Method (IntPtr, IntPtr, ulong)

Moves one memory block to another.

C#

```
public static void Move(IntPtr pDstBlock, IntPtr pSrcBlock, ulong size);
```

Parameters

Parameters	Description
IntPtr pDstBlock	A pointer to destination memory block.
IntPtr pSrcBlock	A pointer to source memory block.
ulong size	Size in bytes to be copied.

Remarks

Copies size bytes from pDstBlock to pSrcBlock.

Make sure that the destination buffer is the same size or larger than the source buffer.

8.2.1.1.1.2.10 PtrToArray Method

8.2.1.1.1.2.10.1 NCore.PtrToArray Method (IntPtr, int)

Allocates a memory block for an array specified by pointer.

C#

```
public static byte[] PtrToArray(IntPtr pBuffer, int bufferLength);
```

Parameters

Parameters	Description
IntPtr pBuffer	A pointer to memory block that contains the data to be allocated as an array.
int bufferLength	The length in bytes of memory buffer.

Returns

An array that contains data from memory block.

8.2.1.1.1.2.10.2 NCore.PtrToArray Method (IntPtr, long)

Allocates a memory block for an array specified by pointer.

C#

```
public static byte[] PtrToArray(IntPtr pBuffer, long bufferLength);
```

Parameters

Parameters	Description
IntPtr pBuffer	A pointer to memory block that contains the data to be allocated as an array.
long bufferLength	The length in bytes of memory buffer.

Returns

An array that contains data from memory block.

8.2.1.1.1.2.11 NCore.PtrToStructureArray Method

Allocates a memory block for a array structure specified by pointer.

C#

```
public static Array PtrToStructureArray(IntPtr pValues, uint count, Type type);
```

Parameters

Parameters	Description
IntPtr pValues	A pointer to memory block that contains data.
uint count	Number of items to be allocated as a string data type array.
Type type	Represents type declaration.

Returns

An array that contains data copied from memory block.

8.2.1.1.1.2.12 ReAlloc Method

8.2.1.1.1.2.12.1 NCore.ReAlloc Method (IntPtr, uint)

Reallocate memory blocks.

C#

```
public static void ReAlloc(ref IntPtr pBlock, uint size);
```

Parameters

Parameters	Description
ref IntPtr pBlock	Pointer to memory block to be allocated.
uint size	Size in bytes of new memory block.

8.2.1.1.1.2.12.2 NCore.ReAlloc Method (IntPtr, ulong)

Reallocate memory blocks.

C#

```
public static void ReAlloc(ref IntPtr pBlock, ulong size);
```

Parameters

Parameters	Description
ref IntPtr pBlock	Pointer to memory block to be allocated.
ulong size	Size in bytes of new memory block.

8.2.1.1.1.2.13 WriteBufferToStream Method

8.2.1.1.1.2.13.1 NCore.WriteBufferToStream Method (IntPtr, int, Stream)

Writes memory buffer to stream.

C#

```
public static void WriteBufferToStream(IntPtr pBuffer, int bufferLength, Stream stream);
```

Parameters

Parameters	Description
IntPtr pBuffer	A pointer to memory buffer.
int bufferLength	Length of the buffer.
Stream stream	The stream where data should be written.

8.2.1.1.1.2.13.2 NCore.WriteBufferToStream Method (IntPtr, long, Stream)

Writes memory buffer to stream.

C#

```
public static void WriteBufferToStream(IntPtr pBuffer, long bufferLength, Stream stream);
```

Parameters

Parameters	Description
IntPtr pBuffer	A pointer to memory buffer.
long bufferLength	Length of the buffer.
Stream stream	The stream where data should be written.

8.2.1.1.2 NDisposable Class

Provides a method to release allocated resources.

C#

```
public abstract class NDisposable : MarshalByRefObject, IDisposable;
```

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

8.2.1.1.2.1 NDisposable Methods

8.2.1.1.2.1.1 NDisposable.Dispose Method

Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

C#

```
public void Dispose();
```

Remarks

This method is used to close or release unmanaged resources. By convention, this method is used for all tasks associated with freeing resources held by an object.

8.2.1.1.3 NeurotecException Class

The exception that is thrown when unknown error occurred in one of Neurotechnology libraries.

C#

```
public sealed class NeurotecException : NeurotecExceptionBase;
```

INeurotecException Properties

	Name	Description
	Code (see page 240)	Gets an error code of the current exception.
	ManagedStackTrace (see page 240)	Gets a string representation of the frames on the managed stack at the time the current exception was thrown.
	UnmanagedStackTrace (see page 240)	Gets a string representation of the frames on the unmanaged stack at the time the current exception was thrown.

NeurotecExceptionBase Class

	Name	Description
	Code (see page 228)	Gets an error code of the current exception.
	ManagedStackTrace (see page 229)	Gets a string representation of the frames on the managed stack at the time the current exception was thrown.
	StackTrace (see page 229)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
	UnmanagedStackTrace (see page 229)	Gets a string representation of the frames on the unmanaged stack at the time the current exception was thrown.

8.2.1.1.4 NeurotecExceptionBase Class

The exception that is thrown when unknown error occurred in one of Neurotechnology libraries.

C#

```
public abstract class NeurotecExceptionBase : SystemException, INeurotecException;
```

INeurotecException Properties

	Name	Description
	Code (see page 240)	Gets an error code of the current exception.
	ManagedStackTrace (see page 240)	Gets a string representation of the frames on the managed stack at the time the current exception was thrown.
	UnmanagedStackTrace (see page 240)	Gets a string representation of the frames on the unmanaged stack at the time the current exception was thrown.

NeurotecExceptionBase Class

	Name	Description
	Code (see page 228)	Gets an error code of the current exception.
	ManagedStackTrace (see page 229)	Gets a string representation of the frames on the managed stack at the time the current exception was thrown.
	StackTrace (see page 229)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
	UnmanagedStackTrace (see page 229)	Gets a string representation of the frames on the unmanaged stack at the time the current exception was thrown.

8.2.1.1.4.1 NeurotecExceptionBase Properties

8.2.1.1.4.1.1 NeurotecExceptionBase.Code Property

Gets an error code of the current exception.

C#

```
public int Code;
```

Property value

Error code.

8.2.1.1.4.1.2 NeurotecExceptionBase.ManagedStackTrace Property

Gets a string representation of the frames on the managed stack at the time the current exception was thrown.

C#

```
public string ManagedStackTrace;
```

Property value

String that describses the contents of the managed stack.

8.2.1.1.4.1.3 NeurotecExceptionBase.StackTrace Property

Gets a string representation of the frames on the call stack at the time the current exception was thrown.

C#

```
public override string StackTrace;
```

Property value

String that describes the contents of the call stack.

8.2.1.1.4.1.4 NeurotecExceptionBase.UnmanagedStackTrace Property

Gets a string representation of the frames on the unmanaged stack at the time the current exception was thrown.

C#

```
public string UnmanagedStackTrace;
```

Property value

String that describes the contents of the unmanaged stack.

8.2.1.1.5 NLibraryInfo Class

Provides definitions of library info class.

C#

```
public sealed class NLibraryInfo;
```

NLibraryInfo Methods

	Name	Description
	Retrieve (see page 230)	Retrieves information about this library.

NLibraryInfo Properties

	Name	Description
	Activated (see page 230)	Checks if the library is registered.
	Company (see page 230)	Retrieves company name.
	Copyright (see page 230)	Retrieves copyright notice of this library.
	Product (see page 230)	Retrieves product name.
	Title (see page 230)	Retrieves library's title.
	Version (see page 230)	Retrieves library version number.

8.2.1.1.5.1 NLibraryInfo Methods

8.2.1.1.5.1.1 NLibraryInfo.Retrieve Method

Retrieves information about this library.

C#

```
public static NLibraryInfo Retrieve(NLibraryGetInfo getInfo);
```

Parameters

Parameters	Description
NLibraryGetInfo getInfo	NLibraryGetInfo (see page 241) object.

Returns

NLibraryInfo (see page 229) object which contains information about this library.

8.2.1.1.5.2 NLibraryInfo Properties

8.2.1.1.5.2.1 NLibraryInfo.Activated Property

Checks if the library is registered.

C#

```
public string Activated;
```

8.2.1.1.5.2.2 NLibraryInfo.Company Property

Retrieves company name.

C#

```
public string Company;
```

8.2.1.1.5.2.3 NLibraryInfo.Copyright Property

Retrieves copyright notice of this library.

C#

```
public string Copyright;
```

8.2.1.1.5.2.4 NLibraryInfo.Product Property

Retrieves product name.

C#

```
public string Product;
```

8.2.1.1.5.2.5 NLibraryInfo.Title Property

Retrieves library's title.

C#

```
public string Title;
```

8.2.1.1.5.2.6 NLibraryInfo.Version Property

Retrieves library version number.

C#

```
public Version Version;
```

8.2.1.1.6 NObject Class

Provides functionality for retrieving information about specified object.

C#

```
public abstract class NObject : NDisposable;
```

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.
	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject.
	Owner (see page 233)	Gets owner of the object.

8.2.1.1.6.1 NObject Methods

8.2.1.1.6.1.1 NObject.CopyParameters Method

Copies parameters values from one NObject (see page 231) object to another.

C#

```
public static void CopyParameters(NObject srcObj, NObject dstObj);
```

Parameters

Parameters	Description
NObject srcObj	Source NObject (see page 231) object which should be copied.
NObject dstObj	Destination NObject (see page 231) object where source object will be copied.

8.2.1.1.6.1.2 NObject.Free Method

Releases memory resources used by specified object.

C#

```
public static void Free(IntPtr hObject);
```

Parameters

Parameters	Description
IntPtr hObject	Pointer to previously allocated memory block to free.

8.2.1.1.6.1.3 NObject.GetNativeType Method

Retrieves native type of object.

C#

```
public NNativeType GetNativeType();
```

Returns

NativeType value.

8.2.1.1.6.1.4 GetParameter Method

8.2.1.1.6.1.4.1 NObject.GetParameter Method (ushort)

Gets value of parameter by parameter id.

C#

```
public object GetParameter(ushort parameterId);
```

Parameters

Parameters	Description
ushort parameterId	The parameter identifier.

Returns

The parameter value.

8.2.1.1.6.1.4.2 NObject.GetParameter Method (ushort, ushort)

Gets value of parameter by parameter id.

C#

```
public object GetParameter(ushort partId, ushort parameterId);
```

Parameters

Parameters	Description
ushort partId	Specifies the part of the specified NObject (see page 231).
ushort parameterId	The parameter identifier.

Returns

The parameter value.

8.2.1.1.6.1.5 NObject.Reset Method

Resets all NObject (see page 231) parameters to default values.

C#

```
public void Reset();
```

8.2.1.1.6.1.6 SetParameter Method

8.2.1.1.6.1.6.1 NObject.SetParameter Method (ushort, object)

Sets value of parameter by parameter id.

C#

```
public void SetParameter(ushort parameterId, object value);
```

Parameters

Parameters	Description
ushort parameterId	The parameter identifier.
object value	The parameter value.

8.2.1.1.6.1.6.2 NObject.SetParameter Method (ushort, ushort, object)

Sets value of parameter by parameter id.

C#

```
public void SetParameter(ushort partId, ushort parameterId, object value);
```

Parameters

Parameters	Description
ushort partId	Specifies the part of the specified NObject (see page 231).
ushort parameterId	The parameter identifier.
object value	The parameter value.

8.2.1.1.6.2 NObject Properties

8.2.1.1.6.2.1 NObject.Handle Property

Gets handle to unmanaged NObject (see page 231).

C#

```
public IntPtr Handle;
```

8.2.1.1.6.2.2 NObject.Owner Property

Gets owner of the object.

C#

```
public NObject Owner;
```

Property value

Object (owner) of NObject (see page 231) type.

8.2.1.1.7 NotActivatedException Class

The exception that is thrown when one of required Neurotechnology libraries is not activated.

C#

```
public class NotActivatedException : NeurotecExceptionBase;
```

INeurotecException Properties

	Name	Description
	Code (see page 240)	Gets an error code of the current exception.
	ManagedStackTrace (see page 240)	Gets a string representation of the frames on the managed stack at the time the current exception was thrown.
	UnmanagedStackTrace (see page 240)	Gets a string representation of the frames on the unmanaged stack at the time the current exception was thrown.

NeurotecExceptionBase Class

	Name	Description
	Code (see page 228)	Gets an error code of the current exception.

	ManagedStackTrace (see page 229)	Gets a string representation of the frames on the managed stack at the time the current exception was thrown.
	StackTrace (see page 229)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
	UnmanagedStackTrace (see page 229)	Gets a string representation of the frames on the unmanaged stack at the time the current exception was thrown.

8.2.1.1.8 NResult Class

Provides functionality for unmanaged functions error handling.

C#

```
public static class NResult;
```

NResult Fields

	Name	Description
	EArgument (see page 235)	Invalid argument.
	EArgumentNull (see page 235)	Argument is NULL (see page 82).
	EArgumentOutOfRange (see page 235)	Argument is out of range.
	EArithmetic (see page 235)	Arithmetic error occured.
	EClr (see page 235)	CLR error occured.
	ECom (see page 235)	COM error occured.
	ECore (see page 235)	Core error occured.
	EDirectoryNotFound (see page 236)	Directory not found.
	EDriveNotFound (see page 236)	Drive not found.
	EEndOfStream (see page 236)	Unexpected end of stream.
	EExternal (see page 236)	External error occured.
	EFailed (see page 236)	Operation failed.
	EFileLoad (see page 236)	Error loading module from file.
	EFileNotFound (see page 236)	File not found.
	EFormat (see page 236)	Argument is of invalid format.
	EIndexOutOfRange (see page 236)	Access with index was out of range.
	EInvalidCast (see page 237)	The cast is invalid.
	EInvalidEnumArgument (see page 237)	Argument is an invalid enum value.
	EInvalidOperation (see page 237)	Attempted to execute operation that is not valid for the object.
	EIO (see page 237)	IO (see page 330) error occured.
	ENotActivated (see page 237)	Operation is not activated.
	ENotImplemented (see page 237)	Operation is not implemented.
	ENotSupported (see page 237)	Operation is not supported.
	ENullReference (see page 237)	Attemted to reference a NULL (see page 82) object.
	EOutOfMemory (see page 237)	Out of memory.
	EOverflow (see page 238)	Overflow occured.
	EParameter (see page 238)	Invalid parameter Id.
	EParameterReadOnly (see page 238)	Attempted to set read-only parameter.
	EPathTooLong (see page 238)	Path is too long.
	ESecurity (see page 238)	Security error occured.

	ESys (see page 238)	Sys error occured.
	EWin32 (see page 238)	Win32 error occured.
	Ok (see page 238)	No error.

NResult Methods

	Name	Description
S	Check (see page 239)	Checks if the result of a function indicates that an error has occured and if so throws an exception.
S	IsFailed (see page 239)	Checks if the result of a function indicates that an error has occured.
S	IsSucceeded (see page 239)	Checks if the result of a function indicates that no error has occured.
S	RaiseError (see page 239)	Creates and throws exception object which represents the error.
S	SetError (see page 239)	For internal use only.

8.2.1.1.8.1 NResult Fields

8.2.1.1.8.1.1 NResult.EArgument Field

Invalid argument.

C#

```
public const int EArgument = -10;
```

8.2.1.1.8.1.2 NResult.EArgumentNull Field

Argument is NULL (see page 82).

C#

```
public const int EArgumentNull = -11;
```

8.2.1.1.8.1.3 NResult.EArgumentOutOfRange Field

Argument is out of range.

C#

```
public const int EArgumentOutOfRange = -12;
```

8.2.1.1.8.1.4 NResult.EArithmetic Field

Arithmetic error occured.

C#

```
public const int EArithmetic = -17;
```

8.2.1.1.8.1.5 NResult.EClr Field

CLR error occured.

C#

```
public const int EClr = -93;
```

8.2.1.1.8.1.6 NResult.ECom Field

COM error occured.

C#

```
public const int ECom = -92;
```

8.2.1.1.8.1.7 NResult.ECore Field

Core error occured.

C#

```
public const int ECore = -2;
```

8.2.1.1.8.1.8 NResult.EDirectoryNotFound Field

Directory not found.

C#

```
public const int EDirectoryNotFound = -19;
```

8.2.1.1.8.1.9 NResult.EDriveNotFound Field

Drive not found.

C#

```
public const int EDriveNotFound = -20;
```

8.2.1.1.8.1.10 NResult.EEndOfStream Field

Unexpected end of stream.

C#

```
public const int EEndOfStream = -15;
```

8.2.1.1.8.1.11 NResult.EExternal Field

External error occured.

C#

```
public const int EExternal = -90;
```

8.2.1.1.8.1.12 NResult.EFailed Field

Operation failed.

C#

```
public const int EFailed = -1;
```

8.2.1.1.8.1.13 NResult.EFileLoad Field

Error loading module from file.

C#

```
public const int EFileLoad = -22;
```

8.2.1.1.8.1.14 NResult.EFileNotFound Field

File not found.

C#

```
public const int EFileNotFound = -21;
```

8.2.1.1.8.1.15 NResult.EFormat Field

Argument is of invalid format.

C#

```
public const int EFormat = -13;
```

8.2.1.1.8.1.16 NResult.EIndexOutOfRange Field

Access with index was out of range.

C#

```
public const int EIndexOutOfRange = -9;
```

8.2.1.1.8.1.17 NResult.EInvalidCast Field

The cast is invalid.

C#

```
public const int EInvalidCast = -18;
```

8.2.1.1.8.1.18 NResult.EInvalidEnumArgument Field

Argument is an invalid enum value.

C#

```
public const int EInvalidEnumArgument = -16;
```

8.2.1.1.8.1.19 NResult.EInvalidOperation Field

Attempted to execute operation that is not valid for the object.

C#

```
public const int EInvalidOperation = -7;
```

8.2.1.1.8.1.20 NResult.EIO Field

IO (see page 330) error occured.

C#

```
public const int EIO = -14;
```

8.2.1.1.8.1.21 NResult.ENotActivated Field

Operation is not activated.

C#

```
public const int ENotActivated = -200;
```

8.2.1.1.8.1.22 NResult.ENotImplemented Field

Operation is not implemented.

C#

```
public const int ENotImplemented = -5;
```

8.2.1.1.8.1.23 NResult.ENotSupported Field

Operation is not supported.

C#

```
public const int ENotSupported = -6;
```

8.2.1.1.8.1.24 NResult.ENullReference Field

Attemted to reference a NULL (see page 82) object.

C#

```
public const int ENullReference = -3;
```

8.2.1.1.8.1.25 NResult.EOutOfMemory Field

Out of memory.

C#

```
public const int EOutOfMemory = -4;
```

8.2.1.1.8.1.26 NResult.EOverflow Field

Overflow occured.

C#

```
public const int EOverflow = -8;
```

8.2.1.1.8.1.27 NResult.EParameter Field

Invalid parameter Id.

C#

```
public const int EParameter = -100;
```

8.2.1.1.8.1.28 NResult.EParameterReadOnly Field

Attempted to set read-only parameter.

C#

```
public const int EParameterReadOnly = -101;
```

8.2.1.1.8.1.29 NResult.EPathTooLong Field

Path is too long.

C#

```
public const int EPathTooLong = -23;
```

8.2.1.1.8.1.30 NResult.ESecurity Field

Security error occured.

C#

```
public const int ESecurity = -24;
```

8.2.1.1.8.1.31 NResult.ESys Field

Sys error occured.

C#

```
public const int ESys = -94;
```

8.2.1.1.8.1.32 NResult.EWin32 Field

Win32 error occured.

C#

```
public const int EWin32 = -91;
```

8.2.1.1.8.1.33 NResult.Ok Field

No error.

C#

```
public const int Ok = 0;
```

8.2.1.1.8.2 NResult Methods

8.2.1.1.8.2.1 NResult.Check Method

Checks if the result of a function indicates that an error has occured and if so throws an exception.

C#

```
public static int Check(int result);
```

Parameters

Parameters	Description
int result	System.Int32 representing function's result.

8.2.1.1.8.2.2 NResult.IsFailed Method

Checks if the result of a function indicates that an error has occured.

C#

```
public static bool IsFailed(int result);
```

Parameters

Parameters	Description
int result	System.Int32 representing function's result

Returns

True if an error has occured, false otherwise.

8.2.1.1.8.2.3 NResult.IsSucceeded Method

Checks if the result of a function indicates that no error has occured.

C#

```
public static bool IsSucceeded(int result);
```

Parameters

Parameters	Description
int result	System.Int32 representing function's result

Returns

True if no error has occured, false otherwise.

8.2.1.1.8.2.4 NResult.RaiseError Method

Creates and throws exception object which represents the error.

C#

```
public static void RaiseError(int error);
```

Parameters

Parameters	Description
int error	Code of an error to be thrown.

8.2.1.1.8.2.5 NResult.SetError Method

For internal use only.

C#

```
public static int SetError(Exception error);
```

8.2.1.2 Interfaces

8.2.1.2.1 INeurotecException Interface

The interface that provides information about Neurotec (see page 219) exceptions.

C#

```
public interface INeurotecException;
```

INeurotecException Properties

	Name	Description
	Code (see page 240)	Gets an error code of the current exception.
	ManagedStackTrace (see page 240)	Gets a string representation of the frames on the managed stack at the time the current exception was thrown.
	UnmanagedStackTrace (see page 240)	Gets a string representation of the frames on the unmanaged stack at the time the current exception was thrown.

8.2.1.2.1.1 INeurotecException Properties

8.2.1.2.1.1.1 INeurotecException.Code Property

Gets an error code of the current exception.

C#

```
int Code;
```

8.2.1.2.1.1.2 INeurotecException.ManagedStackTrace Property

Gets a string representation of the frames on the managed stack at the time the current exception was thrown.

C#

```
string ManagedStackTrace;
```

8.2.1.2.1.1.3 INeurotecException.UnmanagedStackTrace Property

Gets a string representation of the frames on the unmanaged stack at the time the current exception was thrown.

C#

```
string UnmanagedStackTrace;
```

8.2.1.3 Structs, Records, Enums

8.2.1.3.1 Neurotec.NProcessorVendor Enumeration

Specifies the processor's vendor name.

C#

```
[Serializable]
public enum NProcessorVendor {
  Unknown = 0,
  Amd = 1,
  Centaur = 2,
  Cyrix = 3,
  Intel = 4,
  NationalSemiconductor = 5,
```

```
    NexGen = 6,
    RiseTechnology = 7,
    SiS = 8,
    Transmeta = 9,
    Umc = 10,
    Via = 11
}
```

Members

Members	Description
Unknown = 0	The processor's vendor is unknown.
Amd = 1	Advanced Micro Devices, Inc. (AMD).
Centaur = 2	Centaur Technology.
Cyrix = 3	Cyrix.
Intel = 4	Intel Corporation.
NationalSemiconductor = 5	National Semiconductor.
NexGen = 6	NexGen.
RiseTechnology = 7	Rise Technology.
SiS = 8	Silicon Integrated Systems (SIS) Corp.
Transmeta = 9	Transmeta Corporation.
Umc = 10	UMC.
Via = 11	VIA.

8.2.1.4 Types

8.2.1.4.1 Neurotec.NLibraryGetInfo Type

This delegate supports internal Neurotechnology libraries infrastructure and should not be used directly in your code.

C#

```
public delegate int NLibraryGetInfo(IntPtr pValue);
```

8.2.2 Neurotec.Images Namespace

Classes under this namespace provides functionality that enable loading, saving and converting images in various formats.

Module

.NET (see page 219)

Classes

	Name	Description
S	Bmp (see page 242)	Provides functionality for loading and saving images in BMP format.
S	Jpeg (see page 245)	Provides functionality for loading and saving images in JPEG format.
S	LosslessJpeg (see page 248)	Provides functionality for saving images in Lossless JPEG format.
	NGrayscaleImage (see page 249)	Provides functionality for managing 8-bit grayscale images.
	NImage (see page 251)	Provides functionality for managing images.
	NMonochromeImage (see page 263)	Provides functionality for managing 1-bit monochrome images.
	NPixelFormat (see page 264)	Provides functionality for working with pixel format.
	NRgb (see page 268)	Represents an RGB color.

	NRgbImage (see page 270)	Provides functionality for managing 24-bit RGB images.
S	Tiff (see page 271)	Provides functionality for loading and saving images in TIFF format.

8.2.2.1 Classes

8.2.2.1.1 Bmp Class

Provides functionality for loading and saving images in BMP format.

C#

```
public static class Bmp;
```

Bmp Methods

	Name	Description
S	LoadImage (see page 242)	Creates a new instance of the NImage (see page 251) class from memory buffer.
S	LoadImageFromBitmap (see page 243)	Creates a new instance of the NImage (see page 251) class from Bitmap.
S	LoadImageFromHBitmap (see page 244)	Creates a new instance of the NImage (see page 251) class from Windows HBITMAP.
S	SaveImage (see page 244)	Saves NImage (see page 251) to NBuffer (see page 330).
S	SaveImageToBitmap (see page 245)	Saves NImage (see page 251) to Bitmap.
S	SaveImageToHBitmap (see page 245)	Saves NImage (see page 251) to Windows HBITMAP.

8.2.2.1.1.1 Bmp Methods

8.2.2.1.1.1.1 LoadImage Method

8.2.2.1.1.1.1.1 Bmp.LoadImage Method (IntPtr, int)

Creates a new instance of the NImage (see page 251) class from memory buffer.

C#

```
public static NImage LoadImage(IntPtr buffer, int bufferLength);
```

Parameters

Parameters	Description
IntPtr buffer	Pointer to memory buffer.
int bufferLength	Size of memory buffer.

Returns

A NImage (see page 251) object.

8.2.2.1.1.1.1.2 Bmp.LoadImage Method (Stream)

Creates a new instance of the NImage (see page 251) class from stream.

C#

```
public static NImage LoadImage(Stream stream);
```

Parameters

Parameters	Description
Stream stream	Memory stream.

Returns

A NImage (see page 251) object.

8.2.2.1.1.1.3 Bmp.LoadImage Method (byte[])

Creates a new instance of the NImage (see page 251) class from byte array.

C#

```
public static NImage LoadImage(byte[] buffer);
```

Parameters

Parameters	Description
byte[] buffer	A byte array.

Returns

A NImage (see page 251) object.

8.2.2.1.1.1.4 Bmp.LoadImage Method (string)

Creates a new instance of the NImage (see page 251) class from file.

C#

```
public static NImage LoadImage(string fileName);
```

Parameters

Parameters	Description
string fileName	A string that contains the name of the file.

Returns

A NImage (see page 251) object.

8.2.2.1.1.2 LoadImageFromBitmap Method

8.2.2.1.1.2.1 Bmp.LoadImageFromBitmap Method (Bitmap)

Creates a new instance of the NImage (see page 251) class from Bitmap.

C#

```
public static NImage LoadImageFromBitmap(Bitmap bitmap);
```

Parameters

Parameters	Description
Bitmap bitmap	A Bitmap class object.

Returns

A NImage (see page 251) object.

8.2.2.1.1.2.2 Bmp.LoadImageFromBitmap Method (Bitmap, float, float)

Creates a new instance of the NImage (see page 251) class from Bitmap with specified resolution.

C#

```
public static NImage LoadImageFromBitmap(Bitmap bitmap, float horzResolution, float
vertResolution);
```

Parameters

Parameters	Description
Bitmap bitmap	A Bitmap class object.
float horzResolution	A horizontal resolution in pixels per inch of image.
float vertResolution	A vertical resolution in pixels per inch of image.

Returns

A NImage (see page 251) object.

8.2.2.1.1.1.3 Bmp.LoadImageFromHBitmap Method

Creates a new instance of the NImage (see page 251) class from Windows HBITMAP.

C#

```
public static NImage LoadImageFromHBitmap(IntPtr hBitmap);
```

Parameters

Parameters	Description
IntPtr hBitmap	Pointer to handle that specifies Windows HBITMAP.

Returns

A NImage (see page 251) object.

8.2.2.1.1.1.4 SaveImage Method

8.2.2.1.1.1.4.1 Bmp.SaveImage Method (NImage)

Saves NImage (see page 251) to NBuffer (see page 330).

C#

```
public static NBuffer SaveImage(NImage image);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.

Returns

NBuffer (see page 330) object.

8.2.2.1.1.1.4.2 Bmp.SaveImage Method (NImage, Stream)

Saves NImage (see page 251) to stream.

C#

```
public static void SaveImage(NImage image, Stream stream);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.
Stream stream	The data stream used to save the image.

8.2.2.1.1.4.3 Bmp.SaveImage Method (NImage, string)

Saves NImage (see page 251) to file.

C#

```
public static void SaveImage(NImage image, string fileName);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.
string fileName	A string that contains the name of the file.

8.2.2.1.1.5 Bmp.SaveImageToBitmap Method

Saves NImage (see page 251) to Bitmap.

C#

```
public static Bitmap SaveImageToBitmap(NImage image);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.

Returns

A Bitmap object.

8.2.2.1.1.6 Bmp.SaveImageToHBitmap Method

Saves NImage (see page 251) to Windows HBITMAP.

C#

```
public static IntPtr SaveImageToHBitmap(NImage image);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.

Returns

A pointer to handle of Windows HBITMAP.

8.2.2.1.2 Jpeg Class

Provides functionality for loading and saving images in JPEG format.

C#

```
public static class Jpeg;
```

Jpeg Fields

	Name	Description
	DefaultQuality (see page 246)	Specifies default JPEG quality.

Jpeg Methods

	Name	Description
S	LoadImage (see page 246)	Creates NImage (see page 251) object from memory buffer.
S	SaveImage (see page 247)	Saves NImage (see page 251) to NBuffer (see page 330).

8.2.2.1.2.1 Jpeg Fields

8.2.2.1.2.1.1 Jpeg.DefaultQuality Field

Specifies default JPEG quality.

C#

```
public const int DefaultQuality = 75;
```

8.2.2.1.2.2 Jpeg Methods

8.2.2.1.2.2.1 LoadImage Method

8.2.2.1.2.2.1.1 Jpeg.LoadImage Method (IntPtr, int)

Creates NImage (see page 251) object from memory buffer.

C#

```
public static NImage LoadImage(IntPtr buffer, int bufferLength);
```

Parameters

Parameters	Description
IntPtr buffer	Pointer to memory buffer.
int bufferLength	Size of memory buffer.

Returns

A NImage (see page 251) object.

8.2.2.1.2.2.1.2 Jpeg.LoadImage Method (Stream)

Creates a new instance of the NImage (see page 251) class from stream.

C#

```
public static NImage LoadImage(Stream stream);
```

Parameters

Parameters	Description
Stream stream	Memory stream.

Returns

A NImage (see page 251) object.

8.2.2.1.2.2.1.3 Jpeg.LoadImage Method (byte[])

Creates a new instance of the NImage (see page 251) class from byte array.

C#

```
public static NImage LoadImage(byte[] buffer);
```

Parameters

Parameters	Description
byte[] buffer	A byte array.

Returns

A NImage (see page 251) object.

8.2.2.1.2.2.1.4 Jpeg.LoadImage Method (string)

Creates NImage (see page 251) object from JPEG file.

C#

```
public static NImage LoadImage(string fileName);
```

Parameters

Parameters	Description
string fileName	A string that contains the name of the file.

Returns

A NImage (see page 251) object.

8.2.2.1.2.2.2 SaveImage Method

8.2.2.1.2.2.2.1 Jpeg.SaveImage Method (NImage)

Saves NImage (see page 251) to NBuffer (see page 330).

C#

```
public static NBuffer SaveImage(NImage image);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.

Returns

NBuffer (see page 330) object.

8.2.2.1.2.2.2.2 Jpeg.SaveImage Method (NImage, Stream)

Saves NImage (see page 251) object to stream in JPEG format.

C#

```
public static void SaveImage(NImage image, Stream stream);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.
Stream stream	The data stream used to save the image.

8.2.2.1.2.2.2.3 Jpeg.SaveImage Method (NImage, int)

Saves NImage (see page 251) object to byte array in JPEG format with specified quality.

C#

```
public static NBuffer SaveImage(NImage image, int quality);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.
int quality	Specifies quality of JPEG image.

Returns

NBuffer (see page 330) object.

8.2.2.1.2.2.4 Jpeg.SaveImage Method (NImage, int, Stream)

Saves NImage (see page 251) object to stream in JPEG format with specified image quality.

C#

```
public static void SaveImage(NImage image, int quality, Stream stream);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.
int quality	Specifies quality of JPEG image.
Stream stream	The data stream used to save the image.

8.2.2.1.2.2.5 Jpeg.SaveImage Method (NImage, int, string)

Saves NImage (see page 251) object to file in JPEG format with specified quality.

C#

```
public static void SaveImage(NImage image, int quality, string fileName);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.
int quality	Specifies quality of JPEG image.
string fileName	A string that contains the name of the file.

8.2.2.1.2.2.6 Jpeg.SaveImage Method (NImage, string)

Saves NImage (see page 251) object to file in JPEG format.

C#

```
public static void SaveImage(NImage image, string fileName);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.
string fileName	A string that contains the name of the file.

8.2.2.1.3 LosslessJpeg Class

Provides functionality for saving images in Lossless JPEG format.

C#

```
public static class LosslessJpeg;
```

LosslessJpeg Methods

	Name	Description
S	SaveImage (see page 249)	Saves NImage (see page 251) to NBuffer (see page 330) in Lossless JPEG format.

8.2.2.1.3.1 LosslessJpeg Methods

8.2.2.1.3.1.1 SaveImage Method

8.2.2.1.3.1.1.1 LosslessJpeg.SaveImage Method (NImage)

Saves NImage (see page 251) to NBuffer (see page 330) in Lossless JPEG format.

C#

```
public static NBuffer SaveImage(NImage image);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.

Returns

NBuffer (see page 330) object.

8.2.2.1.3.1.1.2 LosslessJpeg.SaveImage Method (NImage, Stream)

Saves NImage (see page 251) object to stream in Lossless JPEG format.

C#

```
public static void SaveImage(NImage image, Stream stream);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.
Stream stream	The data stream used to save the image.

8.2.2.1.3.1.1.3 LosslessJpeg.SaveImage Method (NImage, string)

Saves NImage (see page 251) object to file in Lossless JPEG format.

C#

```
public static void SaveImage(NImage image, string fileName);
```

Parameters

Parameters	Description
NImage image	A NImage (see page 251) object.
string fileName	A string that contains the name of the file.

8.2.2.1.4 NGrayscaleImage Class

Provides functionality for managing 8-bit grayscale images.

C#

```
public sealed class NGrayscaleImage : NImage;
```

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.

	GetNativeType (see page 232)	Retrieves native type of object.
	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

NImage Class

	Name	Description
	Clone (see page 252)	Creates NImage (see page 251) object from another NImage (see page 251) object.
S	Create (see page 252)	Creates an image with specified pixel format, size, stride and resolution.
S	FromBitmap (see page 253)	Creates NImage (see page 251) from Bitmap.
S	FromData (see page 254)	Creates NImage (see page 251) object from data with specified resolution.
S	FromFile (see page 255)	Creates NImage (see page 251) object from file.
S	FromHBitmap (see page 256)	Creates NImage (see page 251) object from handle.
S	FromImage (see page 256)	Creates NImage (see page 251) object from another NImage (see page 251) object.
S	GetWrapper (see page 258)	Creates NImage (see page 251) object wrapper.
	Save (see page 260)	Saves NImage (see page 251) object to file.
	ToBitmap (see page 260)	Creates a Bitmap.
	ToHBitmap (see page 260)	Creates Windows HBITMAP.

NGrayscaleImage Class

	Name	Description
	this (see page 251)	Gets or sets the color of the specified pixel in NImage (see page 251) object.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

NImage Class

	Name	Description
	Height (see page 260)	Gets height of image from NImage (see page 251) object.
	HorzResolution (see page 261)	Gets horizontal resolution in pixels per inch of image.
	LongSize (see page 261)	Gets size of NImage (see page 251) object.
	LongStride (see page 261)	Gets stride of image from NImage (see page 251) object.
	PixelFormat (see page 261)	Gets NPixelFormat (see page 264) of NImage (see page 251) object.
	Pixels (see page 261)	Gets pointer to array of pixels from NImage (see page 251) object.
	Size (see page 262)	Gets size of NImage (see page 251) object.
	Stride (see page 262)	Gets stride of image from NImage (see page 251) object.
	VertResolution (see page 262)	Gets vertical resolution in pixels per inch of image.
	Width (see page 262)	Gets width of image from NImage (see page 251) object.

8.2.2.1.4.1 NGrayscaleImage Methods

8.2.2.1.4.1.1 NGrayscaleImage.this Indexer

Gets or sets the color of the specified pixel in NImage (see page 251) object.

C#

```
public byte this[uint x, uint y];
```

Parameters

Parameters	Description
uint x	The x coordinate of the pixel.
uint y	The y coordinate of the pixel.

Returns

A color of specified pixel.

8.2.2.1.5 NImage Class

Provides functionality for managing images.

C#

```
public class NImage : NObject, ICloneable;
```

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.
	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

NImage Class

	Name	Description
	Clone (see page 252)	Creates NImage object from another NImage object.
S	Create (see page 252)	Creates an image with specified pixel format, size, stride and resolution.
S	FromBitmap (see page 253)	Creates NImage from Bitmap.
S	FromData (see page 254)	Creates NImage object from data with specified resolution.
S	FromFile (see page 255)	Creates NImage object from file.
S	FromHBitmap (see page 256)	Creates NImage object from handle.
S	FromImage (see page 256)	Creates NImage object from another NImage object.
S	GetWrapper (see page 258)	Creates NImage object wrapper.
	Save (see page 260)	Saves NImage object to file.
	ToBitmap (see page 260)	Creates a Bitmap.
	ToHBitmap (see page 260)	Creates Windows HBITMAP.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

NImage Class

	Name	Description
	Height (see page 260)	Gets height of image from NImage object.
	HorzResolution (see page 261)	Gets horizontal resolution in pixels per inch of image.
	LongSize (see page 261)	Gets size of NImage object.
	LongStride (see page 261)	Gets stride of image from NImage object.
	PixelFormat (see page 261)	Gets NPixelFormat (see page 264) of NImage object.
	Pixels (see page 261)	Gets pointer to array of pixels from NImage object.
	Size (see page 262)	Gets size of NImage object.
	Stride (see page 262)	Gets stride of image from NImage object.
	VertResolution (see page 262)	Gets vertical resolution in pixels per inch of image.
	Width (see page 262)	Gets width of image from NImage object.

8.2.2.1.5.1 NImage Methods

8.2.2.1.5.1.1 NImage.Clone Method

Creates NImage (see page 251) object from another NImage (see page 251) object.

C#

```
public object Clone();
```

Returns

A NImage (see page 251) object.

8.2.2.1.5.1.2 Create Method

8.2.2.1.5.1.2.1 NImage.Create Method (NPixelFormat, uint, uint, uint, float, float)

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

C#

```
public static NImage Create(NPixelFormat pixelFormat, uint width, uint height, uint stride,
float horzResolution, float vertResolution);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	Specifies pixel format of the image.
uint width	Specifies width of the image.
uint height	Specifies height of the image.
uint stride	Specifies stride of the image.
float horzResolution	Specifies horizontal resolution in pixels per inch of the image.
float vertResolution	Specifies vertical resolution in pixels per inch of the image.

Returns

If the function succeeds the return value is N_OK (see page 32).

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

- N_E_ARGUMENT (see page 27) - *pixelFormat* has invalid value.
- N_E_ARGUMENT (see page 27) - *stride* is not zero and is less than minimal
- value for specified pixel format and width.
- N_E_ARGUMENT_NULL (see page 28) - *pHImage* is NULL (see page 82).
- N_E_ARGUMENT_OUT_OF_RANGE (see page 28) - *width* or *height* is zero.
- N_E_ARGUMENT_OUT_OF_RANGE (see page 28) - *horzResolution* or *vertResolution* is
- less than zero.
- N_E_OUT_OF_MEMORY (see page 31) - there was not enough memory.

Remarks

If stride is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride (see page 123) function. Created image must be deleted using NImageFree function. *horzResolution* and *vertResolution* can be zero if resolution is not applicable for the image.

8.2.2.1.5.1.2.2 NImage.Create Method (NPixelFormat, uint, uint, ulong, float, float)

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

C#

```
public static NImage Create(NPixelFormat pixelFormat, uint width, uint height, ulong
stride, float horzResolution, float vertResolution);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	Specifies pixel format of the image.
uint width	Specifies width of the image.
uint height	Specifies height of the image.
ulong stride	Specifies stride of the image.
float horzResolution	Specifies horizontal resolution in pixels per inch of the image.
float vertResolution	Specifies vertical resolution in pixels per inch of the image.

Returns

If the function succeeds the return value is N_OK (see page 32).

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

- N_E_ARGUMENT (see page 27) - *pixelFormat* has invalid value.
- N_E_ARGUMENT (see page 27) - *stride* is not zero and is less than minimal value for specified pixel format and width.
- N_E_ARGUMENT_NULL (see page 28) - *pHImage* is NULL (see page 82).
- N_E_ARGUMENT_OUT_OF_RANGE (see page 28) - *width* or *height* is zero.
- N_E_ARGUMENT_OUT_OF_RANGE (see page 28) - *horzResolution* or *vertResolution* is less than zero.
- N_E_OUT_OF_MEMORY (see page 31) - there was not enough memory.

Remarks

If stride is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride (see page 123) function. Created image must be deleted using NImageFree function. *horzResolution* and *vertResolution* can be zero if resolution is not applicable for the image.

8.2.2.1.5.1.3 NImage.FromBitmap Method

Creates NImage (see page 251) from Bitmap.

C#

```
public static NImage FromBitmap(Bitmap bitmap);
```

Parameters

Parameters	Description
Bitmap bitmap	An object used to work with images defined by pixel data.

Returns

A NImage (see page 251) object.

8.2.2.1.5.1.4 FromData Method

8.2.2.1.5.1.4.1 NImage.FromData Method (NPixelFormat, uint, uint, uint, float, float, int, byte[])

Creates NImage (see page 251) object from data with specified resolution.

C#

```
public static NImage FromData(NPixelFormat pixelFormat, uint width, uint height, uint
stride, float horzResolution, float vertResolution, int srcStride, byte[] srcPixels);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	A NPixelFormat (see page 264) of image.
uint width	A width of image.
uint height	A height of image.
uint stride	A stride of image.
float horzResolution	A horizontal resolution in pixels per inch of image.
float vertResolution	A vertical resolution in pixels per inch of image.
int srcStride	A stride of source image.
byte[] srcPixels	A pointer to source pixel array.

Returns

A NImage (see page 251) object.

Remarks

If stride is zero then image stride is automatically calculated. For more information on image stride see Stride (see page 262) property.

Format of memory block srcPixels points to must be the same as described in Pixels (see page 261) property, only stride is equal to srcStride.

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

8.2.2.1.5.1.4.2 NImage.FromData Method (NPixelFormat, uint, uint, uint, float, float, uint, IntPtr)

Creates NImage (see page 251) object from data with specified resolution.

C#

```
public static NImage FromData(NPixelFormat pixelFormat, uint width, uint height, uint
stride, float horzResolution, float vertResolution, uint srcStride, IntPtr srcPixels);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	A NPixelFormat (see page 264) of image.
uint width	A width of image.

uint height	A height of image.
uint stride	A stride of image.
float horzResolution	A horizontal resolution in pixels per inch of image.
float vertResolution	A vertical resolution in pixels per inch of image.
uint srcStride	A stride of source image.
IntPtr srcPixels	A pointer to source pixel array.

Returns

A NImage (see page 251) object.

Remarks

If stride is zero then image stride is automatically calculated. For more information on image stride see Stride (see page 262) property.

Format of memory block srcPixels points to must be the same as described in Pixels (see page 261) property, only stride is equal to srcStride.

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

8.2.2.1.5.1.4.3 NImage.FromData Method (NPixelFormat, uint, uint, ulong, float, float, ulong, IntPtr)

Creates NImage (see page 251) object from data with specified resolution.

C#

```
public static NImage FromData(NPixelFormat pixelFormat, uint width, uint height, ulong
stride, float horzResolution, float vertResolution, ulong srcStride, IntPtr srcPixels);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	A NPixelFormat (see page 264) of image.
uint width	A width of image.
uint height	A height of image.
ulong stride	A stride of image.
float horzResolution	A horizontal resolution in pixels per inch of image.
float vertResolution	A vertical resolution in pixels per inch of image.
ulong srcStride	A stride of source image.
IntPtr srcPixels	A pointer to source pixel array.

Returns

A NImage (see page 251) object.

8.2.2.1.5.1.5 FromFile Method

8.2.2.1.5.1.5.1 NImage.FromFile Method (string)

Creates NImage (see page 251) object from file.

C#

```
public static NImage FromFile(string fileName);
```

Parameters

Parameters	Description
string fileName	A string that contains the name of the file.

Returns

A NImage (see page 251) object.

8.2.2.1.5.1.5.2 NImage.FromFile Method (string, NImageFormat)

Creates NImage (see page 251) object from file with specified NImageFormat.

C#

```
public static NImage FromFile(string fileName, NImageFormat imageFormat);
```

Parameters

Parameters	Description
string fileName	A string that contains the name of the file.
NImageFormat imageFormat	An image NImageFormat object.

Returns

A NImage (see page 251) object.

8.2.2.1.5.1.6 NImage.FromHBitmap Method

Creates NImage (see page 251) object from handle.

C#

```
public static NImage FromHBitmap(IntPtr hBitmap);
```

Parameters

Parameters	Description
IntPtr hBitmap	A pointer to handle.

Returns

A NImage (see page 251) object.

8.2.2.1.5.1.7 FromImage Method

8.2.2.1.5.1.7.1 NImage.FromImage Method (NPixelFormat, uint, NImage)

Creates NImage (see page 251) object from another NImage (see page 251) object.

C#

```
public static NImage FromImage(NPixelFormat pixelFormat, uint stride, NImage srcImage);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	A NPixelFormat (see page 264) of image.
uint stride	A stride of image.
NImage srcImage	A NImage (see page 251) source object.

Returns

A NImage (see page 251) object.

Remarks

If stride is zero then image stride is automatically calculated. For more information on image stride see Stride (see page 262) property.

8.2.2.1.5.1.7.2 NImage.FromImage Method (NPixelFormat, uint, float, float, NImage)

Creates NImage (see page 251) object from another NImage (see page 251) object.

C#

```
public static NImage FromImage(NPixelFormat pixelFormat, uint stride, float horzResolution,
float vertResolution, NImage srcImage);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	A NPixelFormat (see page 264) of image.
uint stride	A stride of image.
float horzResolution	A horizontal resolution in pixels per inch of image.
float vertResolution	A vertical resolution in pixels per inch of image.
NImage srcImage	A NImage (see page 251) source object.

Returns

A NImage (see page 251) object.

Remarks

If stride is zero then image stride is automatically calculated. For more information on image stride see Stride (see page 262) property.

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

8.2.2.1.5.1.7.3 NImage.FromImage Method (NPixelFormat, ulong, NImage)

Creates NImage (see page 251) object from another NImage (see page 251) object.

C#

```
public static NImage FromImage(NPixelFormat pixelFormat, ulong stride, NImage srcImage);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	A NPixelFormat (see page 264) of image.
ulong stride	A stride of image.
NImage srcImage	A NImage (see page 251) source object.

Returns

A NImage (see page 251) object.

Remarks

If stride is zero then image stride is automatically calculated. For more information on image stride see Stride (see page 262) property.

8.2.2.1.5.1.7.4 NImage.FromImage Method (NPixelFormat, ulong, float, float, NImage)

Creates NImage (see page 251) object from another NImage (see page 251) object.

C#

```
public static NImage FromImage(NPixelFormat pixelFormat, ulong stride, float
horzResolution, float vertResolution, NImage srcImage);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	A NPixelFormat (see page 264) of image.

ulong stride	A stride of image.
float horzResolution	A horizontal resolution in pixels per inch of image.
float vertResolution	A vertical resolution in pixels per inch of image.
NImage srcImage	A NImage (see page 251) source object.

Returns

A NImage (see page 251) object.

Remarks

If stride is zero then image stride is automatically calculated. For more information on image stride see Stride (see page 262) property.

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

8.2.2.1.5.1.8 GetWrapper Method

8.2.2.1.5.1.8.1 NImage.GetWrapper Method (NPixelFormat, uint, uint, int, float, float, byte[])

Creates NImage (see page 251) object wrapper.

C#

```
public static NImage GetWrapper(NPixelFormat pixelFormat, uint width, uint height, int
stride, float horzResolution, float vertResolution, byte[] srcPixels);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	A NPixelFormat (see page 264) of image.
uint width	A width of image.
uint height	A height of image.
int stride	A stride of image.
float horzResolution	A horizontal resolution in pixels per inch of image.
float vertResolution	A vertical resolution in pixels per inch of image.
byte[] srcPixels	A source pixel array.

Returns

A NImage (see page 251) object.

Remarks

For more information on image stride see Stride (see page 262) property.

Format of memory block pixels points to must be the same as described in Pixels (see page 261) property.

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

8.2.2.1.5.1.8.2 NImage.GetWrapper Method (NPixelFormat, uint, uint, uint, float, float, IntPtr, bool)

Creates NImage (see page 251) object wrapper.

C#

```
public static NImage GetWrapper(NPixelFormat pixelFormat, uint width, uint height, uint
stride, float horzResolution, float vertResolution, IntPtr pixels, bool ownsPixels);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	A NPixelFormat (see page 264) of image.
uint width	A width of image.

uint height	A height of image.
uint stride	A stride of image.
float horzResolution	A horizontal resolution in pixels per inch of image.
float vertResolution	A vertical resolution in pixels per inch of image.
IntPtr pixels	Pointer to memory block containing pixels for the image.
bool ownsPixels	Specifies whether pixels will be automatically deleted with the image (if set to true).

Returns

A NImage (see page 251) object.

Remarks

For more information on image stride see Stride (see page 262) property.

Format of memory block pixels points to must be the same as described in Pixels (see page 261) property.

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

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

8.2.2.1.5.1.8.3 NImage.GetWrapper Method (NPixelFormat, uint, uint, ulong, float, float, IntPtr, bool)

Creates NImage (see page 251) object wrapper.

C#

```
public static NImage GetWrapper(NPixelFormat pixelFormat, uint width, uint height, ulong
stride, float horzResolution, float vertResolution, IntPtr pixels, bool ownsPixels);
```

Parameters

Parameters	Description
NPixelFormat pixelFormat	A NPixelFormat (see page 264) of image.
uint width	A width of image.
uint height	A height of image.
ulong stride	A stride of image.
float horzResolution	A horizontal resolution in pixels per inch of image.
float vertResolution	A vertical resolution in pixels per inch of image.
IntPtr pixels	Pointer to memory block containing pixels for the image.
bool ownsPixels	Specifies whether pixels will be automatically deleted with the image (if set to true).

Returns

A NImage (see page 251) object.

Remarks

For more information on image stride see Stride (see page 262) property.

Format of memory block pixels points to must be the same as described in Pixels (see page 261) property.

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

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

8.2.2.1.5.1.9 Save Method

8.2.2.1.5.1.9.1 NImage.Save Method (string)

Saves NImage (see page 251) object to file.

C#

```
public void Save(string fileName);
```

Parameters

Parameters	Description
string fileName	A string that contains the name of the file.

8.2.2.1.5.1.9.2 NImage.Save Method (string, NImageFormat)

Saves NImage (see page 251) object to file with specified NImageFormat.

C#

```
public void Save(string fileName, NImageFormat imageFormat);
```

Parameters

Parameters	Description
string fileName	A string that contains the name of the file.
NImageFormat imageFormat	An image NImageFormat object.

8.2.2.1.5.1.10 NImage.ToBitmap Method

Creates a Bitmap.

C#

```
public Bitmap ToBitmap();
```

Returns

A Bitmap object.

8.2.2.1.5.1.11 NImage.ToHBitmap Method

Creates Windows HBITMAP.

C#

```
public IntPtr ToHBitmap();
```

Returns

A Windows HBITMAP.

8.2.2.1.5.2 NImage Properties

8.2.2.1.5.2.1 NImage.Height Property

Gets height of image from NImage (see page 251) object.

C#

```
public uint Height;
```

Property value

A height of image.

8.2.2.1.5.2.2 NImage.HorzResolution Property

Gets horizontal resolution in pixels per inch of image.

C#

```
public float HorzResolution;
```

Property value

A horizontal resolution in pixels per inch of image.

Remarks

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

8.2.2.1.5.2.3 NImage.LongSize Property

Gets size of NImage (see page 251) object.

C#

```
public ulong LongSize;
```

Property value

A size of NImage (see page 251) object.

Remarks

Size (see page 262) of memory block containing image pixels is equal to image height multiplied by image stride. For more information see Height (see page 260) and Stride (see page 262) properties.

8.2.2.1.5.2.4 NImage.LongStride Property

Gets stride of image from NImage (see page 251) object.

C#

```
public ulong LongStride;
```

Property value

A stride of image.

Remarks

Stride (see page 262) (size of one row) of the image depends on image pixel format and width. It cannot be less than value obtained with GetRowLongSize or GetRowSize methods with arguments obtained with PixelFormat (see page 261) and Width (see page 262) properties.

8.2.2.1.5.2.5 NImage.PixelFormat Property

Gets NPixelFormat (see page 264) of NImage (see page 251) object.

C#

```
public NPixelFormat PixelFormat;
```

Property value

A NPixelFormat (see page 264) structure.

8.2.2.1.5.2.6 NImage.Pixels Property

Gets pointer to array of pixels from NImage (see page 251) object.

C#

```
public IntPtr Pixels;
```

Property value

A pointer to pixel array.

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 PixelFormat (see page 261), Width (see page 262), Height (see page 260), Stride (see page 262), and Size (see page 262) properties.

8.2.2.1.5.2.7 NImage.Size Property

Gets size of NImage (see page 251) object.

C#

```
public uint Size;
```

Property value

A size of NImage (see page 251) object.

Remarks

Size of memory block containing image pixels is equal to image height multiplied by image stride. For more information see Height (see page 260) and Stride (see page 262) properties.

8.2.2.1.5.2.8 NImage.Stride Property

Gets stride of image from NImage (see page 251) object.

C#

```
public uint Stride;
```

Property value

A stride of image.

Remarks

Stride (size of one row) of the image depends on image pixel format and width. It cannot be less than value obtained with GetRowLongSize or GetRowSize methods with arguments obtained with PixelFormat (see page 261) and Width (see page 262) properties.

8.2.2.1.5.2.9 NImage.VertResolution Property

Gets vertical resolution in pixels per inch of image.

C#

```
public float VertResolution;
```

Property value

A vertical resolution in pixels per inch of image.

Remarks

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

8.2.2.1.5.2.10 NImage.Width Property

Gets width of image from NImage (see page 251) object.

C#

```
public uint Width;
```

Property value

A width of image.

8.2.2.1.6 NMonochromeImage Class

Provides functionality for managing 1-bit monochrome images.

C#

```
public sealed class NMonochromeImage : NImage;
```

Remarks

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

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.
	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

NImage Class

	Name	Description
	Clone (see page 252)	Creates NImage (see page 251) object from another NImage (see page 251) object.
S	Create (see page 252)	Creates an image with specified pixel format, size, stride and resolution.
S	FromBitmap (see page 253)	Creates NImage (see page 251) from Bitmap.
S	FromData (see page 254)	Creates NImage (see page 251) object from data with specified resolution.
S	FromFile (see page 255)	Creates NImage (see page 251) object from file.
S	FromHBitmap (see page 256)	Creates NImage (see page 251) object from handle.
S	FromImage (see page 256)	Creates NImage (see page 251) object from another NImage (see page 251) object.
S	GetWrapper (see page 258)	Creates NImage (see page 251) object wrapper.
	Save (see page 260)	Saves NImage (see page 251) object to file.
	ToBitmap (see page 260)	Creates a Bitmap.
	ToHBitmap (see page 260)	Creates Windows HBITMAP.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

NImage Class

	Name	Description
	Height (see page 260)	Gets height of image from NImage (see page 251) object.
	HorzResolution (see page 261)	Gets horizontal resolution in pixels per inch of image.
	LongSize (see page 261)	Gets size of NImage (see page 251) object.
	LongStride (see page 261)	Gets stride of image from NImage (see page 251) object.
	PixelFormat (see page 261)	Gets NPixelFormat (see page 264) of NImage (see page 251) object.
	Pixels (see page 261)	Gets pointer to array of pixels from NImage (see page 251) object.
	Size (see page 262)	Gets size of NImage (see page 251) object.
	Stride (see page 262)	Gets stride of image from NImage (see page 251) object.
	VertResolution (see page 262)	Gets vertical resolution in pixels per inch of image.
	Width (see page 262)	Gets width of image from NImage (see page 251) object.

8.2.2.1.7 NPixelFormat Structure

Provides functionality for working with pixel format.

C#

```
[StructLayout(LayoutKind.Sequential)]
public struct NPixelFormat {
  public static readonly NPixelFormat Monochrome = new NPixelFormat(0x00001001);
  public static readonly NPixelFormat Grayscale = new NPixelFormat(0x00301001);
  public static readonly NPixelFormat Rgb = new NPixelFormat(0x00303003);
}
```

Remarks

Image pixel format is not limited to these fields. However only these fields are provided for usage with this SDK.

NPixelFormat Fields

	Name	Description
S R	Grayscale (see page 265)	Each pixel value is stored in 8 bits representing 256 shades of gray.
S R	Monochrome (see page 265)	Each pixel value is stored in 1 bit representing either black (value 0) or white (value 1).
S R	Rgb (see page 265)	Each pixel value is stored in 24 bits consisting of three 8-bit values representing red, green and blue color components.

NPixelFormat Methods

	Name	Description
S	CalcRowLongSize (see page 265)	Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.
S	CalcRowSize (see page 266)	Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.
	Equals (see page 266)	Determines whether the specified Object is equal to the current Object.
	GetHashCode (see page 267)	Gets the hash code. Is intended for a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.

	GetRowLongSize (see page 267)	Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat.
	GetRowSize (see page 267)	Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat.
S	IsValid (see page 268)	Checks whether current NPixelFormat value is valid.

NPixelFormat Properties

	Name	Description
	BitsPerPixel (see page 268)	Each pixel value is stored in 24 bits consisting of three 8-bit values representing red, green and blue color components.

8.2.2.1.7.1 NPixelFormat Fields

8.2.2.1.7.1.1 NPixelFormat.Grayscale Field

Each pixel value is stored in 8 bits representing 256 shades of gray.

C#

```
public static readonly NPixelFormat Grayscale = new NPixelFormat(0x00301001);
```

8.2.2.1.7.1.2 NPixelFormat.Monochrome Field

Each pixel value is stored in 1 bit representing either black (value 0) or white (value 1).

C#

```
public static readonly NPixelFormat Monochrome = new NPixelFormat(0x00001001);
```

8.2.2.1.7.1.3 NPixelFormat.Rgb Field

Each pixel value is stored in 24 bits consisting of three 8-bit values representing red, green and blue color components.

C#

```
public static readonly NPixelFormat Rgb = new NPixelFormat(0x00303003);
```

8.2.2.1.7.2 NPixelFormat Methods

8.2.2.1.7.2.1 CalcRowLongSize Method

8.2.2.1.7.2.1.1 NPixelFormat.CalcRowLongSize Method (uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.

C#

```
public static ulong CalcRowLongSize(uint bitCount, uint length);
```

Parameters

Parameters	Description
uint bitCount	A number of bytes needed to store line of pixels.
uint length	A length of pixels.

Returns

The number of bytes needed to store line of specified length of pixels with specified bits per pixel.

8.2.2.1.7.2.1.2 NPixelFormat.CalcRowLongSize Method (uint, uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel and alignment.

C#

```
public static ulong CalcRowLongSize(uint bitCount, uint length, uint alignment);
```

Parameters

Parameters	Description
uint bitCount	A number of bytes needed to store line of pixels.
uint length	A length of lpixels.
uint alignment	Alignment.

Returns

The number of bytes needed to store line of specified length of pixels with specified bits per pixel.

8.2.2.1.7.2.2 CalcRowSize Method

8.2.2.1.7.2.2.1 NPixelFormat.CalcRowSize Method (uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.

C#

```
public static uint CalcRowSize(uint bitCount, uint length);
```

Parameters

Parameters	Description
uint bitCount	A number of bytes needed to store line of pixels.
uint length	A length of pixels.

Returns

The number of bytes needed to store line of specified length of pixels with specified bits per pixel.

8.2.2.1.7.2.2.2 NPixelFormat.CalcRowSize Method (uint, uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel and alignment.

C#

```
public static uint CalcRowSize(uint bitCount, uint length, uint alignment);
```

Parameters

Parameters	Description
uint bitCount	A number of bytes needed to store line of pixels.
uint length	A length of pixels.
uint alignment	Alignment.

Returns

The number of bytes needed to store line of specified length of pixels with specified bits per pixel.

8.2.2.1.7.2.3 NPixelFormat.Equals Method

Determines whether the specified Object is equal to the current Object.

C#

```
public override bool Equals(object obj);
```

Parameters

Parameters	Description
object obj	The Object to compare with the current Object.

Returns

true if the specified Object is equal to the current Object; otherwise, *false*.

8.2.2.1.7.2.4 NPixelFormat.GetHashCode Method

Gets the hash code. Is intended for a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.

C#

```
public override int GetHashCode();
```

Returns

A hash code for the current Object.

8.2.2.1.7.2.5 GetRowLongSize Method

8.2.2.1.7.2.5.1 NPixelFormat.GetRowLongSize Method (uint)

Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat (see page 264).

C#

```
public ulong GetRowLongSize(uint length);
```

Parameters

Parameters	Description
uint length	A length of pixels.

Returns

The number of bytes needed to store line of specified length of pixels with specified NPixel Format.

8.2.2.1.7.2.5.2 NPixelFormat.GetRowLongSize Method (uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat (see page 264) and alignment.

C#

```
public ulong GetRowLongSize(uint length, uint alignment);
```

Parameters

Parameters	Description
uint length	A length of pixels.
uint alignment	Alignment.

Returns

The number of bytes needed to store line of specified length of pixels with specified NPixel Format.

8.2.2.1.7.2.6 GetRowSize Method

8.2.2.1.7.2.6.1 NPixelFormat.GetRowSize Method (uint)

Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat (see page 264).

C#

```
public uint GetRowSize(uint length);
```

Parameters

Parameters	Description
uint length	A length of pixels.

Returns

The number of bytes needed to store line of specified length of pixels with specified NPixel Format.

8.2.2.1.7.2.6.2 NPixelFormat.GetRowSize Method (uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat (see page 264) and alignment.

C#

```
public uint GetRowSize(uint length, uint alignment);
```

Parameters

Parameters	Description
uint length	A length of pixels.
uint alignment	Alignment.

Returns

The number of bytes needed to store line of specified length of pixels with specified NPixel Format.

8.2.2.1.7.2.7 NPixelFormat.IsValid Method

Checks whether current NPixelFormat (see page 264) value is valid.

C#

```
public static bool IsValid(NPixelFormat value);
```

Parameters

Parameters	Description
NPixelFormat value	The NPixelFormat (see page 264) object.

Returns

true if the object is valid NPixelFormat (see page 264), *false* if not.

8.2.2.1.7.3 NPixelFormat Properties

8.2.2.1.7.3.1 NPixelFormat.BitsPerPixel Property

Each pixel value is stored in 24 bits consisting of three 8-bit values representing red, green and blue color components.

C#

```
public uint BitsPerPixel;
```

Property value

A number of bits. sa NPixelFormat (see page 264).

8.2.2.1.8 NRgb Structure

Represents an RGB color.

C#

```
[StructLayout(LayoutKind.Sequential)]
```

```
public struct NRgb {
}
```

Methods

	Name	Description
	NRgb (see page 269)	Initializes a new instance of the NRgb structure.

NRgb Properties

	Name	Description
	Blue (see page 269)	Gets the blue component value of this NRGB structure.
	Green (see page 269)	Gets the green component value of this NRGB structure.
	Red (see page 269)	Gets the red component value of this NRGB structure.

8.2.2.1.8.1 NRgb.NRgb Constructor

Initializes a new instance of the NRgb structure.

C#

```
public NRgb(byte red, byte green, byte blue);
```

Parameters

Parameters	Description
byte red	The blue component value of this NRGB.
byte green	The green component value of this NRGB.
byte blue	The red component value of this NRGB.

8.2.2.1.8.2 NRgb Properties

8.2.2.1.8.2.1 NRgb.Blue Property

Gets the blue component value of this NRGB structure.

C#

```
public byte Blue;
```

Property value

The blue component value of this NRGB.

8.2.2.1.8.2.2 NRgb.Green Property

Gets the green component value of this NRGB structure.

C#

```
public byte Green;
```

Property value

The green component value of this NRGB.

8.2.2.1.8.2.3 NRgb.Red Property

Gets the red component value of this NRGB structure.

C#

```
public byte Red;
```

Property value

The red component value of this NRGB.

8.2.2.1.9 NRgbImage Class

Provides functionality for managing 24-bit RGB images.

C#

```
public sealed class NRgbImage : NImage;
```

Remarks

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

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.
	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

NImage Class

	Name	Description
	Clone (see page 252)	Creates NImage (see page 251) object from another NImage (see page 251) object.
S	Create (see page 252)	Creates an image with specified pixel format, size, stride and resolution.
S	FromBitmap (see page 253)	Creates NImage (see page 251) from Bitmap.
S	FromData (see page 254)	Creates NImage (see page 251) object from data with specified resolution.
S	FromFile (see page 255)	Creates NImage (see page 251) object from file.
S	FromHBitmap (see page 256)	Creates NImage (see page 251) object from handle.
S	FromImage (see page 256)	Creates NImage (see page 251) object from another NImage (see page 251) object.
S	GetWrapper (see page 258)	Creates NImage (see page 251) object wrapper.
	Save (see page 260)	Saves NImage (see page 251) object to file.
	ToBitmap (see page 260)	Creates a Bitmap.
	ToHBitmap (see page 260)	Creates Windows HBITMAP.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

NImage Class

	Name	Description
	Height (see page 260)	Gets height of image from NImage (see page 251) object.

	HorzResolution (see page 261)	Gets horizontal resolution in pixels per inch of image.
	LongSize (see page 261)	Gets size of NImage (see page 251) object.
	LongStride (see page 261)	Gets stride of image from NImage (see page 251) object.
	PixelFormat (see page 261)	Gets NPixelFormat (see page 264) of NImage (see page 251) object.
	Pixels (see page 261)	Gets pointer to array of pixels from NImage (see page 251) object.
	Size (see page 262)	Gets size of NImage (see page 251) object.
	Stride (see page 262)	Gets stride of image from NImage (see page 251) object.
	VertResolution (see page 262)	Gets vertical resolution in pixels per inch of image.
	Width (see page 262)	Gets width of image from NImage (see page 251) object.

8.2.2.1.10 Tiff Class

Provides functionality for loading and saving images in TIFF format.

C#

```
public static class Tiff;
```

Tiff Methods

	Name	Description
S	LoadImage (see page 271)	Creates NImage (see page 251) object from memory buffer.

8.2.2.1.10.1 Tiff Methods

8.2.2.1.10.1.1 LoadImage Method

8.2.2.1.10.1.1.1 Tiff.LoadImage Method (IntPtr, int)

Creates NImage (see page 251) object from memory buffer.

C#

```
public static NImage LoadImage(IntPtr buffer, int bufferLength);
```

Parameters

Parameters	Description
IntPtr buffer	Pointer to memory buffer.
int bufferLength	Size of memory buffer.

Returns

A NImage (see page 251) object.

8.2.2.1.10.1.1.2 Tiff.LoadImage Method (byte[])

Creates NImage (see page 251) object from byte array.

C#

```
public static NImage LoadImage(byte[] buffer);
```

Parameters

Parameters	Description
byte[] buffer	A byte array. return A NImage (see page 251) object.

Returns

A NImage (see page 251) object.

8.2.2.1.10.1.1.3 Tiff.LoadImage Method (string)

Creates NImage (see page 251) object from TIFF file.

C#

```
public static NImage LoadImage(string fileName);
```

Parameters

Parameters	Description
string fileName	A string that contains the name of the file.

Returns

A NImage (see page 251) object.

8.2.3 Neurotec.SentiSight Namespace

General namespace of SentiSight (see page 296) library classes.

Remarks

DLL: Neurotec.SentiSight.dll. which is a wrapper of SentiSight.dll. Neurotec.SentiSight.dll. and SentiSight.dll should be used together.

Module

.NET (see page 219)

Classes

	Name	Description
	SEEngine (see page 272)	Main class of SentiSight (see page 296) library. It uses helper classes to provide functionality of the library.
	SEModel (see page 292)	Keeps learnt object models.
S	SentiSight (see page 296)	General class for SentiSight library.
	SERecognitionDetails (see page 297)	Provides detailed information about recognition (Recognize method) results.
	SEShape (see page 300)	Provides functions for working with SentiSight (see page 296) shape objects.

Structs, Records, Enums

	Name	Description
	SELrnMode (see page 304)	Enumerates different type of learning parameters.
	SERecSpeed (see page 304)	Enumerates SentiSight (see page 296) recognition speed parameters.
	SERecTransformType (see page 304)	Transform types used in recognition process to align shape or transform images.
	SEStatus (see page 305)	Enumerates different types of SentiSight (see page 296) update status parameters.

8.2.3.1 Classes

8.2.3.1.1 SEEngine Class

Main class of SentiSight (see page 296) library. It uses helper classes to provide functionality of the library.

C#

```
public class SEEngine : NObject;
```

Methods

SEEngine Class

	Name	Description
	SEEngine (see page 274)	Initializes a new instance of SEEngine.

SEEngine Classes

SEEngine Class

	Name	Description
	SELearning (see page 274)	Provides object learning functionality.
	SERecognition (see page 278)	Provides object recognition functionality.
	SESeparation (see page 285)	Provides foreground/background separation functionality.

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.
	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

SEEngine Class

	Name	Description
	CreateModel (see page 291)	Creates SEModel (see page 292) class instance - the model where the learnt object will be kept.
S	FromHandle (see page 291)	Creates new SEEngine object from specified handle.
S	GetInfo (see page 292)	Gets NLibraryInfo (see page 229) object which contains information about SentiSight (see page 296) library.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

SEEngine Class

	Name	Description
	Learning (see page 292)	Gets SentiSightEngine learning module.
	Recognition (see page 292)	Gets SentiSightEngine recognition module.
	Separation (see page 292)	Gets SentiSightEngine separation module.

8.2.3.1.1.1 SEEngine.SEEngine Constructor

Initializes a new instance of SEEngine.

C#

```
public SEEngine();
```

8.2.3.1.1.2 SEEngine Classes

8.2.3.1.1.2.1 SEEngine.SELearning Class

Provides object learning functionality.

C#

```
public sealed class SELearning;
```

SELearning Fields

	Name	Description
	ParameterEnhanceMask (see page 274)	Specifies whether to extend image mask during object learning or not.
	ParameterGeneralizationThreshold (see page 274)	Specifies generalization threshold.
	ParameterMode (see page 275)	Specifies learning mode.

SELearning Methods

	Name	Description
	AddToModel (see page 275)	Adds specified images to the SEModel (see page 292) object.
	GeneralizeModel (see page 276)	Compresses information kept in SEModel (see page 292) object.
	RemoveFromModel (see page 277)	Removes an image from model.

SELearning Properties

	Name	Description
	EnhanceMask (see page 277)	Enables and disables usage of extend image mask during object learning.
	GeneralizationThreshold (see page 277)	Gets or sets generalization threshold value.
	Mode (see page 278)	Gets or sets the learning mode of SentiSight (see page 296) model.

8.2.3.1.1.2.1.1 SELearning Fields

8.2.3.1.1.2.1.1.1 SEEngine.SELearning.ParameterEnhanceMask Field

Specifies whether to extend image mask during object learning or not.

C#

```
public const ushort ParameterEnhanceMask = 50100;
```

8.2.3.1.1.2.1.1.2 SEEngine.SELearning.ParameterGeneralizationThreshold Field

Specifies generalization threshold.

C#

```
public const ushort ParameterGeneralizationThreshold = 50102;
```

8.2.3.1.1.2.1.1.3 SEEngine.SELearning.ParameterMode Field

Specifies learning mode.

C#

```
public const ushort ParameterMode = 50101;
```

8.2.3.1.1.2.1.2 SELearning Methods

8.2.3.1.1.2.1.2.1 AddToModel Method

8.2.3.1.1.2.1.2.1.1 SEEngine.SELearning.AddToModel Method (SEModel, NImage, NImage[], SEShape[], out uint[])

Adds specified images to the SEModel (see page 292) object.

C#

```
public SEStatus[] AddToModel(SEModel model, NImage image, NImage[] masks, SEShape[] shapes,
out uint[] refIds);
```

Parameters

Parameters	Description
SEModel model	A reference to SEModel (see page 292) where image will be added.
NImage image	An image of the object. It must be 3 channel RGB color image.
NImage[] masks	Array of masks to be added to specified model.
SEShape[] shapes	Array of shapes to be added to specified model.
out uint[] refIds	[out] Reference Id of added model.

Returns

Array of SEStatus (see page 305) enumeration.

8.2.3.1.1.2.1.2.1.2 SEEngine.SELearning.AddToModel Method (SEModel, NImage, out uint)

Adds a single image to the SEModel (see page 292) object.

C#

```
public SEStatus AddToModel(SEModel model, NImage image, out uint refId);
```

Parameters

Parameters	Description
SEModel model	A reference to SEModel (see page 292) where image will be added.
NImage image	An image of the object. It must be 3 channel RGB colour image.
out uint refId	[out] Reference Id of added model.

Remarks

This is the main object learning function. It can be called only after SEModel (see page 292) instance was created (using SentiSightEngine.CreateModel (see page 291)). The method AddToModel should be repeatedly called with images of the same object until a complete object model is learnt. After learning is completed the model may be compressed by calling GeneralizeModel (see page 276) method. Also, this model may be saved to external buffer (SEModel.Save method) and loaded (SEModel.Load method) from it later.

Exceptions

Exceptions	Description
NArgumentNullException	The model or the image of the object are NULL (see page 82).

See Also

SentiSightEngine.CreateModel GeneralizeModel SEModel.Load SEModel.Save

8.2.3.1.1.2.1.2.2 GeneralizeModel Method

8.2.3.1.1.2.1.2.2.1 SEEngine.SELearning.GeneralizeModel Method (SEModel, double)

Compresses information kept in SEModel (see page 292) object.

C#

```
[Obsolete("Use GeneralizeModel(SEModel, SEStatus) instead")]
public void GeneralizeModel(SEModel model, double threshold);
```

Parameters

Parameters	Description
SEModel model	Model to be generalized.
Threshold	A generalization threshold. Its values are discussed below. If a recommended value "0" was used, the most suitable threshold will be calculated and used in the method.

Remarks

Object learning method AddToModel (see page 275) acquires information about the object. After object learning is complete, the model may be compressed by calling GeneralizeModel. Generalization function alters contents of SEModel (see page 292) object; and as a consequence it contains less redundant information.

The amount of redundant information and the size of the model after generalization may be controlled through a threshold provided to the method. If this value is "0", the most suitable threshold for this model will be calculated and used in generalization process. If value is not zero, this threshold means maximum similarity score of two images that they both still would be kept in the model. Generalization process compares all the images in learnt model with each other and removes those images which are "too similar" to some others. This "too similar" concept means that their similarity score is above a certain threshold.

Notes

This method is obsolete.

See Also

AddToModel (see page 275)

8.2.3.1.1.2.1.2.2.2 SEEngine.SELearning.GeneralizeModel Method (SEModel, out SEStatus)

Generalizes information kept in SEModel (see page 292) object.

C#

```
public SEModel GeneralizeModel(SEModel model, out SEStatus status);
```

Parameters

Parameters	Description
SEModel model	Model to be generalized.
out SEStatus status	Generalization status.

Remarks

Object learning method AddToModel (see page 275) acquires information about the object. After object learning is complete, the model may be compressed by calling GeneralizeModel. Generalization function alters contents of SEModel (see page 292) object; and as a consequence it contains less redundant information.

The amount of redundant information and the size of the model after generalization may be controlled through a threshold provided to the method. If this value is "0", the most suitable threshold for this model will be calculated and used in generalization process. If value is not zero, this threshold means maximum similarity score of two images that they both still would be kept in the model. Generalization process compares all the images in learnt model with each other and removes those images which are "too similar" to some others. This "too similar" concept means that their similarity score is above a certain threshold.

See Also

AddToModel (see page 275)

8.2.3.1.1.2.1.2.3 SEEngine.SELearning.RemoveFromModel Method

Removes an image from model.

C#

```
public void RemoveFromModel(SEModel model, uint refId);
```

Parameters

Parameters	Description
SEModel model	A reference to model from which an image should be removed.
uint refId	An image that should be removed Id.

8.2.3.1.1.2.1.3 SELearning Properties

8.2.3.1.1.2.1.3.1 SEEngine.SELearning.EnhanceMask Property

Enables and disables usage of extend image mask during object learning.

C#

```
public bool EnhanceMask;
```

Remarks

This parameter is used during object learning part by calling AddToModel (see page 275) method. It is valid only if an image mask is provided to the method. If the parameter is set to true, the image mask is extended to eliminate noise, sharp corners and accidental halls in it. If parameter is set to false, the image mask is used as it is provided to the AddToModel (see page 275) method.

The default value for this parameter is true, thus the image mask is extended before using it in the object learning function.

See Also

AddToModel (see page 275)

8.2.3.1.1.2.1.3.2 SEEngine.SELearning.GeneralizationThreshold Property

Gets or sets generalization threshold value.

C#

```
public double GeneralizationThreshold;
```

Property value

Generalization threshold.

See Also

Neurotec.SentiSight.SEEngine.SELearning.GeneralizeModel (see page 276)

8.2.3.1.1.2.1.3.3 SEEngine.SELearning.Mode Property

Gets or sets the learning mode of SentiSight (see page 296) model.

C#

```
public SELrnMode Mode;
```

Property value

The learning mode of SentiSight (see page 296) model.

8.2.3.1.1.2.2 SEEngine.SERecognition Class

Provides object recognition functionality.

C#

```
public sealed class SERecognition;
```

SERecognition Classes

	Name	Description
	RecognitionDetailsCollection (see page 279)	Represents a collection of the recognition details values that can be individually accessed by index.

SERecognition Fields

	Name	Description
	ParameterSpeed (see page 279)	Sets SentiSight (see page 296) recognition speed.
	ParameterThreshold (see page 279)	Specifies recognition threshold.
	ParameterTransformType (see page 279)	Sets transform type to be used in recognition to align shape.
	ParameterUseTracking (see page 279)	Specifies whether to track the object during object recognition or not.
	ParamSpeed (see page 279)	Defines SentiSight (see page 296) recognition speed.
	ParamTransformType (see page 280)	Defines type to be used in recognition to align shape.
	ParamUseTracking (see page 280)	Parameter used to specify whether to use tracking for recognition.

SERecognition Methods

	Name	Description
	AddModel (see page 280)	Adds a learnt model of the object. These models will be used in Recognize (see page 281) method.
	GetModelIds (see page 281)	Returns user all Ids for the model being recognized.
	Recognize (see page 281)	Compares a test image with models which were set to the recognition module.
	RemoveAllModels (see page 283)	Removes all models added to the recognition engine.
	RemoveModel (see page 283)	Removes the model and it will not be used in #Recognize (see page 281)() method.

SERecognition Properties

	Name	Description
	ModelCount (see page 284)	Gets a number of models used in recognition module.

	RecognitionDetails (see page 284)	Gets the RecognitionDetailsCollection (see page 279).
	Speed (see page 284)	Sets or gets a recognition speed of SentiSight (see page 296) engine.
	Threshold (see page 284)	Gets or sets recognition threshold.
	TrackingImageSize (see page 284)	Gets an image size used by tracking.
	TransformType (see page 285)	Sets or gets the transform type for SentiSight (see page 296) recognition.
	UseTracking (see page 285)	Enables and disables tracking the object during object recognition.

8.2.3.1.1.2.2.1 SERecognition Classes

8.2.3.1.1.2.2.1.1 SEEngine.SERecognition.RecognitionDetailsCollection Class

Represents a collection of the recognition details values that can be individually accessed by index.

C#

```
public sealed class RecognitionDetailsCollection :
NObjectReadOnlyCollection<SERecognitionDetails>;
```

8.2.3.1.1.2.2.2 SERecognition Fields

8.2.3.1.1.2.2.2.1 SEEngine.SERecognition.ParameterSpeed Field

Sets SentiSight (see page 296) recognition speed.

C#

```
public const ushort ParameterSpeed = 50201;
```

8.2.3.1.1.2.2.2.2 SEEngine.SERecognition.ParameterThreshold Field

Specifies recognition threshold.

C#

```
public const ushort ParameterThreshold = 50203;
```

8.2.3.1.1.2.2.2.3 SEEngine.SERecognition.ParameterTransformType Field

Sets transform type to be used in recognition to align shape.

C#

```
public const ushort ParameterTransformType = 50202;
```

8.2.3.1.1.2.2.2.4 SEEngine.SERecognition.ParameterUseTracking Field

Specifies whether to track the object during object recognition or not.

C#

```
public const ushort ParameterUseTracking = 50200;
```

Remarks

This parameter should be used only with a sequence of test images where the neighbouring images differ only slightly. Also, constant light conditions and constant background usually improve tracking results.

Object recognition with tracking requires a constant image size and that size must be set before tracking is started. TrackingImageSize (see page 284) property is used to manipulate that size.

8.2.3.1.1.2.2.2.5 SEEngine.SERecognition.ParamSpeed Field

Defines SentiSight (see page 296) recognition speed.

C#

```
[Obsolete("Use ParameterSpeed instead.")]
public const ushort ParamSpeed = ParameterSpeed;
```

8.2.3.1.1.2.2.6 SEEngine.SERecognition.ParamTransformType Field

Defines type to be used in recognition to align shape.

C#

```
[Obsolete("Use ParameterTransformType instead.")]
public const ushort ParamTransformType = ParameterTransformType;
```

Notes

This parameter is obsolete. Use ParameterTransformType (see page 279) instead.

8.2.3.1.1.2.2.7 SEEngine.SERecognition.ParamUseTracking Field

Parameter used to specify whether to use tracking for recognition.

C#

```
[Obsolete("Use ParameterUseTracking instead.")]
public const ushort ParamUseTracking = ParameterUseTracking;
```

8.2.3.1.1.2.2.3 SERecognition Methods

8.2.3.1.1.2.2.3.1 SEEngine.SERecognition.AddModel Method

Adds a learnt model of the object. These models will be used in Recognize (see page 281) method.

C#

```
public SEStatus AddModel(SEModel model, object modelId);
```

Parameters

Parameters	Description
modelID	A specified model identifier. Later it will be refered in recognition functions as returned value of best recognized model id. Its value may be numeric, literal or any other type. It must be unique for all models added to recognition module.

Remarks

SentiSightEngine.Recognition (see page 292) method Recognize (see page 281) compare a test image (or test model) with models which were aggregated using AddModel. A model is an instance of SEModel (see page 292) class; it can be created (SentiSightEngine.CreateModel (see page 291)) and filled (SentiSightEngine.SELearning.AddToModel (see page 275)). A caller of AddModel must provide an identifier of that model. The identifier will be used in recognition functions returning value of best recognized model id. Moreover, this id can be used in SentiSightEngine.SERecognition.GetRecognitionDetails method to extract detailed information about recognition results.

Also, this identificier may be used to remove the model (RemoveModel (see page 283)) from recognition module.

Exceptions

Exceptions	Description
ArgumentNullException	The model is NULL (see page 82).
ArgumentOutOfRangeException	Provided modelID is already occupied.

See Also

Recognize RemoveModel ModelCount

8.2.3.1.1.2.2.3.2 SEEngine.SERecognition.GetModelIds Method

Returns user all Ids for the model being recognized.

C#

```
public object[] GetModelIds();
```

Returns

Array of model Ids.

8.2.3.1.1.2.2.3.3 Recognize Method

8.2.3.1.1.2.2.3.3.1 SEEngine.SERecognition.Recognize Method (NImage)

Compares a test image with models which were set to the recognition module.

C#

```
public bool Recognize(NImage image);
```

Parameters

Parameters	Description
NImage image	A test image to be recognized. It should be 3 channels RGB image. If tracking is used, the image size must be the same as set to the SentiSightEngine.Recognition (see page 292) object.

Returns

The identificator of the nearest model to the test image. This id is the same as provided to AddModel (see page 280) method.

Remarks

This method may be called after one or more models of learnt objects were set to the SentiSightEngine.Recognition (see page 292) object (using AddModel (see page 280)). The models were learnt using object learning method SentiSightEngine.SELearning.AddToModel (see page 275).

Method Recognize compares a test image (provided as parameter image) with all the models and returns the nearest model identifier. This id can be used later to extract more information about recognition results using GetRecognitionDetails.

If property UseTracking (see page 285) is set to enabled in the SEEngine.Recognition (see page 292) object the Recognize method will keep track of the object in a sequence of test images. The property UseTracking (see page 285) should be used only with a sequence of test images where neighboring images differ only slightly. Object tracking performs better in constant light conditions and constant background. When `Recognize` method is called for the first time (with tracking switched on) the SERecognitionDetails (see page 297) object will indicate a successful recognition, i.e. a parameter Recognized will be true and other data will be meaningful. When Recognize method will be called second time and later on (after a successful first time), it will return information only about successful tracking (returned model id will be the same as the first time and a parameter Tracked in SERecognitionDetails (see page 297) will be true; other parameters are not valid except of CenterX and CenterY). If Recognize method was unable to recognize the object for the first time (Recognized is false) it will try to do it again in the successive calls of the method.

The tracking is used only when SERecSpeed (see page 304) set to high and only one learnt model added to SentiSight (see page 296) recognition engine.

Exceptions

Exceptions	Description
ParameterException	No models were set to recognition module

InvalidOperationException	Tracking is used but image size is not set to the recognition module.
ArgumentException	Tracking is used and the image size does not agree with the size set to the recognition model.
FormatException	Format of image is unsupported.

See Also

Neurotec.SentiSight.SEEngine.SERecognition.AddModel
Neurotec.SentiSight.SEEngine.SERecognition.ParameterUseTracking

8.2.3.1.1.2.2.3.3.2 SEEngine.SERecognition.Recognize Method (NImage, double)

Compares a test image with models which were set to the recognition module.

C#

```
[Obsolete("Use Recognize(NImage) instead.")]
public bool Recognize(NImage image, double similarityThreshold);
```

Parameters

Parameters	Description
NImage image	A test image to be recognized. It should be 3 channels RGB image. If tracking is used, the image size must be the same as set to the SentiSightEngine.Recognition (see page 292) object.
double similarityThreshold	A minimum similarity score indicating successful recognition.

Returns

The identificator of the nearest model to the test image. This id is the same as provided to AddModel (see page 280) method.

Remarks

This method may be called after one or more models of learnt objects were set to the SentiSightEngine.Recognition (see page 292) object (using AddModel (see page 280)). The models were learnt using object learning method SentiSightEngine.SELearning.AddToModel (see page 275).

Method Recognize compares a test image (provided as p image) with all the models and returns the nearest model identifier. This id can be used later to extract more information about recognition results using GetRecognitionDetails.

A provided similarityThreshold is used to discard irrelevant recognition information. If actual similarity score of the test image and its nearest model is higher than similarityThreshold, then GetRecognitionDetails will return meaningful object. If actual similarity score is less than p similarityThreshold, the parameter Recognized of SERecognitionDetails (see page 297) will return false and other data is invalid. A value of "0" of similarityThreshold will result in successful recognition in most cases. The indicator of unsuccessful recognition is a parameter p Recognized returned as false value of SERecognitionDetails (see page 297) object.

If property UseTracking (see page 285) is set to enabled in the SentiSightEngine.Recognition (see page 292) object the Recognize method will keep track of the object in a sequence of test images. The property UseTracking (see page 285) should be used only with a sequence of test images where neighboring images differ only slightly. Object tracking performs better in constant light conditions and constant background. When Recognize method is called for the first time (with tracking switched on) the SERecognitionDetails (see page 297) object will indicate a successful recognition, i.e. a parameter p Recognized will be true and other data will be meaningful. When Recognize method will be called second time and later on (after a successful first time), it will return information only about successful tracking (returned model id will be the same as the first time and a parameter Tracked in SERecognitionDetails (see page 297) will be true; other parameters are not valid except of CenterX and CenterY). If Recognize method was unable to recognize the object for the first time (Recognized is false) it will try to do it again in the successive calls of the method. A parameter similarityThreshold of Recognize method controls a minimum similarity score which results in successful recognition (SERecognitionDetails (see page 297) has a

parameter's Recognized value true). During object tracking a parameter similarityThreshold is meaningless. Before calling Recognize method with tracking, it is neccessary to set the size of the test images. The size must be set to the SentiSightEngine.Recognition (see page 292) object using property TrackingImageSize (see page 284).

The tracking is used only when SERecSpeed (see page 304) set to high and only one learnt model added to SentiSight (see page 296) recognition engine.

Exceptions

Exceptions	Description
ParameterException	No models were set to recognition module
InvalidOperationException	Tracking is used but image size is not set to the recognition module.
ArgumentException	Tracking is used and the image size does not agree with the size set to the recognition model.
FormatException	Format of p image is unsupported.

See Also

Neurotec.SentiSight.SEEngine.SERecognition.AddModel
Neurotec.SentiSight.SEEngine.SERecognition.ParameterUseTracking

8.2.3.1.1.2.2.3.4 SEEngine.SERecognition.RemoveAllModels Method

Removes all models added to the recognition engine.

C#

```
public void RemoveAllModels();
```

8.2.3.1.1.2.2.3.5 SEEngine.SERecognition.RemoveModel Method

Removes the model and it will not be used in #Recognize (see page 281)() method.

C#

```
public void RemoveModel(object modelId);
```

Parameters

Parameters	Description
modelID	A model identificator. It should be the same as provided to AddModel (see page 280) method.

Remarks

SentiSightEngine.Recognition (see page 292) method Recognize (see page 281) compares a test image (or test model) with models which were aggregated using AddModel (see page 280) method. AddModel (see page 280) accepts a p modelID indentifier from a caller. Some aggregated models can be removed using RemoveModel with the same modelID as parameter. At least one model must be set before Recognize (see page 281) method is called.

Exceptions

Exceptions	Description
NArgumentOutOfRangeException	A provided modelID is not valid.
NException	Memory corruption.

See Also

AddModel (see page 280)

8.2.3.1.1.2.2.4 SERecognition Properties

8.2.3.1.1.2.2.4.1 SEEngine.SERecognition.ModelCount Property

Gets a number of models used in recognition module.

C#

```
[Obsolete("Use GetModelIds instead.")]
public int ModelCount;
```

Remarks

SentiSightEngine.Recognition (see page 292) method Recognize (see page 281) compares a test image (or test model) with models which were aggregated using AddModel (see page 280) method. Here referred model is an instance of SEModel (see page 292) class which is created in SentiSightEngine.CreateModel (see page 291) method, filled using SentiSightEngine.SELearning.AddToModel (see page 275) method, and saved/loaded using SEModel.Save and SEModel.Load.

See Also

AddModel RemoveModel Recognize

8.2.3.1.1.2.2.4.2 SEEngine.SERecognition.RecognitionDetails Property

Gets the RecognitionDetailsCollection (see page 279).

C#

```
public RecognitionDetailsCollection RecognitionDetails;
```

Property value

A RecognitionDetailsCollection (see page 279) object.

See Also

RecognitionDetailsCollection (see page 279)

8.2.3.1.1.2.2.4.3 SEEngine.SERecognition.Speed Property

Sets or gets a recognition speed of SentiSight (see page 296) engine.

C#

```
public SERecSpeed Speed;
```

Property value

A recognition speed.

8.2.3.1.1.2.2.4.4 SEEngine.SERecognition.Threshold Property

Gets or sets recognition threshold.

C#

```
public double Threshold;
```

Property value

Recognition (see page 292) threshold value.

See Also

Neurotec.SentiSight.SEEngine.SERecognition.Recognize (see page 281)

8.2.3.1.1.2.2.4.5 SEEngine.SERecognition.TrackingImageSize Property

Gets an image size used by tracking.

C#

```
public Size TrackingImageSize;
```

Remarks

Tracking of the object in a sequence of test images is enabled/disabled through UseTracking (see page 285) property. Actual tracking is performed by sequentially calling Recognize (see page 281) method.

The image size must be set before calling Recognize (see page 281) for the first time; images of the same size must be used in the whole tracking process.

See Also

UseTracking Recognize

8.2.3.1.1.2.2.4.6 SEEngine.SERecognition.TransformType Property

Sets or gets the transform type for SentiSight (see page 296) recognition.

C#

```
public SERecTransformType TransformType;
```

Property value

A transform type.

8.2.3.1.1.2.2.4.7 SEEngine.SERecognition.UseTracking Property

Enables and disables tracking the object during object recognition.

C#

```
public bool UseTracking;
```

Remarks

This identifier is used during object recognition with test images (Recognize (see page 281) method). If the parameter is set to true the tracking of the object in a sequence of test images is enabled. Otherwise, if the parameter is set to false the tracking is disabled and object recognition uses the other algorithm for comparison.

Default parameter value is false, thus object tracking is disabled. This parameter should be used only with a sequence of test images where the neighbouring images differ only slightly. Also, constant light conditions and constant background usually improve tracking results.

See Also

Recognize (see page 281)

8.2.3.1.1.2.3 SEEngine.SESeparation Class

Provides foreground/background separation functionality.

C#

```
public sealed class SESeparation;
```

SESeparation Fields

	Name	Description
	ParameterUseAdaptiveAlg (see page 286)	Sets a parameter to use an adaptive algorithm for separation.

SESeparation Methods

	Name	Description
	AccumulateBackground (see page 286)	Accumulates background for foreground/background separation process.
	GetObjectModelSize (see page 287)	Gets a minimum size of buffer needed to save a colour model aggregated using Separate (see page 289) method.
	LoadHolderModel (see page 287)	Loads holder model from memory buffer.
	LoadObjectModel (see page 288)	Loads separation model from memory buffer.
	ResetBackgroundModel (see page 288)	Resets current background model.
	ResetHolderModel (see page 288)	Resets current holder model.
	ResetObjectModel (see page 289)	Resets current object model.
	SaveObjectModel (see page 289)	Writes a colour model of the object into memory buffer.
	Separate (see page 289)	Separates foreground (an object) from background and extracts image mask which can be used in object learning (SentiSightEngine.SELearning.AddToModel (see page 275) methods).

SESeparation Properties

	Name	Description
	ImageSize (see page 290)	Image size used in whole separation process.
	UseAdaptiveAlg (see page 290)	Indicates whether to use adaptive algorithm.

8.2.3.1.1.2.3.1 SESeparation Fields

8.2.3.1.1.2.3.1.1 SEEngine.SESeparation.ParameterUseAdaptiveAlg Field

Sets a parameter to use an adaptive algorithm for separation.

C#

```
public const ushort ParameterUseAdaptiveAlg = 50000;
```

Remarks

If value is NTrue (see page 82) the separation algorithm automatically adapts to changing ligting conditions. The default value for this parameter is NFalse (see page 82),

8.2.3.1.1.2.3.2 SESeparation Methods

8.2.3.1.1.2.3.2.1 SEEngine.SESeparation.AccumulateBackground Method

Accumulates background for foreground/background separation process.

C#

```
public bool AccumulateBackground(NImage image, out NImage background);
```

Parameters

Parameters	Description
Image	An image of background. It is a current image of background for accumulation. The image should have the same size as it was set using ImageSize (see page 290) property. Also, it should be 3 channels RGB colour image.
Background	Currently accumulated background.

Returns

Identifier specifying whether more images of background are needed to build accurate background model. If returned value is false, then background accumulation may be stopped. If returned value is true, background accumulation should be continued.

Remarks

The size of the image should be set before starting separation process (AccumulateBackground is the first step in this process). Image size can be accessed as ImageSize (see page 290) property. Background images contain some noise, so background accumulation is intended for elimination of that noise. Some images of the same background under the same light conditions should be fed to AccumulateBackground method in order to build accurate inner background model. This model will be used later in foreground/background separation (Separate (see page 289) method).

Exceptions

Exceptions	Description
NArgumentNullException	A provided image is NULL (see page 82)
NFormatException	The image format is unsupported.
NInvalidOperationException	Extraction of data from the image failed.
NException	Other failure.

See Also

Separate ResetBackgroundModel

8.2.3.1.1.2.3.2.2 SEEngine.SESeparation.GetObjectModelSize Method

Gets a minimum size of buffer needed to save a colour model aggregated using Separate (see page 289) method.

C#

```
public int GetObjectModelSize();
```

Remarks

Foreground/background separation (Separate (see page 289)()) builds a colour model of the object. This model can be retrieved using function SaveModel with provided buffer.

Exceptions

Exceptions	Description
NException	Memory corruption.

See Also

SaveModel

8.2.3.1.1.2.3.2.3 SEEngine.SESeparation.LoadHolderModel Method

Loads holder model from memory buffer.

C#

```
public void LoadHolderModel(byte[] buffer);
```

Parameters

Parameters	Description
byte[] buffer	A data buffer containing a colour model of the holder.

Remarks

Foreground/background separation method Separate (see page 289) is able to exclude a holder (a hand or a stick which holds the object) from the object and consider it as background.

At the beginning, a colour model of the holder must be learnt. Therefore, a number of images of the holder must be fed to Separate (see page 289) method which separates the holder from background and builds the colour model of it. The colour model of the holder will be saved in the SentiSightEngine.SESeparation (see page 285) object. It can be retrieved from the SentiSightEngine.SESeparation (see page 285) object using SaveModel method with provided data buffer. The colour model can be set as holder using LoadHolderModel method with the same data buffer as a parameter.

Exceptions

Exceptions	Description
NParameterError	A p Buffer size is invalid.
NError	A p Buffer data is invalid.

See Also

SaveModel Separate ResetHolderModel

8.2.3.1.1.2.3.2.4 SEEngine.SESeparation.LoadObjectModel Method

Loads separation model from memory buffer.

C#

```
public void LoadObjectModel(byte[] buffer);
```

Parameters

Parameters	Description
byte[] buffer	A byte array which contains s separation model.

8.2.3.1.1.2.3.2.5 SEEngine.SESeparation.ResetBackgroundModel Method

Resets current background model.

C#

```
public void ResetBackgroundModel();
```

Remarks

Foreground/background separation method Separate (see page 289) keeps an accumulated background model (built with AccumulateBackground (see page 286) method). The background can be cleaned using ResetBackgroundModel method. After that, one needs to accumulate background again.

See Also

AccumulateBackground (see page 286)

8.2.3.1.1.2.3.2.6 SEEngine.SESeparation.ResetHolderModel Method

Resets current holder model.

C#

```
public void ResetHolderModel();
```

Remarks

Foreground/background separation method Separate (see page 289) is able to exclude a holder (a hand or a stick which holds the object) from the object and consider it as background. A holder model can be set using LoadHolderModel (see page 287) method. If one decided not to use holder, a holder can be removed from the SentiSightEngine.Separation (see page 292) object by calling ResetHolderModel. By default, no holder is set to the SentiSightEngine.Separation (see page 292) object since the holder must be learnt on the same background and in the same light conditions as the object will be separated.

See Also

Separate LoadHolderModel

8.2.3.1.1.2.3.2.7 SEEngine.SESeparation.ResetObjectModel Method

Resets current object model.

C#

```
public void ResetObjectModel();
```

Remarks

Foreground/background separation method Separate (see page 289) builds a colour model of the object. Method ResetObjectModel resets the colour model and it becomes empty.

See Also

Separate LoadModel

8.2.3.1.1.2.3.2.8 SEEngine.SESeparation.SaveObjectModel Method

Writes a colour model of the object into memory buffer.

C#

```
public int SaveObjectModel(byte[] buffer);
```

Parameters

Parameters	Description
byte[] buffer	A buffer where data of the colour model will be written. A buffer must be allocated before calling the method. A size of the buffer must be at least as returned from GetModelSize size.

Remarks

Foreground/background separation method Separate (see page 289) builds a colour model of the object. That model can be saved (using SaveModel method) and can be loaded (LoadModel or LoadHolderModel (see page 287) methods) later. A buffer provided to the SaveModel method should be allocated by the caller. A size of the buffer must be equal or greater than that returned by GetModelSize method. If buffer size is greater than minimum required, then only first (minimum size) bytes will be filled by the function.

See Also

LoadModel LoadHolderModel GetModelSize

8.2.3.1.1.2.3.2.9 SEEngine.SESeparation.Separate Method

Separates foreground (an object) from background and extracts image mask which can be used in object learning (SentiSightEngine.SELearning.AddToModel (see page 275) methods).

C#

```
public bool Separate(NImage image, out NImage mask, out NImage background, out NImage
segmented);
```

Parameters

Parameters	Description
Image	An image of the object to be separated. It should be the same size as set to ImageSize (see page 290) property. Also, the image should be 3 channels RGB colour image.
Mask	A grayscale image of the same size as input one, with "0" values in the grid places of background and "255" values in the grid places of the object.

Background	An image showing current background (the accumulated one). This image is 3 channel RGB image of the same size as the input one.
Segmented	An image with applied mask. The grid places of background are gray; the places of the object present the object itself.

Returns

The value true indicates a valid mask, i.e. it contains at least one pixel of the object. The value false indicates that all pixels in the mask are black, i.e. there was no object separated.

Remarks

It separates the object from the background. If a holder model was set (using LoadHolderModel (see page 287)) it will be excluded from the object and considered as background.

This function also builds a colour model of the object. Later this model can be retrieved calling SaveModel with provided buffer. This colour model can be reset by calling ResetObjectModel (see page 289) or loaded from memory by calling LoadModel for appendance.

Exceptions

Exceptions	Description
ArgumentNullException	A provided image is NULL (see page 82).
InvalidOperationException	The image size is not set to the separation module.
ArgumentException	A provided image is not valid. Either, the size does not match ImageSize (see page 290) property, or image is not 3 channels RGB colour image.

See Also

AccumulateBackground LoadHolderModel SaveModel ResetObjectModel ResetHolderModel SentiSightEngine.SELearning.AddToModel

8.2.3.1.1.2.3.3 SESeparation Properties

8.2.3.1.1.2.3.3.1 SEEngine.SESeparation.ImageSize Property

Image size used in whole separation process.

C#

```
public Size ImageSize;
```

Remarks

The image size must be set before starting background accumulation (first step of foreground/background separation process). Images of the same size must be used for background accumulation and also for separation functions.

See Also

SESeparation.AccumulateBackground (see page 286)

8.2.3.1.1.2.3.3.2 SEEngine.SESeparation.UseAdaptiveAlg Property

Indicates whether to use adaptive algorithm.

C#

```
public bool UseAdaptiveAlg;
```

Property value

A boolean value indicating whether to use adaptive algorithm.

8.2.3.1.1.3 SEEngine Methods

8.2.3.1.1.3.1 SEEngine.CreateModel Method

Creates SEModel (see page 292) class instance - the model where the learnt object will be kept.

C#

```
public SEModel CreateModel();
```

Returns

New SEModel (see page 292) instance. It should be filled during object learning process.

Remarks

The model will be used later object learning process. SentiSightEngine.SELearning.AddToModel (see page 275) method will add data to this model. After object learning is complete, the model can be saved (SEModel.Save method) and loaded (SEModel.Load method).

See Also

SentiSightEngine.SELearning.AddToModel SentiSightEngine.SELearning.GeneralizeModel SEModel.Save SEModel.Load

8.2.3.1.1.3.2 FromHandle Method

8.2.3.1.1.3.2.1 SEEngine.FromHandle Method (IntPtr)

Creates new SEEngine (see page 272) object from specified handle.

C#

```
public static SEEngine FromHandle(IntPtr handle);
```

Parameters

Parameters	Description
IntPtr handle	Handle (see page 233) used to create SEEngine (see page 272) object.

Returns

Created SEEngine (see page 272) object.

8.2.3.1.1.3.2.2 SEEngine.FromHandle Method (IntPtr, bool)

Creates new SEEngine (see page 272) object from specified handle.

C#

```
public static SEEngine FromHandle(IntPtr handle, bool ownsHandle);
```

Parameters

Parameters	Description
IntPtr handle	Handle (see page 233) used to create SEEngine (see page 272) object.
bool ownsHandle	Handle (see page 233) owner.

Returns

Created SEEngine (see page 272) object.

8.2.3.1.1.3.3 SEEngine.GetInfo Method

Gets NLibraryInfo (see page 229) object which contains information about SentiSight (see page 296) library.

C#

```
[Obsolete("Use SentiSight.GetInfo() instead.")]
public static NLibraryInfo GetInfo();
```

Returns

NLibraryInfo (see page 229) structure which contains library information.

8.2.3.1.1.4 SEEngine Properties

8.2.3.1.1.4.1 SEEngine.Learning Property

Gets SentiSightEngine learning module.

C#

```
public SELearning Learning;
```

See Also

SentiSightEngine.SESeparation

8.2.3.1.1.4.2 SEEngine.Recognition Property

Gets SentiSightEngine recognition module.

C#

```
public SERecognition Recognition;
```

See Also

SentiSightEngine.Recognition

8.2.3.1.1.4.3 SEEngine.Separation Property

Gets SentiSightEngine separation module.

C#

```
public SESeparation Separation;
```

See Also

SentiSightEngine.SESeparation

8.2.3.1.2 SEModel Class

Keeps learnt object models.

C#

```
public sealed class SEModel : NObject, ICloneable;
```

Remarks

An instance of SEModel class can be created through SentiSightEngine.CreateModel method of SentiSightEngine object. A SEModel object must be created before object learning starts. The SEModel object can be saved and loaded using Save (see page 295) and Load (see page 294) methods.

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.
	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

SEModel Class

	Name	Description
	Clone (see page 293)	Returns a copy of SEModel object.
S	FromHandle (see page 293)	Creates new SEModel object from specified handle.
	GetSize (see page 294)	Gets a minimum buffer size needed to save the model.
	Load (see page 294)	Retrieves data of the model from memory.
	Save (see page 295)	Saves data of the model to memory.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

SEModel Class

	Name	Description
	IsEmpty (see page 295)	Checks if the SEModel is empty.
	IsLocked (see page 296)	Checks if the SEModel is locked.

8.2.3.1.2.1 SEModel Methods

8.2.3.1.2.1.1 SEModel.Clone Method

Returns a copy of SEModel (see page 292) object.

C#

```
public object Clone();
```

8.2.3.1.2.1.2 FromHandle Method

8.2.3.1.2.1.2.1 SEModel.FromHandle Method (IntPtr)

Creates new SEModel (see page 292) object from specified handle.

C#

```
public static SEModel FromHandle(IntPtr handle);
```

Parameters

Parameters	Description
IntPtr handle	Handle (see page 233) used to create SEModel (see page 292) object.

Returns

Created SEModel (see page 292) object.

8.2.3.1.2.1.2.2 SEModel.FromHandle Method (IntPtr, bool)

Creates new SEModel (see page 292) object from specified handle.

C#

```
public static SEModel FromHandle(IntPtr handle, bool ownsHandle);
```

Parameters

Parameters	Description
IntPtr handle	Handle (see page 233) used to create SEModel (see page 292) object.
bool ownsHandle	Handle (see page 233) owner.

Returns

Created SEModel (see page 292) object.

8.2.3.1.2.1.3 SEModel.GetSize Method

Gets a minimum buffer size needed to save the model.

C#

```
public int GetSize();
```

Remarks

This method may be called after creation of the model (SentiSightEngine.CreateModel method) and object learning (AddToModel methods of the SEModel (see page 292) object). GetSize method returns a minimum size of a buffer needed to save the model using method Save (see page 295).

See Also

Save (see page 295)

8.2.3.1.2.1.4 SEModel.Load Method

Retrieves data of the model from memory.

C#

```
public void Load(byte[] buffer);
```

Parameters

Parameters	Description
byte[] buffer	A data buffer to be read.

Remarks

A valid data buffer containing information about the model must be provided to the method. A data buffer may be filled using Save (see page 295) method after object learning is completed.

Exceptions

Exceptions	Description
OutOfMemoryException	A provided buffer is invalid.

See Also

WriteToMemory

8.2.3.1.2.1.5 Save Method

8.2.3.1.2.1.5.1 SEModel.Save Method ()

Saves data of the model to memory.

C#

```
public byte[] Save();
```

Exceptions

Exceptions	Description
OutOfMemory	A data corruption error.

See Also

LoadFromMemory, MinBufferSize

8.2.3.1.2.1.5.2 SEModel.Save Method (byte[])

Saves data of the model to memory.

C#

```
public int Save(byte[] buffer);
```

Parameters

Parameters	Description
byte[] buffer	A data buffer where data of the model will be written.

Remarks

This method can be called after object learning is complete. Firstly, a required size of a data buffer should be retrieved using GetSize (see page 294) method. Then a data buffer should be allocated or used previously allocated one. After that, Save method can be called with the data buffer as parameter. If data buffer is larger than minimum required size, then only first (the number is returned by GetSize (see page 294) method bytes of it will be occupied.

Exceptions

Exceptions	Description
ArgumentException	A buffer is too small to write the data of the model.
OutOfMemory	A data corruption error.

See Also

LoadFromMemory, MinBufferSize

8.2.3.1.2.2 SEModel Properties

8.2.3.1.2.2.1 SEModel.IsEmpty Property

Checks if the SEModel (see page 292) is empty.

C#

```
public bool IsEmpty;
```

Property value

A boolean value indicating whether SEModel (see page 292) is empty.

8.2.3.1.2.2.2 SEModel.IsLocked Property

Checks if the SEModel (see page 292) is locked.

C#

```
public bool IsLocked;
```

Property value

A boolean value indicating whether a model is locked.

8.2.3.1.3 SentiSight Class

General class for SentiSight library.

C#

```
public static class SentiSight;
```

SentiSight Fields

	Name	Description
	DllName (see page 296)	Name of DLL containing unmanaged part of this class.

SentiSight Methods

	Name	Description
S	GetInfo (see page 296)	Gets NLibraryInfo (see page 229) object which contains information about SentiSight library.

8.2.3.1.3.1 SentiSight Fields

8.2.3.1.3.1.1 SentiSight.DllName Field

Name of DLL containing unmanaged part of this class.

C#

```
public const string DllName = "SentiSight";
```

8.2.3.1.3.2 SentiSight Methods

8.2.3.1.3.2.1 SentiSight.GetInfo Method

Gets NLibraryInfo (see page 229) object which contains information about SentiSight (see page 296) library.

C#

```
public static NLibraryInfo GetInfo();
```

Returns

NLibraryInfo (see page 229) structure which contains library information.

8.2.3.1.4 SERecognitionDetails Class

Provides detailed information about recognition (Recognize method) results.

C#

```
public sealed class SERecognitionDetails : NObject;
```

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.
	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

SERecognitionDetails Class

	Name	Description
	GetImageToModelTransform (see page 297)	Fills 9 elements 3X3 row wise matrix representing transform.
	GetModelToImageTransform (see page 298)	Fills 9 element 3X3 row wise matrix representing transform.
	GetShape (see page 298)	Gets a shape with a particular transform type.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

SERecognitionDetails Class

	Name	Description
	IsTracked (see page 299)	Checks if the recognition details were tracked.
	ModelId (see page 299)	Gets user specified Id for the model being recognized.
	Score (see page 299)	Gets similarity score.
	Shape (see page 299)	Gets a SEShape (see page 300) object.
	TransformType (see page 300)	Gets a type of transform found by recognition process (shape alignment and transform matrices).

8.2.3.1.4.1 SERecognitionDetails Methods

8.2.3.1.4.1.1 GetImageToModelTransform Method

8.2.3.1.4.1.1.1 SERecognitionDetails.GetImageToModelTransform Method ()

Fills 9 elements 3X3 row wise matrix representing transform.

C#

```
public double[,] GetImageToModelTransform();
```

Returns

3X3 row wise matrix which contains double values of transformation.

8.2.3.1.4.1.1.2 SERecognitionDetails.GetImageToModelTransform Method (SERecTransformType)

Fills 9 elements 3X3 row wise matrix representing transform.

C#

```
public double[,] GetImageToModelTransform(SERecTransformType transformType);
```

Parameters

Parameters	Description
SERecTransformType transformType	A recognition transform type.

Returns

3X3 row wise matrix which contains double values of transformation.

8.2.3.1.4.1.2 GetModelToImageTransform Method

8.2.3.1.4.1.2.1 SERecognitionDetails.GetModelToImageTransform Method ()

Fills 9 element 3X3 row wise matrix representing transform.

C#

```
public double[,] GetModelToImageTransform();
```

Returns

3X3 row wise matrix which contains double values of transformation.

8.2.3.1.4.1.2.2 SERecognitionDetails.GetModelToImageTransform Method (SERecTransformType)

Fills 9 element 3X3 row wise matrix representing transform.

C#

```
public double[,] GetModelToImageTransform(SERecTransformType transformType);
```

Parameters

Parameters	Description
SERecTransformType transformType	A recognition transform type.

Returns

3X3 row wise matrix which contains double values of transformation.

8.2.3.1.4.1.3 SERecognitionDetails.GetShape Method

Gets a shape with a particular transform type.

C#

```
public SEShape GetShape(SERecTransformType transformType);
```

Parameters

Parameters	Description
SERecTransformType transformType	A transform type (this transform type is used in recognition process to align shapes or transform images).

Returns

A SERecTransformType (see page 304) object.

See Also

SERecTransformType (see page 216)

8.2.3.1.4.2 SERecognitionDetails Properties

8.2.3.1.4.2.1 SERecognitionDetails.IsTracked Property

Checks if the recognition details were tracked.

C#

```
public bool IsTracked;
```

Property value

A bool value indicating if the recognition details were tracked.

8.2.3.1.4.2.2 SERecognitionDetails.ModelId Property

Gets user specified Id for the model being recognized.

C#

```
public object ModelId;
```

Property value

A model specified by Id.

See Also

SERecDetailsGetModelId (see page 185)

8.2.3.1.4.2.3 SERecognitionDetails.Score Property

Gets similarity score.

C#

```
public double Score;
```

Property value

A similarity score.

8.2.3.1.4.2.4 SERecognitionDetails.Shape Property

Gets a SEShape (see page 300) object.

C#

```
public SEShape Shape;
```

Property value

A SEShape (see page 300) object.

8.2.3.1.4.2.5 SERecognitionDetails.TransformType Property

Gets a type of transform found by recognition process (shape alignment and transform matrices).

C#

```
public SERecTransformType TransformType;
```

Property value

A type of transform found by recognition process.

See Also

SERecDetailsGetTransformType (see page 188)

8.2.3.1.5 SEShape Class

Provides functions for working with SentiSight (see page 296) shape objects.

C#

```
public sealed class SEShape : NObject, ICloneable;
```

Methods

SEShape Class

	Name	Description
	SEShape (see page 301)	Initializes a new instance of the SEShape class.

SEShape Classes

SEShape Class

	Name	Description
	PointCollection (see page 301)	Represents a collection of the SEShape point values that can be individually accessed by index.

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.
	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

SEShape Class

	Name	Description
	Clone (see page 301)	Returns a copy of SEShape object.
S	FromHandle (see page 301)	Retrieves a new SEShape object referenced by the specified handle.
	Rotate (see page 302)	Rotates shape by amount of radians around the center of the shape.
	Scale (see page 302)	Scales shape around the center of the shape, below 1 shrinks, above 1 enlarges the shape.

	TestPoint (see page 302)	Gets test point and shape relative position.
	Translate (see page 303)	Translates the shape. Negative distance is translated to the coordinate origin side, positive - out of the coordinate origin.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

SEShape Class

	Name	Description
	Center (see page 303)	Gets automatically calculated shape center point.
	Heading (see page 303)	Gets or sets user specified orientation of the shape in radians 0..2pi.
	IsLocked (see page 303)	Gets if the shape is locked.
	IsValid (see page 303)	Checks if a shape is valid.
	Points (see page 303)	Retrieves points collection.

8.2.3.1.5.1 SEShape.SEShape Constructor

Initializes a new instance of the SEShape class.

C#

```
public SEShape();
```

8.2.3.1.5.2 SEShape Classes

8.2.3.1.5.2.1 SEShape.PointCollection Class

Represents a collection of the SEShape (see page 300) point values that can be individually accessed by index.

C#

```
public sealed class PointCollection : NStructCollection<PointD>;
```

8.2.3.1.5.3 SEShape Methods

8.2.3.1.5.3.1 SEShape.Clone Method

Returns a copy of SEShape (see page 300) object.

C#

```
public object Clone();
```

8.2.3.1.5.3.2 SEShape.FromHandle Method

Retrieves a new SEShape (see page 300) object referenced by the specified handle.

C#

```
public static SEShape FromHandle(IntPtr handle);
```

Parameters

Parameters	Description
IntPtr handle	The handle of the SEShape (see page 300) object.

Returns

The SEShape (see page 300) object identified by a handle.

8.2.3.1.5.3.3 SEShape.Rotate Method

Rotates shape by amount of radians around the center of the shape.

C#

```
public void Rotate(double angle);
```

Parameters

Parameters	Description
double angle	An angle in radians to rotate a shape.

8.2.3.1.5.3.4 SEShape.Scale Method

Scales shape around the center of the shape, below 1 shrinks, above 1 enlarges the shape.

C#

```
public void Scale(double scale);
```

Parameters

Parameters	Description
double scale	Shape scale.

8.2.3.1.5.3.5 TestPoint Method

8.2.3.1.5.3.5.1 SEShape.TestPoint Method (PointD)

Gets test point and shape relative position.

C#

```
public double TestPoint(PointD point);
```

Parameters

Parameters	Description
PointD point	A point which should be tested.

Returns

A measured distance.

8.2.3.1.5.3.5.2 SEShape.TestPoint Method (PointD, bool)

Gets test point and shape relative position.

C#

```
public double TestPoint(PointD point, bool measureDistance);
```

Parameters

Parameters	Description
PointD point	A point which should be tested.
bool measureDistance	If measureDistance is false distance is -1, 0, 1. If measureDistance is true distance is negative, zero, positive Euclidean distance, for point being outside, on the edge and inside of the shape respectively.

Returns

A measured distance.

8.2.3.1.5.3.6 SEShape.Translate Method

Translates the shape. Negative distance is translated to the coordinate origin side, positive - out of the coordinate origin.

C#

```
public void Translate(PointD distance);
```

Parameters

Parameters	Description
PointD distance	A distance to translated shape.

8.2.3.1.5.4 SEShape Properties

8.2.3.1.5.4.1 SEShape.Center Property

Gets automatically calculated shape center point.

C#

```
public PointD Center;
```

Property value

A shape center point.

8.2.3.1.5.4.2 SEShape.Heading Property

Gets or sets user specified orientation of the shape in radians 0..2pi.

C#

```
public double Heading;
```

Property value

An orientation of the shape in radians.

8.2.3.1.5.4.3 SEShape.IsLocked Property

Gets if the shape is locked.

C#

```
public bool IsLocked;
```

Property value

A boolean value indicating if the shape is locked.

8.2.3.1.5.4.4 SEShape.IsValid Property

Checks if a shape is valid.

C#

```
public bool IsValid;
```

Property value

A boolean value indicating if a shape is valid.

8.2.3.1.5.4.5 SEShape.Points Property

Retrieves points collection.

C#

```
public PointCollection Points;
```

Property value

Points collection.

8.2.3.2 Structs, Records, Enums

8.2.3.2.1 Neurotec.SentiSight.SELrnMode Enumeration

Enumerates different type of learning parameters.

C#

```
public enum SELrnMode {
  LowProfile = 0,
  HighProfile = 1,
  HighProfileEx = 2
}
```

Members

Members	Description
LowProfile = 0	Adds additional not rotation invariant information, improves recognition quality for not rotated objects (has no impact on rotated object recognition). About 5%-10% slower and has about 20%-60% bigger template size (this information is used with low recognition speed only).
HighProfile = 1	Fastest and has the smallest template size, suitable in most situations.
HighProfileEx = 2	Adds additional information, improves recognition for all types of objects. About 50%-100% slower and has about 5%-10% bigger template size (this information is used with low recognition speed only).

8.2.3.2.2 Neurotec.SentiSight.SERecSpeed Enumeration

Enumerates SentiSight (see page 296) recognition speed parameters.

C#

```
public enum SERecSpeed {
  Low = 0,
  High = 256
}
```

Members

Members	Description
Low = 0	The lowest recognition speed.
High = 256	The highest recognition speed.

8.2.3.2.3 Neurotec.SentiSight.SERecTransformType Enumeration

Transform types used in recognition process to align shape or transform images.

C#

```
public enum SERecTransformType {
}
```

Remarks

If auto, selects best possible transform, if particular transform is selected but current condition do not allow to perform it simple transform is selected. Transform complexity: Similarity->Affine->Perspective.

8.2.3.2.4 Neurotec.SentiSight.SEStatus Enumeration

Enumerates different types of SentiSight (see page 296) update status parameters.

C#

```
public enum SEStatus {
  Succeeded = 1,
  MaskIsEmpty = 2,
  ShapeNotValid = 3,
  NothingToLearn = 4,
  ModelIsEmpty = 5
}
```

Members

Members	Description
Succeeded = 1	Indicates that operation succeeded.
MaskIsEmpty = 2	Indicates that model was not updated with new information because passed mask is empty, all zero.
ShapeNotValid = 3	Indicates that model was not updated with new information because passed shape is not valid.
NothingToLearn = 4	Indicates that no data to learned was indicated.
ModelIsEmpty = 5	Indicates that model contain no objects.

8.2.4 Neurotec.DeviceManager Namespace

Classes under this namespace provides functionality for working with cameras.

Module

.NET (see page 219)

Classes

	Name	Description
	Camera (see page 305)	Provides methods for handling a camera as a physical device.
	CameraMan (see page 314)	Provides functionality for managing cameras.
	CameraVideoFormat (see page 316)	Camera (see page 305)'s video format structure.

8.2.4.1 Classes

8.2.4.1.1 Camera Class

Provides methods for handling a camera as a physical device.

C#

```
public sealed class Camera : NObject;
```

Remarks

One instance represents one physical device.

Camera Fields

Camera Class

	Name	Description
	ParameterAutomaticSettings (see page 307)	Identifier that enables automatic control of camera settings.
	ParameterExposure (see page 308)	Identifier that controls camera exposure (total amount of light allowed to fall on the camera during taking video frame). The exposure range can be acquired using Neurotec.DeviceManager.Camera.ParameterExposureMin (see page 308) and Neurotec.DeviceManager.Camera.ParameterExposureMax (see page 308).
	ParameterExposureMax (see page 308)	Identifier specifying maximal camera exposure.
	ParameterExposureMin (see page 308)	Identifier specifying minimal camera exposure.
	ParameterGain (see page 308)	Identifier that controls camera gain. The gain range can be acquired using Neurotec.DeviceManager.Camera.ParameterGainMix and Neurotec.DeviceManager.Camera.ParameterGainMax (see page 308).
	ParameterGainMax (see page 308)	Identifier specifying maximal camera gain.
	ParameterGainMin (see page 309)	Identifier specifying minimal camera gain.
	ParameterIpChannelId (see page 309)	Identifier specifying Cisco IP camera streaming Channel Id (see page 312).
	ParameterIpChannelName (see page 309)	Identifier specifying Cisco IP camera streaming Channel name.
	ParameterIpPassword (see page 309)	Identifier specifying Cisco IP camera password.
	ParameterIpUserName (see page 309)	Identifier specifying Cisco IP camera user name.
	ParameterMirrorHorizontal (see page 309)	Identifier specifying horizontal mirroring of the frame.
	ParameterMirrorVertical (see page 309)	Identifier specifying horizontal mirroring of the frame.
	ParameterVideoDropFrames (see page 309)	Identifier specifying whether Virtual video file camera should drop frames or not. Default is set to False so that every returned frame is the next frame in video file.
	ParameterVideoFileName (see page 310)	Identifier specifying Virtual video file camera video file name.

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.

	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

Camera Class

	Name	Description
	GetCurrentFrame (see page 310)	Gets current frame from capture device.
	GetVideoFormats (see page 310)	Gets available video formats for the capture device.
	StartCapturing (see page 310)	Starts capturing.
	StopCapturing (see page 310)	Stops capturing.
	ToString (see page 310)	Retrieves a string representation of the object.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

Camera Class

	Name	Description
	AutomaticSettings (see page 311)	Gets or sets automatic control of camera settings.
	Exposure (see page 311)	Gets or sets camera exposure (total amount of light allowed to fall on the camera during taking video frame). The exposure range can be acquired using Neurotec.DeviceManager.Camera.ExposureMax (see page 311) and Neurotec.DeviceManager.Camera.ExposureMix..
	ExposureMax (see page 311)	Gets maximal camera exposure.
	ExposureMin (see page 311)	Gets minimal camera exposure.
	Gain (see page 311)	Gets or sets camera gain. The gain range can be acquired using Neurotec.DeviceManager.CameraParameter.GainMin (see page 312) and Neurotec.DeviceManager.Camera.GainMax (see page 312).
	GainMax (see page 312)	Gets maximal camera gain.
	GainMin (see page 312)	Gets minimal camera gain.
	Id (see page 312)	Gets associated device identifier.
	IpChannelId (see page 312)	Gets/sets Cisco IP camera streaming Channel Id (see page 312).
	IpChannelName (see page 313)	Gets/sets Cisco IP camera streaming Channel name.
	IpPassword (see page 313)	Gets/sets Cisco IP camera password.
	IpUserName (see page 313)	Gets/sets Cisco IP camera user name.
	IsCapturing (see page 313)	Checks camera status.
	MirrorHorizontal (see page 313)	Gets or sets horizontal mirroring of the frame.
	MirrorVertical (see page 313)	Gets or sets vertical mirroring of the frame.
	Owner (see page 314)	Gets owner of the object.
	VideoFormat (see page 314)	Gets or sets camera video format. Can not be set while the camera is capturing.

8.2.4.1.1.1 Camera Fields

8.2.4.1.1.1.1 Camera.ParameterAutomaticSettings Field

Identifier that enables automatic control of camera settings.

C#

```
public const ushort ParameterAutomaticSettings = 10301;
```

8.2.4.1.1.1.2 Camera.ParameterExposure Field

Identifier that controls camera exposure (total amount of light allowed to fall on the camera during taking video frame). The exposure range can be acquired using Neurotec.DeviceManager.Camera.ParameterExposureMin (see page 308) and Neurotec.DeviceManager.Camera.ParameterExposureMax (see page 308).

C#

```
public const ushort ParameterExposure = 10420;
```

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.1.3 Camera.ParameterExposureMax Field

Identifier specifying maximal camera exposure.

C#

```
public const ushort ParameterExposureMax = 10422;
```

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.1.4 Camera.ParameterExposureMin Field

Identifier specifying minimal camera exposure.

C#

```
public const ushort ParameterExposureMin = 10421;
```

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.1.5 Camera.ParameterGain Field

Identifier that controls camera gain. The gain range can be acquired using Neurotec.DeviceManager.Camera.ParameterGainMix and Neurotec.DeviceManager.Camera.ParameterGainMax (see page 308).

C#

```
public const ushort ParameterGain = 10410;
```

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.1.6 Camera.ParameterGainMax Field

Identifier specifying maximal camera gain.

C#

```
public const ushort ParameterGainMax = 10412;
```

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.1.7 Camera.ParameterGainMin Field

Identifier specifying minimal camera gain.

C#

```
public const ushort ParameterGainMin = 10411;
```

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.1.8 Camera.ParameterIpChannelId Field

Identifier specifying Cisco IP camera streaming Channel Id (see page 312).

C#

```
public const ushort ParameterIpChannelId = 10503;
```

8.2.4.1.1.1.9 Camera.ParameterIpChannelName Field

Identifier specifying Cisco IP camera streaming Channel name.

C#

```
public const ushort ParameterIpChannelName = 10504;
```

8.2.4.1.1.1.10 Camera.ParameterIpPassword Field

Identifier specifying Cisco IP camera password.

C#

```
public const ushort ParameterIpPassword = 10502;
```

8.2.4.1.1.1.11 Camera.ParameterIpUserName Field

Identifier specifying Cisco IP camera user name.

C#

```
public const ushort ParameterIpUserName = 10501;
```

8.2.4.1.1.1.12 Camera.ParameterMirrorHorizontal Field

Identifier specifying horizontal mirroring of the frame.

C#

```
public const ushort ParameterMirrorHorizontal = 10200;
```

8.2.4.1.1.1.13 Camera.ParameterMirrorVertical Field

Identifier specifying horizontal mirroring of the frame.

C#

```
public const ushort ParameterMirrorVertical = 10201;
```

8.2.4.1.1.1.14 Camera.ParameterVideoDropFrames Field

Identifier specifying whether Virtual video file camera should drop frames or not. Default is set to False so that every returned frame is the next frame in video file.

C#

```
public const ushort ParameterVideoDropFrames = 10602;
```

8.2.4.1.1.15 Camera.ParameterVideoFileName Field

Identifier specifying Virtual video file camera video file name.

C#

```
public const ushort ParameterVideoFileName = 10601;
```

8.2.4.1.2 Camera Methods

8.2.4.1.2.1 Camera.GetCurrentFrame Method

Gets current frame from capture device.

C#

```
public NImage GetCurrentFrame();
```

Returns

A frame which type is NImage (see page 251).

8.2.4.1.2.2 Camera.GetVideoFormats Method

Gets available video formats for the capture device.

C#

```
public CameraVideoFormat[] GetVideoFormats();
```

Returns

An array that contains video format values.

8.2.4.1.2.3 Camera.StartCapturing Method

Starts capturing.

C#

```
public void StartCapturing();
```

8.2.4.1.2.4 Camera.StopCapturing Method

Stops capturing.

C#

```
public void StopCapturing();
```

8.2.4.1.2.5 Camera.ToString Method

Retrieves a string representation of the object.

C#

```
public override string ToString();
```

Returns

A string representation of an object.

8.2.4.1.3 Camera Properties

8.2.4.1.1.3.1 Camera.AutomaticSettings Property

Gets or sets automatic control of camera settings.

C#

```
public bool AutomaticSettings;
```

Property value

true if automatic control of camera is enabled; otherwise **false**.

8.2.4.1.1.3.2 Camera.Exposure Property

Gets or sets camera exposure (total amount of light allowed to fall on the camera during taking video frame). The exposure range can be acquired using Neurotec.DeviceManager.Camera.ExposureMax (see page 311) and Neurotec.DeviceManager.Camera.ExposureMix..

C#

```
public uint Exposure;
```

Property value

Integer specifying camera exposure.

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.3.3 Camera.ExposureMax Property

Gets maximal camera exposure.

C#

```
public uint ExposureMax;
```

Property value

Integer specifying maximal possible camera exposure.

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.3.4 Camera.ExposureMin Property

Gets minimal camera exposure.

C#

```
public uint ExposureMin;
```

Property value

Integer specifying minimal possible camera exposure.

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.3.5 Camera.Gain Property

Gets or sets camera gain. The gain range can be acquired using Neurotec.DeviceManager.CameraParameter.GainMin (see page 312) and Neurotec.DeviceManager.Camera.GainMax (see page 312).

C#

```
public uint Gain;
```

Property value

Integer specifying camera gain.

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.3.6 Camera.GainMax Property

Gets maximal camera gain.

C#

```
public uint GainMax;
```

Property value

Integer specifying maximal possible camera gain.

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.3.7 Camera.GainMin Property

Gets minimal camera gain.

C#

```
public uint GainMin;
```

Property value

Integer specifying minimal possible camera gain.

Remarks

This parameter is used only when automatic settings is disabled. See Neurotec.DeviceManager.Camera.ParameterAutomaticSettings (see page 307)

8.2.4.1.1.3.8 Camera.Id Property

Gets associated device identifier.

C#

```
public string Id;
```

Property value

An associated device identifier.

8.2.4.1.1.3.9 Camera.IpChannelId Property

Gets/sets Cisco IP camera streaming Channel Id (see page 312).

C#

```
public string IpChannelId;
```

Property value

String specifying camera streaming information

8.2.4.1.1.3.10 Camera.IpChannelName Property

Gets/sets Cisco IP camera streaming Channel name.

C#

```
public string IpChannelName;
```

Property value

String specifying camera streaming information

8.2.4.1.1.3.11 Camera.IpPassword Property

Gets/sets Cisco IP camera password.

C#

```
public string IpPassword;
```

Property value

String specifying camera login information

8.2.4.1.1.3.12 Camera.IpUserName Property

Gets/sets Cisco IP camera user name.

C#

```
public string IpUserName;
```

Property value

String specifying camera login information

8.2.4.1.1.3.13 Camera.IsCapturing Property

Checks camera status.

C#

```
public bool IsCapturing;
```

Property value

A boolean value indicating a camera status. If the camera is capturing the return value is true. Otherwise the return value is false.

8.2.4.1.1.3.14 Camera.MirrorHorizontal Property

Gets or sets horizontal mirroring of the frame.

C#

```
public bool MirrorHorizontal;
```

Property value

true if the frame is mirroring horizontally; otherwise, **false**.

8.2.4.1.1.3.15 Camera.MirrorVertical Property

Gets or sets vertical mirroring of the frame.

C#

```
public bool MirrorVertical;
```

Property value

true if the frame is mirroring vertically; otherwise, **false**.

8.2.4.1.1.3.16 Camera.Owner Property

Gets owner of the object.

C#

```
public new CameraMan Owner;
```

Property value

Object (owner) of CameraMan (see page 314) type.

8.2.4.1.1.3.17 Camera.VideoFormat Property

Gets or sets camera video format. Can not be set while the camera is capturing.

C#

```
public CameraVideoFormat VideoFormat;
```

Property value

Camera (see page 305) video format description.

8.2.4.1.2 CameraMan Class

Provides functionality for managing cameras.

C#

```
public sealed class CameraMan : NObject;
```

Remarks

For supported cameras see CameraMan Module Supported Cameras.

Methods

CameraMan Class

	Name	Description
	CameraMan (see page 315)	Initializes a new instance the CameraMan class.

CameraMan Classes

CameraMan Class

	Name	Description
	CameraCollection (see page 315)	Represents the collection of Camera (see page 305) objects.

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.

	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

CameraMan Class

	Name	Description
	Cameras (see page 316)	Gets CameraMan.CameraCollection (see page 315) collection.

8.2.4.1.2.1 CameraMan.CameraMan Constructor

Initializes a new instance the CameraMan class.

C#

```
public CameraMan(ISynchronizeInvoke synInvoke);
```

Parameters

Parameters	Description
ISynchronizeInvoke synInvoke	A reference to an interface that provides a way to synchronously or asynchronously execute a delegate.

8.2.4.1.2.2 CameraMan Classes

8.2.4.1.2.2.1 CameraMan.CameraCollection Class

Represents the collection of Camera (see page 305) objects.

C#

```
public sealed class CameraCollection : NObjectStaticReadOnlyCollection<Camera>;
```

CameraCollection Methods

	Name	Description
	IndexOf (see page 315)	Searches for the specified Camera (see page 305) and returns the zero-based index of the first occurrence within the entire CameraCollection.
	this (see page 316)	Gets Camera (see page 305) from collection by ID.

8.2.4.1.2.2.1.1 CameraCollection Methods

8.2.4.1.2.2.1.1.1 CameraMan.CameraCollection.IndexOf Method

Searches for the specified Camera (see page 305) and returns the zero-based index of the first occurrence within the entire CameraCollection (see page 315).

C#

```
public int IndexOf(string id);
```

Parameters

Parameters	Description
string id	string id of Camera (see page 305) to be found

Returns

The zero-based index of the first occurrence of value within the entire CameraCollection (see page 315), if found;

otherwise, -1.

8.2.4.1.2.2.1.1.2 CameraMan.CameraCollection.this Indexer

Gets Camera (see page 305) from collection by ID.

C#

```
public Camera this[string id];
```

Parameters

Parameters	Description
string id	An associated device identifier.

Returns

A Camera (see page 305) object.

8.2.4.1.2.3 CameraMan Properties

8.2.4.1.2.3.1 CameraMan.Cameras Property

Gets CameraMan.CameraCollection (see page 315) collection.

C#

```
public CameraCollection Cameras;
```

Property value

A collection of cameras.

Exceptions

Exceptions	Description
ObjectDisposedException	The exception that is thrown when an operation is performed on a disposed object.

8.2.4.1.3 CameraVideoFormat Structure

Camera (see page 305)'s video format structure.

C#

```
[Serializable]
[StructLayout(LayoutKind.Sequential)]
public struct CameraVideoFormat {
}
```

CameraVideoFormat Methods

	Name	Description
	ToString (see page 317)	Gets a formated string of video format.

CameraVideoFormat Properties

	Name	Description
	FrameHeight (see page 317)	Gets or sets frame height.
	FrameRate (see page 317)	Gets or sets frame's video rate.
	FrameWidth (see page 317)	Gets or sets frame's width.

8.2.4.1.3.1 CameraVideoFormat Methods

8.2.4.1.3.1.1 CameraVideoFormat.ToString Method

Gets a formated string of video format.

C#

```
public override string ToString();
```

Returns

A string representation of a video format.

8.2.4.1.3.2 CameraVideoFormat Properties

8.2.4.1.3.2.1 CameraVideoFormat.FrameHeight Property

Gets or sets frame height.

C#

```
public int FrameHeight;
```

8.2.4.1.3.2.2 CameraVideoFormat.FrameRate Property

Gets or sets frame's video rate.

C#

```
public float FrameRate;
```

8.2.4.1.3.2.3 CameraVideoFormat.FrameWidth Property

Gets or sets frame's width.

C#

```
public int FrameWidth;
```

8.2.5 Neurotec.Video Namespace

Provides functionality for reading video files.

Remarks

DLL: Neurotec.Video.dll. which is a wrapper of NVideo.dll. Neurotec.Video.dll. and NVideo.dll should be used together.

Module

.NET (see page 219)

Classes

	Name	Description
	NVideoReader (see page 318)	Provides funcionality for reading video file.

	NVideoWriter (see page 320)	Provides functionality for writing to a video file.

8.2.5.1 Classes

8.2.5.1.1 NVideoReader Class

Provides funcionality for reading video file.

C#

```
public sealed class NVideoReader : NObject;
```

Methods

NVideoReader Class

	Name	Description
	NVideoReader (see page 319)	Initializes a new instance of NVideoReader. Sets video file and opens it.

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.
	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

NVideoReader Class

	Name	Description
	GetFrame (see page 319)	Reads one frame from video file.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

NVideoReader Class

	Name	Description
	FrameCount (see page 319)	Gets a number of frames in the video file.
	FrameHeight (see page 319)	Gets a height of frames in the video file.
	FrameRate (see page 320)	Gets a frame rate of the video file.
	FrameWidth (see page 320)	Gets a width of frames in the video file.

8.2.5.1.1.1 NVideoReader.NVideoReader Constructor

Initializes a new instance of NVideoReader. Sets video file and opens it.

C#

```
public NVideoReader(string fileName);
```

Parameters

Parameters	Description
string fileName	Path of the video file to be opened.

Exceptions

Exceptions	Description
ArgumentNullException	Provided p fileName is NULL (see page 82).
ParameterException	Failed to open the file or set it to the NVideoReader object.

8.2.5.1.1.2 NVideoReader Methods

8.2.5.1.1.2.1 NVideoReader.GetFrame Method

Reads one frame from video file.

C#

```
public NImage GetFrame(int frameIndex);
```

Parameters

Parameters	Description
int frameIndex	Index of the frame to be read from the video file.

Returns

Image containing the video frame of specified index. When it is not needed anymore, the image must be freed using Dispose (see page 227)().

Exceptions

Exceptions	Description
ArgumentOutOfRangeException	Frame index is larget than the number of frames in the video file.

8.2.5.1.1.3 NVideoReader Properties

8.2.5.1.1.3.1 NVideoReader.FrameCount Property

Gets a number of frames in the video file.

C#

```
public int FrameCount;
```

Property value

The number of frames in the video file which was assigned to the NVideoReader (see page 318) object in constructor.

8.2.5.1.1.3.2 NVideoReader.FrameHeight Property

Gets a height of frames in the video file.

C#

```
public uint FrameHeight;
```

Property value

The height of frames in the video file which was assigned to the NVideoReader (see page 318) object in constructor.

See Also

FrameWidth (see page 320)

8.2.5.1.1.3.3 NVideoReader.FrameRate Property

Gets a frame rate of the video file.

C#

```
public double FrameRate;
```

Property value

A frame rate (number of frames presented in one second) of the video file. The video file was assigned to the NVideoReader (see page 318) object in constructor.

Exceptions

Exceptions	Description
Exception	Video (see page 317) file is not set.

8.2.5.1.1.3.4 NVideoReader.FrameWidth Property

Gets a width of frames in the video file.

C#

```
public uint FrameWidth;
```

Property value

Width of frames in the video file which was assigned to the NVideoReader (see page 318) object in constructor.

See Also

FrameHeight (see page 319)

8.2.5.1.2 NVideoWriter Class

Provides functionality for writing to a video file.

C#

```
public sealed class NVideoWriter : NObject;
```

Remarks

When using XVID encoder make sure that automatic optimisations are turned off. Otherwise NVideoWriter may crash due to a bug in XVID code.

If writing video files with XVID still crashes, then turn off all optimisations. Optimisation settings can be located in Options->Other Options...->Common starting in a popup dialog box where you choose XVID codec.

Methods

NVideoWriter Class

	Name	Description
	NVideoWriter (see page 321)	Initializes a new instance of NVideoWriter. Creates a video file.

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NObject Class

	Name	Description
S	CopyParameters (see page 231)	Copies parameters values from one NObject (see page 231) object to another.
S	Free (see page 231)	Releases memory resources used by specified object.
	GetNativeType (see page 232)	Retrieves native type of object.
	GetParameter (see page 232)	Gets value of parameter by parameter id.
	Reset (see page 232)	Resets all NObject (see page 231) parameters to default values.
	SetParameter (see page 232)	Sets value of parameter by parameter id.

NVideoWriter Class

	Name	Description
	WriteFrame (see page 321)	Appends one frame to the video file.

NObject Properties

NObject Class

	Name	Description
	Handle (see page 233)	Gets handle to unmanaged NObject (see page 231).
	Owner (see page 233)	Gets owner of the object.

8.2.5.1.2.1 NVideoWriter.NVideoWriter Constructor

Initializes a new instance of NVideoWriter. Creates a video file.

C#

```
public NVideoWriter(string fileName, int width, int height, double frameRate,
NVideoWriterOptions options);
```

Parameters

Parameters	Description
string fileName	Path of the video file to be created.
int width	Width of the video file frames in pixels.
int height	Height of the video file frames in pixels.
double frameRate	Frames per second of the resulting video.

8.2.5.1.2.2 NVideoWriter Methods

8.2.5.1.2.2.1 NVideoWriter.WriteFrame Method

Appends one frame to the video file.

C#

```
public void WriteFrame(NImage frame);
```

Parameters

Parameters	Description
NImage frame	frame to add to the video file.

8.2.6 Neurotec.Licensing Namespace

Provides functionality for getting, releasing licenses.

Module

.NET (see page 219)

Classes

	Name	Description
	NLicense (see page 322)	Provides functionality for getting, releasing licenses.
	NLicenseInfo (see page 328)	Provides information about NLicensing (see page 154) library.
	NLicensing (see page 329)	Provides methods for retrieving information about the NLicensing (see page 154) library.

8.2.6.1 Classes

8.2.6.1.1 NLicense Class

Provides functionality for getting, releasing licenses.

C#

```
public static class NLicense;
```

NLicense Methods

	Name	Description
	GetInfo (see page 322)	Retrieves license information of specified product.
	IsComponentActivated (see page 323)	Checks if component specified by name is activated.
	Obtain (see page 323)	Obtains license.
	Release (see page 327)	Releases license.

8.2.6.1.1.1 NLicense Methods

8.2.6.1.1.1.1 NLicense.GetInfo Method

Retrieves license information of specified product.

C#

```
public static NLicenseInfo GetInfo(string product);
```

Parameters

Parameters	Description
string product	String that contains name of a product to retrieve license information.

Returns

NLicenseInfo (see page 328) object that contains properties for retrieving license information.

8.2.6.1.1.1.2 NLicense.IsComponentActivated Method

Checks if component specified by name is activated.

C#

```
public static bool IsComponentActivated(string name);
```

Parameters

Parameters	Description
string name	String which contains component name.

Returns

Bool value indicating if the specified component was activated.

Remarks

These component names for `name` parameter can be used:

- Matchers.Fusion
- Matchers.Fingers
- Matchers.Faces
- Matchers.Irises
- Matchers.Palms
- Matchers.Fingers Pro
- Matchers.Faces Pro
- Matchers.Irises Pro
- Matchers.Palms Pro
- Extractors.Fingers
- Extractors.Faces Detection
- Extractors.Faces Extraction
- Extractors.Irises
- Extractors.Palms

8.2.6.1.1.1.3 Obtain Method

8.2.6.1.1.1.3.1 NLicense.Obtain Method (string, int, string)

Obtains license.

C#

```
public static bool Obtain(string address, int port, string product);
```

Parameters

Parameters	Description
string address	Server address where license manager is installed as a server. "/local" is used for this parameter when license is tied with the computer using serial number.
int port	License manager server port.
string product	String which contains license name.

Returns

Returns true if license available; otherwise, false.

Remarks

These values are available for `product` parameter (by default concurrent license is obtained):

1. **VeriFinger SDK**:
   1. SingleComputerLicense:VFExtractor or concurrent:VFExtractor
   2. SingleComputerLicense:VFMatcher

VFExtractor - VeriFinger Extractor

VFMatcher - VeriFinger Matcher

2. **VeriLook SDK**:
   1. SingleComputerLicense:VLExtractor or concurrent:VLExtractor
   2. SingleComputerLicense:VLMatcher

VLExtractor - VeriLook Extractor

VLMatcher - VeriLook Matcher

3. **VeriLook Surveillance SDK**:
   1. SingleComputerLicense:VLSurveillance
4. **MegaMatcher SDK**:
   1. SingleComputerLicense:MegaMatcherServer
   2. SingleComputerLicense:MegaMatcherClusterServer
   3. SingleComputerLicense:MegaMatcherClusterNode
   4. SingleComputerLicense:MegaMatcherClient
   5. concurrent:MegaMatcherClient
5. **VeriEye SDK**:
   1. SingleComputerLicense:VEExtractor
   2. SingleComputerLicense:VEMatcher

VEExtractor - VeriEye Extractor

VEMatcher - VeriEye Matcher

6. **BSS SDK**:
   1. SingleComputerLicense:VFBSS
   2. SingleComputerLicense:VLBSS
   3. SingleComputerLicense:VEBSS
7. **VeriFinger VAR SDK**:
   1. SingleComputerLicense:VFExtractorV
   2. SingleComputerLicense:VFMatcherV

VFExtractorV - VeriFinger Extractor

VFMatcherV - VeriFinger Matcher

8. **VeriLook VAR SDK**:
   1. SingleComputerLicense:VLExtractorV
   2. SingleComputerLicense:VLMatcherV

VLExtractorV - VeriLook Extractor

VLMatcherV - VeriLook Matcher

9. **VeriEye VAR SDK**:
   1. SingleComputerLicense:VEExtractorV
   2. SingleComputerLicense:VEMatcherV

VEExtractorV - VeriEye Extractor

VEMatcherV - VeriEye Matcher

10. **SentiSight (see page 272) SDK:**
    1. SingleComputerLicense:SentiSight (see page 272)

Note: nonconcurrent licenses (SingleComputerLicense) **can not be released** with NLicenseRelease (see page 158) function, but .NLicenseRelease (see page 158) function must be called before program exits.

If concurrent license is in use, then NLicenseRelease (see page 158) function should be called after features extraction.

Notes

Acquired licens e must be released using Release (see page 327) method

8.2.6.1.1.1.3.2 NLicense.Obtain Method (string, string, string)

Obtains license.

C#

```
public static bool Obtain(string address, string port, string product);
```

Parameters

Parameters	Description
string address	Server address where license manager is installed as a server. "/local" is used for this parameter when license is tied with the computer using serial number.
string port	License manager server port.
string product	String which contains license name.

Returns

Returns true if license available; otherwise, false.

Remarks

These values are available for `product` parameter (by default concurrent license is obtained):

1. **VeriFinger SDK**:
   1. SingleComputerLicense:VFExtractor or concurrent:VFExtractor
   2. SingleComputerLicense:VFMatcher

VFExtractor - VeriFinger Extractor

VFMatcher - VeriFinger Matcher

2. **VeriLook SDK**:
   1. SingleComputerLicense:VLExtractor or concurrent:VLExtractor
   2. SingleComputerLicense:VLMatcher

VLExtractor - VeriLook Extractor

VLMatcher - VeriLook Matcher

3. **VeriLook Surveillance SDK**:

1. SingleComputerLicense:VLSurveillance

4. **MegaMatcher SDK**:
   1. SingleComputerLicense:MegaMatcherServer
   2. SingleComputerLicense:MegaMatcherClusterServer
   3. SingleComputerLicense:MegaMatcherClusterNode
   4. SingleComputerLicense:MegaMatcherClient
   5. concurrent:MegaMatcherClient
5. **VeriEye SDK**:
   1. SingleComputerLicense:VEExtractor
   2. SingleComputerLicense:VEMatcher

   VEExtractor - VeriEye Extractor

   VEMatcher - VeriEye Matcher
6. **BSS SDK**:
   1. SingleComputerLicense:VFBSS
   2. SingleComputerLicense:VLBSS
   3. SingleComputerLicense:VEBSS
7. **VeriFinger VAR SDK**:
   1. SingleComputerLicense:VFExtractorV
   2. SingleComputerLicense:VFMatcherV

   VFExtractorV - VeriFinger Extractor

   VFMatcherV - VeriFinger Matcher
8. **VeriLook VAR SDK**:
   1. SingleComputerLicense:VLExtractorV
   2. SingleComputerLicense:VLMatcherV

   VLExtractorV - VeriLook Extractor

   VLMatcherV - VeriLook Matcher
9. **VeriEye VAR SDK**:
   1. SingleComputerLicense:VEExtractorV
   2. SingleComputerLicense:VEMatcherV

   VEExtractorV - VeriEye Extractor

   VEMatcherV - VeriEye Matcher
10. **SentiSight (see page 272) SDK:**
    1. SingleComputerLicense:SentiSight (see page 272)

Note: nonconcurrent licenses (SingleComputerLicense) **can not be released** with NLicenseRelease (see page 158) function, but .NLicenseRelease (see page 158) function must be called before program exits.

If concurrent license is in use, then NLicenseRelease (see page 158) function should be called after features extraction.

Notes

Acquired license must be released using Release (see page 327) method.

8.2.6.1.1.1.4 NLicense.Release Method

Releases license.

C#

```
public static void Release(string product);
```

Parameters

Parameters	Description
string product	Name of a product to release its license.

Returns

Returns true if license available; otherwise, false.

Remarks

1. **VeriFinger SDK**:
   1. SingleComputerLicense:VFExtractor or concurrent:VFExtractor
   2. SingleComputerLicense:VFMatcher

   VFExtractor - VeriFinger Extractor

   VFMatcher - VeriFinger Matcher
2. **VeriLook SDK**:
   1. SingleComputerLicense:VLExtractor or concurrent:VLExtractor
   2. SingleComputerLicense:VLMatcher

   VLExtractor - VeriLook Extractor

   VLMatcher - VeriLook Matcher
3. **VeriLook Surveillance SDK**:
   1. SingleComputerLicense:VLSurveillance
4. **MegaMatcher SDK**:
   1. SingleComputerLicense:MegaMatcherServer
   2. SingleComputerLicense:MegaMatcherClusterServer
   3. SingleComputerLicense:MegaMatcherClusterNode
   4. SingleComputerLicense:MegaMatcherClient
   5. concurrent:MegaMatcherClient
5. **VeriEye SDK**:
   1. SingleComputerLicense:VEExtractor
   2. SingleComputerLicense:VEMatcher

   VEExtractor - VeriEye Extractor

   VEMatcher - VeriEye Matcher
6. **BSS SDK**:
   1. SingleComputerLicense:VFBSS
   2. SingleComputerLicense:VLBSS
   3. SingleComputerLicense:VEBSS
7. **VeriFinger VAR SDK**:
   1. SingleComputerLicense:VFExtractorV

2. SingleComputerLicense:VFMatcherV

VFExtractorV - VeriFinger Extractor

VFMatcherV - VeriFinger Matcher

8. **VeriLook VAR SDK**:
   1. SingleComputerLicense:VLExtractorV
   2. SingleComputerLicense:VLMatcherV

VLExtractorV - VeriLook Extractor

VLMatcherV - VeriLook Matcher

9. **VeriEye VAR SDK**:
   1. SingleComputerLicense:VEExtractorV
   2. SingleComputerLicense:VEMatcherV

VEExtractorV - VeriEye Extractor

VEMatcherV - VeriEye Matcher

10. **SentiSight (see page 272) SDK:**
    1. SingleComputerLicense:SentiSight (see page 272)

Note: nonconcurrent licenses (SingleComputerLicense) can not be released with Release method, but .Release function must be called before program exits.

If concurrent license is in use, then Release method should be called after features extraction.

8.2.6.1.2 NLicenseInfo Class

Provides information about NLicensing (see page 154) library.

C#

```
[StructLayout(LayoutKind.Sequential)]
public sealed class NLicenseInfo;
```

NLicenseInfo Properties

	Name	Description
	DistributorId (see page 328)	Gets distributor id.
	IsObtained (see page 328)	Gets license status.
	SerialNumber (see page 329)	Gets serial number of a license.

8.2.6.1.2.1 NLicenseInfo Properties

8.2.6.1.2.1.1 NLicenseInfo.DistributorId Property

Gets distributor id.

C#

```
public int DistributorId;
```

Property value

Distributor id.

8.2.6.1.2.1.2 NLicenseInfo.IsObtained Property

Gets license status.

C#

```
public bool IsObtained;
```

Property value

True if license is obtained; otherwise, false.

8.2.6.1.2.1.3 NLicenseInfo.SerialNumber Property

Gets serial number of a license.

C#

```
public int SerialNumber;
```

Returns

Serial number.

8.2.6.1.3 NLicensing Class

Provides methods for retrieving information about the NLicensing (see page 154) library.

C#

```
public static class NLicensing;
```

NLicensing Fields

	Name	Description
	DllName (see page 329)	Name of DLL containing unmanaged part of this class.

NLicensing Methods

	Name	Description
S	GetInfo (see page 329)	Gets information about the library.

8.2.6.1.3.1 NLicensing Fields

8.2.6.1.3.1.1 NLicensing.DllName Field

Name of DLL containing unmanaged part of this class.

C#

```
public const string DllName = "NLicensing";
```

8.2.6.1.3.2 NLicensing Methods

8.2.6.1.3.2.1 NLicensing.GetInfo Method

Gets information about the library.

C#

```
public static NLibraryInfo GetInfo();
```

Returns

Returns NLibraryInfo (see page 229) structure with library information.

8.2.7 Neurotec.IO Namespace

Classes under this namespace provides infrastructure for Neurotechnology components.

Module

.NET (see page 219)

Classes

	Name	Description
	NBuffer (see page 330)	Provides methods for manipulating arrays of primitive types.
	NMemoryStream (see page 332)	Provides access to unmanaged blocks of memory from managed code.
	NStream (see page 334)	This class supports internal Neurotechnology libraries infrastructure and should not be used directly in your code.

8.2.7.1 Classes

8.2.7.1.1 NBuffer Class

Provides methods for manipulating arrays of primitive types.

C#

```
public class NBuffer : NDisposable;
```

Methods

NBuffer Class

	Name	Description
	NBuffer (see page 331)	Initializes a new instance of the NBuffer class.

NDisposable Methods

	Name	Description
	Dispose (see page 227)	Performs tasks associated with freeing, releasing, or resetting unmanaged resources.

NBuffer Class

	Name	Description
	ToArray (see page 331)	Copies the elements of the NBuffer to a new byte array.
	WriteTo (see page 331)	Writes the contents of NBuffer to memory stream.

NBuffer Properties

NBuffer Class

	Name	Description
	Length (see page 331)	Gets the length of NBuffer.
	LongLength (see page 331)	Gets a 64-bit integer that represents the length of NBuffer.
	Ptr (see page 332)	Gets a representation of NBuffer pointer.
	UIntPtrLength (see page 332)	Gets UIntPtr that represents the length of NBuffer.

8.2.7.1.1.1 NBuffer.NBuffer Constructor

Initializes a new instance of the NBuffer class.

C#

```
public NBuffer(IntPtr ptr, int length);
```

Parameters

Parameters	Description
IntPtr ptr	A pointer to buffer.
int length	Length (see page 331) of a buffer.

8.2.7.1.1.2 NBuffer Methods

8.2.7.1.1.2.1 NBuffer.ToArray Method

Copies the elements of the NBuffer (see page 330) to a new byte array.

C#

```
public byte[] ToArray();
```

Returns

A byte array containing copies of the elements of the NBuffer (see page 330).

8.2.7.1.1.2.2 NBuffer.WriteTo Method

Writes the contents of NBuffer (see page 330) to memory stream.

C#

```
public void WriteTo(Stream stream);
```

Parameters

Parameters	Description
Stream stream	The stream to write the contents of NBuffer (see page 330) to.

8.2.7.1.1.3 NBuffer Properties

8.2.7.1.1.3.1 NBuffer.Length Property

Gets the length of NBuffer (see page 330).

C#

```
public int Length;
```

Property value

The length of NBuffer (see page 330).

8.2.7.1.1.3.2 NBuffer.LongLength Property

Gets a 64-bit integer that represents the length of NBuffer (see page 330).

C#

```
public long LongLength;
```

8.2.7.1.1.3.3 NBuffer.Ptr Property

Gets a representation of NBuffer (see page 330) pointer.

C#

```
public IntPtr Ptr;
```

8.2.7.1.1.3.4 NBuffer.UIntPtrLength Property

Gets UIntPtr that represents the length of NBuffer (see page 330).

C#

```
public UIntPtr UIntPtrLength;
```

8.2.7.1.2 NMemoryStream Class

Provides access to unmanaged blocks of memory from managed code.

C#

```
public sealed class NMemoryStream : NStream;
```

Methods

NMemoryStream Class

	Name	Description
	NMemoryStream (see page 333)	Initializes a new, empty instance of the NMemoryStream class.

NStream Methods

	Name	Description
	Close (see page 334)	Closes current stream.
	Flush (see page 335)	Clears all buffers for this stream and causes any buffered data to be written to the underlying device.
S	FromHandle (see page 335)	Gets NStream (see page 334) from it's handle.
S	FromStream (see page 335)	Creates new NStream (see page 334) from Stream object.
	Read (see page 336)	Reads a sequence of bytes from the current stream.
	ReadByte (see page 336)	Reads a byte from the stream.
	Seek (see page 337)	Sets the position within the current stream.
	SetLength (see page 337)	Sets the length of the current stream.
	Write (see page 337)	Writes a sequence of bytes to the current stream.
	WriteByte (see page 338)	Writes a byte to the current position in the stream.

NMemoryStream Class

	Name	Description
	ToArray (see page 333)	Saves stream to byte array.
	WriteTo (see page 333)	Writes NMemoryStream to Stream.

NStream Properties

	Name	Description
	CanRead (see page 338)	Gets a value indicating whether the current stream supports reading.
	CanSeek (see page 338)	Gets a value indicating whether the current stream supports seeking.
	CanWrite (see page 338)	Gets a value indicating whether the current stream supports writing.
	Handle (see page 338)	Gets handle to unmanaged NStream (see page 334) object.
	Length (see page 339)	Gets size of a stream in bytes.

	Position (see page 339)	Gets number of bytes from the start of the data to the current position in a stream.

NMemoryStream Class

	Name	Description
	Capacity (see page 334)	Gets the stream size of memory assigned to a stream (capacity).

8.2.7.1.2.1 NMemoryStream Constructor

8.2.7.1.2.1.1 NMemoryStream.NMemoryStream Constructor ()

Initializes a new, empty instance of the NMemoryStream class.

C#

```
public NMemoryStream();
```

8.2.7.1.2.1.2 NMemoryStream.NMemoryStream Constructor (long)

Initializes a new instance of the NMemoryStream class using the specified memory length.

C#

```
public NMemoryStream(long capacity);
```

Parameters

Parameters	Description
long capacity	The length of the memory to use.

8.2.7.1.2.1.3 NMemoryStream.NMemoryStream Constructor (long, long)

Initializes a new instance of the NMemoryStream class using the specified memory length, total amount of memory.

C#

```
public NMemoryStream(long capacity, long maxCapacity);
```

Parameters

Parameters	Description
long capacity	The length of the memory to use.
long maxCapacity	The max amount of memory assigned to the stream.

8.2.7.1.2.2 NMemoryStream Methods

8.2.7.1.2.2.1 NMemoryStream.ToArray Method

Saves stream to byte array.

C#

```
public byte[] ToArray();
```

Returns

Byte array.

8.2.7.1.2.2.2 NMemoryStream.WriteTo Method

Writes NMemoryStream (see page 332) to Stream.

C#

```
public void WriteTo(Stream stream);
```

Parameters

Parameters	Description
Stream stream	Stream object.

8.2.7.1.2.3 NMemoryStream Properties

8.2.7.1.2.3.1 NMemoryStream.Capacity Property

Gets the stream size of memory assigned to a stream (capacity).

C#

```
public long Capacity;
```

8.2.7.1.3 NStream Class

This class supports internal Neurotechnology libraries infrastructure and should not be used directly in your code.

C#

```
public class NStream : Stream;
```

NStream Methods

	Name	Description
	Close (see page 334)	Closes current stream.
	Flush (see page 335)	Clears all buffers for this stream and causes any buffered data to be written to the underlying device.
S	FromHandle (see page 335)	Gets NStream from it's handle.
S	FromStream (see page 335)	Creates new NStream from Stream object.
	Read (see page 336)	Reads a sequence of bytes from the current stream.
	ReadByte (see page 336)	Reads a byte from the stream.
	Seek (see page 337)	Sets the position within the current stream.
	SetLength (see page 337)	Sets the length of the current stream.
	Write (see page 337)	Writes a sequence of bytes to the current stream.
	WriteByte (see page 338)	Writes a byte to the current position in the stream.

NStream Properties

	Name	Description
	CanRead (see page 338)	Gets a value indicating whether the current stream supports reading.
	CanSeek (see page 338)	Gets a value indicating whether the current stream supports seeking.
	CanWrite (see page 338)	Gets a value indicating whether the current stream supports writing.
	Handle (see page 338)	Gets handle to unmanaged NStream object.
	Length (see page 339)	Gets size of a stream in bytes.
	Position (see page 339)	Gets number of bytes from the start of the data to the current position in a stream.

8.2.7.1.3.1 NStream Methods

8.2.7.1.3.1.1 NStream.Close Method

Closes current stream.

C#

```
public override void Close();
```

8.2.7.1.3.1.2 NStream.Flush Method

Clears all buffers for this stream and causes any buffered data to be written to the underlying device.

C#

```
public override void Flush();
```

8.2.7.1.3.1.3 FromHandle Method

8.2.7.1.3.1.3.1 NStream.FromHandle Method (IntPtr)

Gets NStream (see page 334) from it's handle.

C#

```
public static NStream FromHandle(IntPtr handle);
```

Parameters

Parameters	Description
IntPtr handle	Handle (see page 338) to unmanaged NStream (see page 334) object.

Returns

NStream (see page 334) object.

8.2.7.1.3.1.3.2 NStream.FromHandle Method (IntPtr, bool)

Gets NStream (see page 334) from it's handle.

C#

```
public static NStream FromHandle(IntPtr handle, bool ownsHandle);
```

Parameters

Parameters	Description
IntPtr handle	Handle (see page 338) to unmanaged NStream (see page 334) object.
bool ownsHandle	If set to true, on destruction of the NStream (see page 334) the NStream (see page 334) pointed to by the handle will also be destroyed.

Returns

NStream (see page 334) object.

8.2.7.1.3.1.4 NStream.FromStream Method

Creates new NStream (see page 334) from Stream object.

C#

```
public static NStream FromStream(Stream stream, bool ownsStream);
```

Parameters

Parameters	Description
Stream stream	Memory stream.
bool ownsStream	If set to true, on destruction of the NStream (see page 334) the original Stream will also be destroyed.

Returns

NStream (see page 334) object.

8.2.7.1.3.1.5 Read Method

8.2.7.1.3.1.5.1 NStream.Read Method (IntPtr, int)

Reads a sequence of bytes from the current stream.

C#

```
public int Read(IntPtr pBuffer, int count);
```

Parameters

Parameters	Description
IntPtr pBuffer	Memory buffer.
int count	Number of bytes to be read.

Returns

Integer type variable with number of bytes successfully read.

8.2.7.1.3.1.5.2 NStream.Read Method (IntPtr, long)

Reads a sequence of bytes from the current stream.

C#

```
public long Read(IntPtr pBuffer, long count);
```

Parameters

Parameters	Description
IntPtr pBuffer	Memory buffer.
long count	Number of bytes to be read.

Returns

Long type variable with number of bytes successfully read.

8.2.7.1.3.1.5.3 NStream.Read Method (byte[], int, int)

Reads a sequence of bytes from the current stream.

C#

```
public override int Read(byte[] buffer, int offset, int count);
```

Parameters

Parameters	Description
byte[] buffer	An array of bytes. When this method returns, the buffer contains the specified byte array with the values between offset and (offset + count - 1) replaced by the bytes read from the current source.
int offset	The zero-based byte offset in buffer at which to begin storing the data read from the current stream.
int count	Number of bytes to be read from the current stream.

Returns

Integer type variable with number of bytes successfully read.

8.2.7.1.3.1.6 NStream.ReadByte Method

Reads a byte from the stream.

C#

```
public override int ReadByte();
```

Returns

Value of byte at current position in a stream.

8.2.7.1.3.1.7 NStream.Seek Method

Sets the position within the current stream.

C#

```
public override long Seek(long offset, SeekOrigin origin);
```

Parameters

Parameters	Description
long offset	A byte offset relative to the origin parameter.
SeekOrigin origin	A value indicating the reference point used to obtain the new position.

Returns

The new position within the current stream.

8.2.7.1.3.1.8 NStream.SetLength Method

Sets the length of the current stream.

C#

```
public override void SetLength(long value);
```

Parameters

Parameters	Description
long value	New desired length of the current stream in bytes.

8.2.7.1.3.1.9 Write Method

8.2.7.1.3.1.9.1 NStream.Write Method (IntPtr, int)

Writes a sequence of bytes to the current stream.

C#

```
public void Write(IntPtr pBuffer, int count);
```

Parameters

Parameters	Description
IntPtr pBuffer	Memory buffer.
int count	Number of bytes to be written.

8.2.7.1.3.1.9.2 NStream.Write Method (IntPtr, long)

Writes a sequence of bytes to the current stream.

C#

```
public void Write(IntPtr pBuffer, long count);
```

Parameters

Parameters	Description
IntPtr pBuffer	Memory buffer.
long count	Number of bytes to be written.

8.2.7.1.3.1.9.3 NStream.Write Method (byte[], int, int)

Writes a sequence of bytes to the current stream.

C#

```
public override void Write(byte[] buffer, int offset, int count);
```

Parameters

Parameters	Description
byte[] buffer	An array of bytes.
int offset	The zero-based byte offset in buffer at which to begin copying bytes to the current stream.
int count	Number of bytes to be written to the current stream.

8.2.7.1.3.1.10 NStream.WriteByte Method

Writes a byte to the current position in the stream.

C#

```
public override void WriteByte(byte value);
```

Parameters

Parameters	Description
byte value	The byte to write to the stream.

8.2.7.1.3.2 NStream Properties

8.2.7.1.3.2.1 NStream.CanRead Property

Gets a value indicating whether the current stream supports reading.

C#

```
public override bool CanRead;
```

8.2.7.1.3.2.2 NStream.CanSeek Property

Gets a value indicating whether the current stream supports seeking.

C#

```
public override bool CanSeek;
```

8.2.7.1.3.2.3 NStream.CanWrite Property

Gets a value indicating whether the current stream supports writing.

C#

```
public override bool CanWrite;
```

8.2.7.1.3.2.4 NStream.Handle Property

Gets handle to unmanaged NStream (see page 334) object.

C#

```
public IntPtr Handle;
```

8.2.7.1.3.2.5 NStream.Length Property

Gets size of a stream in bytes.

C#

```
public override long Length;
```

8.2.7.1.3.2.6 NStream.Position Property

Gets number of bytes from the start of the data to the current position in a stream.

C#

```
public override long Position;
```

8.3 Change Log

In this section change log of the Neurotechnology API components is provided. These abbreviations are used:

- **FIX** - Fixed
- **CHN** - Changed
- **UPD** - Updated
- **ADD** - Added
- **REM** - Removed
- **MER** - Merged

Version 2.1.0.1

- **UPD**: NImages (see page 98) library to 2.6.1.0 (see page 342)
- **UPD**: NVideo (see page 161) library to 2.1.0.1 (see page 345)

8.3.1 C Reference

8.3.1.1 NCore Library

Version 3.1.1.1

- **UPD**: N_E_COM (see page 28), N_E_SYS (see page 31) and N_E_WIN32 (see page 31) now are inner errors if they are cause of another error.

Version 3.1.1.0

- **CHN**: Move NFileAccess (see page 61) and NByteOrder (see page 61) enum definitions from NTypes.h (see page 83) to NStream.h.

Version 3.1.0.4

- **FIX**: Minor fixes.

Version 3.1.0.3

- **FIX**: Minor fixes.

Version 3.1.0.2

- **FIX**: Minor fixes.

Version 3.1.0.1

- **FIX**: NTypeSupportsParameters function name.

Version 3.1.0.0

- **ADD**: HNObject, a handle to base object for other objects and operations (free and parameters) on it.
- **ADD**: HNType, a handle to type information for an object and operations on it.
- **ADD**: New, more specific error codes.
- **CHN**: N_CALLBACK is now pointer-to-function.
- **REM**: Obsolete defines, functions and members.
- **UPD**: Refactoring and update of headers.

Version 3.0.0.3

- **FIX**: Crash when file open fails on *nix.

Version 3.0.0.2

- **UPD**: String parameter handling in parameter framework.

Version 3.0.0.1

- **FIX**: Minor fixes.

Version 3.0.0.0

- **UPD**: Unicode support made cross-platform.
- **ADD**: System error (errno) intergration to NErrors.
- **ADD**: SSSE4, SSE4.1, SSE4.2, SSE4A, LZCNT and POPCNT detection support to NProcessorInfo.
- **UPD**: NLibraryInfo structure.
- **ADD**: NCoreOnXxx functions for NCore (see page 24) initialization/cleanup when it is a static library (does nothing when it is a dynamic library).

Version 2.4.4.0

- **UPD**: Error framework made cross-platform.

Version 2.4.3.0

- **ADD**: Aligned memory management functions.

Version 2.4.2.1

- **FIX**: Minor fixes.

Version 2.4.2.0

- **ADD**: NPointF (see page 34), NPointD (see page 34), NSizeF (see page 36), NSizeD (see page 36), NRectF (see page 35) and NRectD (see page 35) types.
- **ADD**: NRange type.
- **UPD**: Internal optimizations.

Version 2.4.1.1

- **FIX**: Minor fixes.

Version 2.4.1.0

- **ADD**: NLibraryInfo module.
- **ADD**: NCoreGetInfo (see page 25) 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.
- **ADD**: Stream integration with .NET.
- **UPD**: Exception integration with .NET.

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 (see page 64) structure.

Version 1.0.0.1

- **FIX**: Minor fixes in headers.

Version 1.0.0.0

- Initial release.

8.3.1.2 NDeviceManager Library

Version 3.1.0.1

- **FIX**: Minor fixes.

Version 3.1.0.0

- **CHN**: IrisCameraCallback, IrisCameraSetCallback renamed accordingly to IrisCameraIrisesAcquiredCallback, IrisCameraSetIrisesAcquiredCallback.
- **REM**: Obsolete defines, functions and members.
- **UPD**: Now uses NImages (see page 98) library version 2.6.
- **UPD**: Now uses NCore (see page 24) library version 3.1.

Version 3.0.1.1

- **FIX**: Minor fixes for Camera parameters.

Version 3.0.1.0

- **ADD**: New Camera parameters to support IP cameras.

Version 3.0.0.1

- **UPD**: Internal updates.

Version 3.0.0.0

- New version of NDeviceManager (see page 85) released.
- **MER**: Previous libraries CameraMan, FPScannerMan, IrisCameraMan were merged into one library.

8.3.1.3 NImages Library

Version 2.6.1.3

- **FIX**: NMonochromeImageSetPixel (see page 131) inverting value.

Version 2.6.1.2

- **FIX**: WSQ and IHead multi-thread issues.

Version 2.6.1.1

- **FIX**: Reading of some JP2 files with alpha channel.

Version 2.6.1.0

- **UPD**: Support 8-bit RGB-palletized JP2 files.

Version 2.6.0.1

- **FIX**: Cross-platform usage issues.

Version 2.6.0.0

- **ADD**: HNMonochromeImage, HNGrayscaleImage and HNRgbImage handle types to clearly indicate which NImage subclass is used.
- **REM**: Obsolete defines, functions and members.
- **UPD**: Now uses NCore (see page 24) library version 3.1.

Version 2.5.0.4

- **FIX**: NImage diagonal flip produces artifact on some images.
- **FIX**: NImage rotate does not "rotate" horizontal and vertical resolutions.

Version 2.5.0.3

- **FIX**: BmpLoadImageFromHBitmap (see page 140) when HBITMAP is not DIBSECTION.

Version 2.5.0.2

- **FIX**: Save to memory, stream or file without extension in JPEG NImageFormat fails.

Version 2.5.0.1

- **FIX**: Empty file created during save if image or license check error occurs.
- **FIX**: Format-neutral image save with .jpl extension should save in Lossless JPEG format.
- **FIX**: Add missing Lossless JPEG part activation info.
- **FIX**: NImagesGetGrayscaleColorWrapperEx (see page 126) function signature.

Version 2.5.0.0

- **ADD**: Lossless JPEG format support.
- **UPD**: Pro and standard versions of the NImages (see page 98) are now the same library.
- **UPD**: Unicode support made cross-platform.

Version 2.4.0.2

- **FIX**: Crash during CMYK JPEG files reading.
- **FIX**: Memory leak during save in JPEG 2000 format.

Version 2.4.0.1

- **UPD**: Internal updates.

Version 2.4.0.0

- **ADD**: PNG format support.
- **ADD**: JPEG2000 format support.
- **CHN**: RGB to grayscale conversion now uses correct formula.

Version 2.3.0.0

- **ADD**: Image rotation, flipping and cropping functionality.

Version 2.2.0.2

- **ADD**: NImagesGetInfo (see page 127) 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.
- **FIX**: Memory leak when reading WSQ files.

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

- **UPD**: Updated to use Microsoft Visual C++ Runtime Library 8.0.
- **CHN**: Renamed to NImages (see page 98) to be consistent with other components.

Version 2.0.0.2

- **FIX**: Some BMP files reading.

Version 2.0.0.1

- CHN: Minor internal changes.

Version 2.0.0.0

- **CHN**: Some interface changes.
- **ADD**: Support for monochrome and RGB images.
- **ADD**: Support for files with multiple images.
- **ADD**: BMP, TIFF (load-only) and NIST IHead formats support.
- **FIX**: WSQ reading errors with some files.

Version 1.0.0.2

- **FIX**: Minor fixes in header files.

Version 1.0.0.1

- **FIX**: Minor fixes.

Version 1.0.0.0

- Initial release.

8.3.1.4 NLicensing Library

Version 3.1.0.0

- **ADD**: NLicenseIsComponentActivated (see page 156) function.
- **UPD**: Now uses NCore (see page 24) library version 3.1.

Version 3.0.0.0

- New version of the NLicensing (see page 154) library released.

8.3.1.5 NVideo Library

Version 2.1.0.2

- **FIX**: support for certains codecs that use Decompressor filters.

Version 2.1.0.1

- **FIX**: Reading frames from multiple files.

Version 2.1.0.0

- **REM**: Obsolete defines, functions and members.
- **UPD**: Now uses NImages (see page 98) library version 2.6.
- **UPD**: Now uses NCore (see page 24) library version 3.1.

Version 2.0.0.0

- **ADD**: NVideoWriter module.

Version 1.0.0.0

- Initial release of NVideo (see page 161) library.

8.3.1.6 SentiSight Library

Version 2.1.0.0

- **FIX**: Memory leak in SERecSetTrackingImageSize (see page 195) function is fixed.
- **FIX**: Fixed wrong behaviour of SEShapeGetCenter (see page 206) returning incorrect shape center value in recognition details and some other cases.
- **FIX**: Fixed SELrnGeneralizeModel (see page 176) crashes on models learned from unsuitable objects.
- **FIX**: Fixed Low recognition speed crashing on some big images (Rewritten to use much less memory). The order of returned recognition details can be different from previous version.
- **CHN**: Structure of the header files is changed.
- **REM**: SERotRectD structure is removed.
- **UPD**: SEStatus (see page 215) enum is added.
- **CHN**: SEModelUpdateStatus (see page 218) is deprecated SEStatus (see page 215) must be used.
- **UPD**: SEP_REC_THRESHOLD (see page 218) parameter is added.
- **UPD**: SEP_LRN_GENERALIZATION_THRESHOLD (see page 217) parameter is added.
- **CHN**: HSentiSightEngine (see page 218) is deprecated HSEEngine must be used.
- **CHN**: SEShapeIsValid (see page 209) function now fully checks shape for absence of self-intersection.

- **UPD**: Shape points manipulation Ex functions are added which always perform operations on points. Use SEShapeIsValid (see page 209) to check validity of the shape.
- **CHN**: Old shape points manipulation functions are deprecated shape points manipulation Ex functions must be used.
- **UPD**: SESepGetObjectModelSize (see page 197) is added.
- **CHN**: SESepGetModelSize is deprecated SESepGetObjectModelSize (see page 197) must be used.
- **UPD**: SESepSaveObjectModelToMemory (see page 201) is added.
- **CHN**: SESepSaveModelToMemory (see page 201) is deprecated SESepSaveObjectModelToMemory (see page 201) must be used.
- **UPD**: SESepLoadObjectModelFromMemory (see page 199) is added.
- **CHN**: SESepLoadModelFromMemory is deprecated SESepLoadObjectModelFromMemory (see page 199) must be used.
- **UPD**: SEModelSaveToMemoryEx (see page 181) is added.
- **CHN**: SEModelSaveToMemory (see page 181) is deprecated SEModelSaveToMemoryEx (see page 181) must be used.
- **UPD**: SECreateModel (see page 173) is added.
- **CHN**: SEModelCreate is deprecated SECreateModel (see page 173) must be used.
- **UPD**: SEModelClone (see page 178) is added.
- **UPD**: SELrnAddToModel (see page 174) now reports SEStatus (see page 215).
- **UPD**: SELrnAddToModel (see page 174) and SELrnAddToModelEx (see page 175) reports sesNothingToLearn if model is not updated because image is not suitable for learning.
- **UPD**: SELrnGeneralizeModelEx (see page 176) is added. The function status can be sesSucceeded or sesModelIsEmpty if input model is empty. Use SEP_LRN_GENERALIZATION_THRESHOLD (see page 217) parameter to set / get value of generalization threshold.
- **CHN**: SELrnGeneralizeModel (see page 176) is deprecated SELrnGeneralizeModelEx (see page 176) must be used.
- **UPD**: SERecGetModelIds (see page 189) is added.
- **CHN**: SERecGetModelCount (see page 189) is deprecated SERecGetModelIds (see page 189) must be used.
- **UPD**: SERecAddModelEx (see page 183) is added. The function status can be sesSucceeded or sesModelIsEmpty if input model is empty.
- **CHN**: SERecAddModel (see page 182) is deprecated SERecAddModelEx (see page 183) must be used.
- **UPD**: SERecRecognizeImageEx (see page 193) is added. Use SEP_REC_THRESHOLD (see page 218) parameter to set / get value of recognition threshold.
- **CHN**: SERecRecognizeImage (see page 191) is deprecated SERecRecognizeImageEx (see page 193) must be used.
- **UPD**: SERecDetailsIsTracked (see page 174) is added.
- **CHN**: SERecDetailsGetTracked is deprecated SERecDetailsIsTracked (see page 174) must be used.
- **REM**: Obsolete defines, functions and members.
- **UPD**: Now uses NImages (see page 98) library version 2.6.
- **UPD**: Now uses NCore (see page 24) library version 3.1.

Version 2.0.0.0 Version 1.1.0.0 Version 1.0.0.0

- Initial release of SentiSight library.

8.3.2 .NET API Reference

8.3.2.1 Neurotec Library

Version 3.1.1.3

- **UPD**: More symmetric exception integration.

Version 3.1.1.2

- **FIX**: NLibraryInfo portability issues.

Version 3.1.1.1

- **FIX**: Namespaces for collection classes.

Version 3.1.1.0

- **FIX**: Namespaces for some objects.

Version 3.1.0.4

- **FIX**: Marshalling of enumeration arrays in collection classes.

Version 3.1.0.3

- **FIX**: Hide some unused members of NCollection.

Version 3.1.0.2

- **FIX**: Minor fixes.

Version 3.1.0.1

- **UPD**: Updated to reflect changes in unmanaged code.

Version 3.1.0.0

- **ADD**: HNObject operations to NObject (see page 45) class (parameters).
- **ADD**: HNType, a handle to type information for an object and operations on it.
- **ADD**: New, more specific error codes and exception classes.
- **ADD**: NReadOnlyCollection and NCollection classes and their subclasses as base classes for other NObject (see page 45) internal collections.
- **REM**: Parameter-specific types (now NObject (see page 45) has that functionality).
- **REM**: Obsolete members and types.

Version 3.0.0.1

- **FIX**: Stream wrapper crash sometimes.

Version 3.0.0.0

- **ADD**: System error (errno) intergration for Neurotec (see page 219) exceptions.
- **ADD**: SSSE4, SSE4.1, SSE4.2, SSE4A, LZCNT and POPCNT detection support to NProcessorInfo.
- **UPD**: NLibraryInfo structure.

Version 2.4.5.1

- **FIX**: Minor fixes.

Version 2.4.5.0

- **ADD**: PointD, SizeD, RectangleD types.
- **ADD**: NRange type.

Version 2.4.4.0

- **ADD**: NMemoryStream class.
- **ADD**: NBuffer.Wrap method.

Version 2.4.3.1

- **ADD**: Constants in NResult (see page 68) for defined Neurotec (see page 219) functions return codes.

Version 2.4.2.1

- **ADD**: NBuffer class.
- **ADD**: Extended unmanaged memory management.

Version 2.4.1.1

- **FIX**: Cross-AppDomain usage issues.

Version 2.4.1.0

- **ADD**: NLibraryInfo class.
- **ADD**: NCore.GetInfo method.

Version 2.4.0.0

- **UPD**: Reflects changes in unmanaged code.

Version 2.3.1.0

- **ADD**: NProcessorInfo class.

Version 2.3.0.0

- **ADD**: Stream integration with unmanaged code.
- **UPD**: Exception integration with unmanaged code.

Version 2.2.2.0

- **ADD**: More robust unmanaged error handling on Windows.

Version 2.2.1.0

- **ADD**: Classes for internal architecture support.

Version 2.2.0.0

- **UPD**: Updated to support changes in unmanaged code.

Version 2.1.0.0

- **REM**: LicenseManagerException class.
- **CHN**: Now uses Microsoft .NET Framework 2.0.

Version 2.0.1.2

- **FIX**: Minor fixes in parameters framework.

Version 2.0.1.1

- **FIX**: Minor fixes.
- **UPD**: Minor updates in structures.

Version 2.0.1.0

- **ADD**: NParameters class 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 (see page 64) structure.

Version 1.0.0.1

- **FIX**: All error codes are mapped to appropriate exceptions.

Version 1.0.0.0

- Initial release.

8.3.2.2 Neurotec.DeviceManager

Version 3.1.0.2

- **UPD**: Minor updates because of Neurotec (see page 219) changes.

Version 3.1.0.1

- **FIX**: Added FPScanners do not trigger events.

Version 3.1.0.0

- **CHN**: IrisCameraEventArgs, IrisCameraEventHandler renamed accordingly to IrisCameraIrisesAcquiredEventArgs, IrisCameraIrisesAcquiredEventHandler.
- **CHN**: IrisCameraMan.IrisCameras deprecated in favor of new IrisCameraMan.Cameras.
- **REM**: Obsolete members and types.
- **UPD**: Now uses Neurotec.Images (see page 241) library version 2.6.
- **UPD**: Now uses Neurotec (see page 219) library version 3.1.

Version 3.0.1.1

- **FIX**: Camera properties type.

Version 3.0.1.0

- **ADD**: New Camera properties to support IP cameras.

Version 3.0.0.0

- New version of Neurotec.DeviceManager (see page 305) released.
- **MER**: Previous libraries CameraMan, FPScannerMan, IrisCameraMan were merged into one library.

8.3.2.3 Neurotec.Images

Version 2.6.0.4

- **UPD**: Minor updates because of Neurotec (see page 219) changes.

Version 2.6.0.3

- **FIX**: NImage.GetWrapper and NImage.FromData argument checks.

Version 2.6.0.2

- **UPD**: Minor updates because of Neurotec (see page 219) changes.

Version 2.6.0.1

- **FIX**: GIF open error.

Version 2.6.0.0

- **REM**: Obsolete members and types.
- **UPD**: Now uses Neurotec (see page 219) library version 3.1.

Version 2.5.0.1

- **FIX**: InvalidCastException during GIF file reading.

Version 2.5.0.0

- **ADD**: Lossless JPEG format support.

Version 2.4.1.2

- **FIX**: Minor fixes.

Version 2.4.1.1

- **FIX**: NImageFile disposal errors.

Version 2.4.1.0

- **CHN**: Methods that return System.Byte arrays now return Neurotec.NBuffer.
- **ADD**: NImage methods for image creation from/wrapping of byte array.

Version 2.4.0.0

- **ADD**: PNG format support.
- **ADD**: JPEG2000 format support.

Version 2.3.0.2

- **FIX**: Wrapped images disposing issues.

Version 2.3.0.1

- **FIX**: Cross-AppDomain usage issues.

Version 2.3.0.0

- **ADD**: Image rotation, flipping and cropping functionality.

Version 2.2.1.1

- **ADD**: NImages.GetInfo method.

Version 2.2.1.0

- **CHN**: NImageFile Close and Dispose methods behavior to be consistent with .NET Dispose pattern and removed IsOpened property in Pro version.

Version 2.2.0.0

- **ADD**: Loading from Stream.
- **UPD**: Saving to Stream.

Version 2.1.0.2

- **UPD**: Updated internal structure.

Version 2.1.0.1

- **FIX**: Object disposing issues.

Version 2.1.0.0

- **UPD**: Updated to support changes in unmanaged code.

Version 2.0.3.1

- **FIX**: Minor fixes.

Version 2.0.3.0

- **UPD**: Updated to support changes in unmanaged code.

Version 2.0.2.0

- **ADD**: NImages.GetOpenFileFilter, NImages.GetSaveFileFilter, NImages.GetOpenFileFilterString and NImages.GetSaveFileFilterString methods.
- **FIX**: Reading of read-only files.

Version 2.0.1.0

- **ADD**: NImage.FromHandle method overload with bool value specifying whether NImage will own the specified handle.
- **CHN**: Now uses Microsoft .NET Framework 2.0.

Version 2.0.0.0

- **CHN**: Assembly renamed to Neurotec.Images.dll to be consistent with other components.
- **CHN**: Some interface changes.
- **ADD**: Support for monochrome and RGB images.
- **ADD**: Support for files with multiple images.
- **ADD**: NIST IHead format support.
- **ADD**: BMP and TIFF (load-only) low-level support.

Version 1.0.0.0

- Initial release.

8.3.2.4 Neurotec.Licensing

Version 3.1.0.0

- **ADD**: NLicense.IsComponentActivated method.
- **UPD**: Now uses Neurotec (see page 219) library version 3.1.

Version 3.0.0.0

- New version of the Neurotec.Licensing (see page 322) library released.

8.3.2.5 Neurotec.Video Library

Version 2.1.0.0

- **REM**: Obsolete members and types.
- **UPD**: Now uses Neurotec.Images (see page 241) library version 2.6.
- **UPD**: Now uses Neurotec (see page 219) library version 3.1.

Version 2.0.0.0

- **ADD**: NVideoWriter class.

Version 1.0.0.0

- Initial release of Neurotec.Video (see page 317) library.

8.3.2.6 Neurotec.SentiSight Library

Version 2.1.0.2

- **UPD**: Minor updates because of Neurotec (see page 219) changes.

Version 2.1.0.1

- **UPD**: Minor updates because of Neurotec (see page 219) changes.

Version 2.1.0.0

- **REM**: SERotRectD structure is removed.
- **CHN**: SEModelUpdateStatus (see page 218) is changed to SEStatus (see page 215).
- **CHN**: Parameters' names are renamed from Param* to Parameter*.
- **UPD**: SEEngine.Recognition (see page 11) ParameterThreshold and Threshold are added.
- **UPD**: SEEngine.Learning ParameterGeneralizationThreshold and GeneralizationThreshold are added.
- **CHN**: SentiSightEngine is obsolete use SEEngine.SentiSight class is added.
- **CHN**: SEShape.IsValid now fully checks shape for absence of self-intersection.
- **FIX**: Added missing SEShape.Points property.
- **CHN**: SEShape.PointCollection now always perform operations on points and does not return status. Use SEShape.IsValid to check validity of the shape.
- **UPD**: SEEngine.Separation.GetObjectModelSize is added.
- **CHN**: SEEngine.Separation.GetModelSize is obsolete SEEngine.Separation.GetObjectModelSize must be used.
- **UPD**: SEEngine.Separation.SaveObjectModelToMemory is added.
- **CHN**: SEEngine.Separation.SaveModelToMemory is obsolete SEEngine.Separation.SaveObjectModelToMemory must be used.
- **UPD**: SEEngine.Separation.LoadObjectModelFromMemory is added.
- **CHN**: SEEngine.Separation.LoadModelFromMemory is obsolete SEEngine.Separation.LoadObjectModelFromMemory must be used.
- **CHN**: New SEModel.Save metods.
- **UPD**: SEModel.Clone is added.
- **UPD**: SEEngine.Learning.AddToModel methods now report SEStatus (see page 215).
- **UPD**: SEEngine.Learning.AddToModel methods report SEStatus.NothingToLearn if model is not updated because image is not suitable for learning.
- **UPD**: New SEEngine.Learning.GeneralizeModel is added. The function status can be SEStatus.Succeeded or SEStatus.ModelIsEmpty if input model is empty. Use GeneralizationThreshold parameter to set / get value of generalization threshold.
- **CHN**: Old SEEngine.Learning.GeneralizeModel is obsolete new SEEngine.Learning.GeneralizeModel must be used.
- **UPD**: SEEngine.Recognition.GetModelIds is added.
- **CHN**: SEEngine.Recognition.ModelCount is obsolete SEEngine.Recognition.GetModelIds must be used.
- **CHN**: SEEngine.Recognition.AddModel now returns SEStatus (see page 215). The function status can be SEStatus.Succeeded or SEStatus.ModelIsEmpty if input model is empty.

- **UPD**: New SEEngine.Recognition.Recognize is added. Use Threshold parameter to set / get value of recognition threshold.
- **CHN**: Old SEEngine.Recognition.Recognize is obsolete new SEEngine.Recognition.Recognize must be used.
- **REM**: Obsolete defines, functions and members.
- **UPD**: Now uses NImages (see page 98) library version 2.6.
- **UPD**: Now uses NCore (see page 24) library version 3.1.

Version 2.0.0.0 Version 1.1.0.0 Version 1.0.0.0

- Initial release of Neurotec.SentiSight (see page 272) library.

9 Axis M1114

Axis M1114 is compact and adaptable network camera. To use this camera with SentiSight 2.1 SDK it is necessary to change configuration file (`CmmAxisIpCameras.ini`). User should enter camera's IP address in `CmmAxisIpCameras.ini` file:

rtsp://<ip address>

<ip address> - IP address of camera

`CmmAxisIpCameras.ini` file is saved in `\bin\Win32_x86\Cmm` folder.

Files

Axis M1114 network camera requires these files:

- `CmmAxis.dll`
- `ffmpeg libraries`: avcodec-52.dll, avdevice-52.dll, avformat-52.dll,avutil-49.dll, CmmCisco.dll, libcurl.dll, libeay32.dll, libsasl.dll, libssl32.dll, openldap.dll, ssleay32.dll, swscale-0.dll

Files for this camera are placed in `\bin\Win32_x86\Cmm\Additional\` folder. These files must be copied to `\bin\Win32_x86\Cmm\` before using this camera.

10 Support

Neurotechnology provides customer support during the entire period, while the customer develops and uses his own system using our products. If you face problems using our products, have any questions or suggestions, you can reach us by email: support@neurotechnology.com.

Index

D

E

F

H

I

J

L

M

N

O

P

R

S

T

U

V

W