NeroAPI

v5.5.9.14

The NeroAPI will only work with
a fully installed Nero version!

v1.52, Copyright 2002 Ahead Software AG

Ahead Software AG, Im Stockmaedle 18, 76307 Karlsbad, Germany

1. Contents

1. Contents ... 2
2. License Agreement ... 6
3. Introduction ... 7
3.1. Motivation ... 7
3.2. Overview ... 7
3.3. Requirements ... 7
3.4. Required Skills ... 8
3.5. Compatibility Between Different NeroAPI Versions ... 8
3.5.1. Source Compatibility ... 8
3.5.2. Binary Compatibility ... 8
3.6. Related Topics ... 9
4. Quick Start ... 10
4.1. Running Precompiled Sample Applications ... 10
4.1.1. NeroAPITest ... 10
4.1.2. NeroAPITest Comand Line Examples ... 12
4.2. Compiling the Samples ... 13
4.3. Accessing the *NeroAPI* in Your Applications ... 13
4.4. Points of Consideration ... 13
5. Detailed Discussion of the NeroAPITest Sample ... 14
6. Creating A Simple MFC Application ... 17
6.1. Nero Fiddled While Rome Burned! ... 17
6.2. Creating The Framework ... 17
6.3. Adding NeroAPI files ... 19
6.4. Adding Controls ... 20
6.5. Adding Member Variables ... 22
6.5.1. Variables For Controls ... 22
6.5.2. Other Variables ... 23
6.6. Adding Message Handling Functions For Controls ... 24
6.6.1. OnBrowse() ... 24
6.6.2. OnBurn() ... 25
6.6.3. OnOK() ... 28
6.6.4. OnCancel() ... 28
6.6.5. OnAbort() ... 28
6.7. Adding Utility Functions ... 30
6.7.1. NeroAPIInit() ... 30
6.7.2. NeroAPIFree() ... 33
6.7.3. AppendString() ... 33
6.8. Adding Callback Functions ... 34
6.8.1. IdleCallback() ... 34
6.8.2. UserDialog() ... 35
6.8.3. ProgressCallback() ... 36
6.8.4. AbortedCallback() ... 37
6.8.5. AddLogLine() ... 37
6.8.6. SetPhaseCallback() ... 37
6.9. Build And Run NeroFiddles ... 38

7. API Types and Functions ... 39
7.1. Types ... 39
7.1.1. NERO_DEVICEHANDLE ... 39
7.1.2. NERO_CALLBACK ... 39
7.1.3. NERO_IDLE_CALLBACK ... 40
7.1.4. NERO_PROGRESS_CALLBACK ... 40
7.1.5. NERO_ABORTED_CALLBACK ... 40
7.1.6. NERO_MAJOR_PHASE_CALLBACK ... 40
7.1.7. NERO_IO_CALLBACK ... 41
7.1.8. NERO_STATUS_CALLBACK ... 41
7.1.9. NERO_ADD_LOG_LINE_CALLBACK ... 41
7.1.10. NERO_SET_PHASE_CALLBACK ... 41
7.1.11. NERO_DISABLE_ABORT_CALLBACK ... 41
7.1.12. NERO_PROGRESS ... 42
7.1.13. NERO_DLG_WAITCD_MEDIA_INFO ... 42
7.1.14. NERO_SETTINGS ... 42
7.1.15. NEROAPI_INIT_ERROR ... 44
7.1.16. NEROAPI_OPTION ... 44
7.1.17. NEROAPI_SCSI_DEVTYPE ... 45
7.1.18. NERO_SPEED_INFOS ... 45
7.1.19. NERO_ACCESSTYPE ... 46
7.1.20. NERO_MEDIA_TYPE ... 46
7.1.21. NERO_MEDIA_SET ... 48
7.1.22. NERO_SCSI_DEVICE_INFO ... 48
7.1.23. NERO_SCSI_DEVICE_INFOS ... 50
7.1.24. NERO_TRACK_TYPE ... 50
7.1.25. NERO_TRACK_INFO ... 50
7.1.26. NERO_MEDIUM_TYPE ... 51
7.1.27. NERO_CD_INFO ... 52
7.1.28. NERO_TEXT_TYPE ... 53
7.1.29. NERO_MAJOR_PHASE ... 53
7.1.30. NERO_IO ... 54
7.1.31. NERO_ISO_ITEM ... 55
7.1.32. NERO_AUDIO_ITEM_HANDLE ... 56
7.1.33. NERO_AUDIO_ITEM_INFO ... 56
7.1.34. NERO_DATA_EXCHANGE_TYPE ... 56
7.1.35. NERO_TRACKMODE_TYPE ... 57
7.1.36. NERO_FREESTYLE_TRACK ... 57
7.1.37. NERO_DATA_EXCHANGE ... 58
7.1.38. NERO_AUDIO_TRACK ... 58
7.1.39. NERO_VIDEO_ITEM_TYPE ... 59
7.1.40. NERO_VIDEO_ITEM ... 59
7.1.41. NERO_CD_FORMAT ... 60
7.1.42. NERO_WRITE_CD ... 61
7.1.43. NERO_WRITE_IMAGE ... 62
7.1.44. NERO_WRITE_FREESTYLE_CD ... 63
7.1.45. NEROAPI_BURN_ERROR ... 64
7.1.46. NeroCITEArgs ... 65

7.1.47. NERO_WRITE_VIDEO_CD ... 65
7.1.48. NERO_WRITE_FILE_SYSTEM_CONTENT ... 66
7.1.49. NERO_CONFIG_RESULT ... 67
7.1.50. NERO_AUDIO_FORMAT_INFO ... 67
7.2. Functions ... 68
7.2.1. NeroSetExpectedAPIVersion(); ... 68
7.2.2. NeroSetExpectedAPIVersionEx() ... 68
7.2.3. NeroFreeMem() ... 68
7.2.4. NeroGetLastError() ... 69
7.2.5. NeroGetErrorLog() ... 69
7.2.6. NeroClearErrors() ... 69
7.2.7. NeroGetAPIVersion() ... 70
7.2.8. NeroGetAPIVersionEx() ... 70
7.2.9. NeroInit() ... 70
7.2.10. NeroSetOption() ... 71
7.2.11. NeroDone() ... 71
7.2.12. NeroGetAvailableDrivesEx() ... 71
7.2.13. NeroGetTypeNameOfMedia() ... 71
7.2.14. NeroOpenDevice() ... 72
7.2.15. NeroCloseDevice() ... 72
7.2.16. NeroCreateIsoItem() ... 72
7.2.17. NeroCreateIsoItemOfSize() ... 73
7.2.18. NeroFreeIsoItem() ... 73
7.2.19. NeroCopyIsoItem ... 73
7.2.20. NeroCreateIsoTrackEx() ... 73
7.2.21. NeroFreeIsoTrack() ... 74
7.2.22. NeroImportIsoTrackEx() ... 74
7.2.23. NeroFreeCDStamp() ... 75
7.2.24. NeroGetCDInfo() ... 75
7.2.25. NeroGetAvailableSpeeds ... 76
7.2.26. NeroBurn() ... 76
7.2.27. NeroDAE() ... 77
7.2.28. NeroIsDeviceReady() ... 78
7.2.29. NeroEjectLoadCD() ... 78
7.2.30. NeroWaitForMedia ... 78
7.2.31. NeroGetLocalizedWaitCDTexts ... 79
7.2.32. NeroGetCDRWErasingTime () ... 79
7.2.33. NeroEraseCDRW() ... 79
8. ISO Track Creation ... 80
9. ISO Track Classes ... 81
9.1. Overview ... 81
9.2. class CNeroDataCallback ... 82
9.3. class CNeroIsoHandle ... 82
9.4. class CNeroIsoIterator ... 83
9.5. struct CNeroIsoEntry ... 83
9.6. struct CNeroIsoTrack ... 84
10. The FileSystemContent Interface ... 87
10.1. Overview ... 87

10.2. Namespace setting ... 88
10.3. InterfaceBase ... 88
10.4. File System Reading Interfaces ... 88
10.4.1. IFileContent ... 89
10.4.2. IDirectoryEntry ... 89
10.4.3. IDirectory ... 90
10.4.4. IFileSystemContent ... 90
10.5. File System Content Creation Interfaces ... 90
10.5.1. IDataInputStream ... 90
10.5.2. IFileProducer ... 91
10.5.3. IDirectoryEntryContainer ... 91
10.5.4. IDirectoryContainer ... 92
10.5.5. IFileSystemDescContainer ... 93
11. Media Type Formats ... 94
11.1. Audio ... 94
11.2. Video ... 94
11.2.1. SVCD Creation with *Nero* ... 94
12. Error Codes ... 96
12.1. Verification Errors ... 96
12.2. WMA Item Errors ... 96
12.3. MPEG Audio Item Errors ... 97
12.4. Wave Audio Item Errors ... 97
12.5. SCSI Hard-disk Page File Errors ... 97
12.6. AVI VCD File Item Errors ... 97
12.7. ISO DOC Errors ... 98
12.8. Transfer Errors ... 98
12.9. Image Compression Errors ... 99
12.10. CD Recording Errors ... 100
12.11. Audio Generation Errors ... 104
12.12. Image Generation Errors ... 104
12.13. ISO 9660 File System Generation Errors ... 105
12.14. Joliet Name Creation Errors ... 105
12.15. Phase Errors ... 106
12.16. MMC Errors ... 107
12.17. MP 6200 Errors ... 112
13. Known Limitations ... 117
14. Bibliography ... 118
14.1. C Programming Books ... 118
14.2. C Programming Online Resources ... 118
14.3. C++ Programming Books ... 118
14.4. C++ Online Resources ... 119
14.5. General CD/CD-ROM Online Resources ... 119
14.6. Audio CD Online Resources ... 119
14.7. Super Video CD Online Resources ... 119

2. License Agreement

IMPORTANT: PLEASE READ THE SOFTWARE LICENSE AGREEMENT ("LICENSE") CAREFULLY BEFORE USING THE SOFTWARE.

USING THE SOFTWARE INDICATES YOUR ACKNOWLEDGMENT THAT YOU HAVE READ THE LICENSE AND AGREE TO ITS TERMS.

The license agreement is contained in a text file, “NeroSDK_License.txt”, to be found in the root folder of the installation package.

3. Introduction

3.1. Motivation

The *NeroSDK* is a tool for inclusion of *Nero* functionality in your own applications through the use of the *NeroAPI*.

Since it became available we have provided documentation in form of source code comments and a readme-file.
That kind of documentation proved as being ample for seasoned professionals, though somewhat tedious.
Soon more and more people began to use this SDK. It became obvious that a "manual within the code" was not enough for programmers who are less familiar with getting the grip on somebody else's source code.
You asked for this documentation, we have created it. We hope this little manual to be just what you need.

Unfortunately nothing is ever perfect. So the author would be grateful if you sent your suggestions or pointed out errors, both in our code and documentation.

3.2. Overview

This paper, the documentation of the *NeroAPI*, contains some practical guidelines on how to use the API (Application Programming Interface) that is implemented in the NeroAPI.dll with the help of NeroSDK (Nero Software Development Kit). The SDK is available for OEMs (Original Equipment Manufacturers) and registered users of *Nero*.

We believe that it will help you add CD- and DVD-burning capability to your own applications in less time. A detailed discussion of what the programming samples do, and how they do it, together with a list of all types and functions, should enable you to get your routines working in no time.

A brief description of *Nero's* Audio-, Video- and Super-Video-CD capabilities will ensure that you do not fail because of using the wrong file format.

3.3. Requirements

This documentation assumes that *Nero* 5.5.9.14 is already installed on your computer. The current NeroSDK version (NeroSDK 1.0.1) supports the commandset of Nero 5.5.9.14.

Depending on your individual needs, your *NeroAPI*-based application might work with an earlier version. Therefore, this documentation will indicate which *NeroAPI* introduced a particular feature, function, or type.

For additional information please take a look at *3.5 Compatibility Between Different NeroAPI Versions*.

3.4. Required Skills

This documentation is directed towards Software developers who have gathered some experience in programming C or C++.
It is absolutely required that you know the basic concepts of the C programming language to use the *NeroSDK*.

If you have no or little experience with C/C++, you will find a list of books and Internet addresses that we regard as very useful for learning the language.
C and C++ are still the most commonly used programming languages, and once you've mastered them you will learn any other programming language with ease.

3.5. Compatibility Between Different NeroAPI Versions

Since several programs must access the same *NeroAPI* package, not all of them can be updated every time a new version of *Nero* is released.

Nero needs to ensure both source and binary compatibility with its future versions. Version 5.0.3.9 of *Nero* and the following versions meet that requirement, while older versions do not.

3.5.1. Source Compatibility

Applications written for one version of *NeroSDK* must work with more recent versions of *Nero*, without having to change their source code.

3.5.2. Binary Compatibility

Applications written for one version of *NeroSDK* must work with more recent versions of *Nero* without having to compile the application again.

3.6. Related Topics

Closely related to *NeroAPI* is *NeroCOM*, a Type Library for the Component Object Model. *NeroCOM* presents another approach for accessing the power of Nero.

Please look up additional information concerning *NeroCOM* at http://www.nero.com.

4. Quick Start

4.1. Running Precompiled Sample Applications

4.1.1. NeroAPITest

This application can

- read information about a CD
- burn audio CDs
- burn ISO CDs
- burn UDF (Universal Disc Format) CDs
- burn ISO/UDF CDs
- burn DVDs
- burn mixed mode CDs
- burn Video and Super Video CD
- extract CDA tracks.

Open a command window (MS-DOS shell) and type “NeroAPITest “ followed by a command and in most cases a parameter list.

See the following table for valid parameters. Square brackets indicate that a parameter is optional. However, when writing an Audio/ISO CD, you have to supply at least one valid set of parameters.

Command	**Function**	
--listdrives	List available drives	
	Parameters	**Description**
	None	
Command	**Function**	
--cdinfo	Get information about a CD	
	Parameters	**Description**
	--drivename ‘x’	Supply drive letter.
Command	**Function**	
--write	Write Audio/ISO/UDF/Mixed Mode CD or DVD	
	Parameters	**Description**
	--drivename 'x'	Supply drive letter.
	[--real]	Do not simulate burning process.
	[--TAO]	Track At Once.
	[--bup]	Burn with buffer underrun protection.
	[--writebuffersize ‘x’]	Set the size of the write buffer in KByte.
	[--artist ‘artist’]	Supply artist’s name for Audio CD.
	[--title ‘title’]	Supply Audio CD title.

Command	Function	
	[--speed 'x']	Select desired speed.
	[--pause x]	Pause length in blocks. Valid pause lengths are 0 to 7500. In TAO only default pause length is supported (150).
	[--audioinindex0]	Write audio data into index 0 to prevent silent pauses between tracks. TAO and –audioinindex0 are mutually exclusive.
	['audio file1'] ['audio file2'] ...	List of Audio files to burn.
	[--cdextra]	Use the CDExtra feature. Two sessions will be created, the first containing Audio tracks, the second containing one ISO track.
	[--iso 'volume name']	Name the ISO-volume.
	[--udf 'volume name']	UDF
	[--isoudf 'volume name']	ISO and UDF
	[--dvd]	Burn ISO DVD
	[--iso-no-joliet]	No long filenames.
	[--iso-mode2]	Select ISO mode 2.
	['dir/file1'] ['dir/file2'] ...	List of files to burn. Can be directory tree or file.
--write	Write Video CD	
	Parameters	**Description**
	--drivename 'xxx'	Supply drive letter.
	--videocd	Selection of Video CD type
	[--real]	Do not simulate burning process.
	[--TAO]	Track At Once.
	[--bup]	Burn with buffer underrun protection.
	[--writebuffersize 'x']	Set the size of the write buffer in KByte.
	[--speed 'x']	Select desired speed.
	['mpeg/jpeg file1'] *['mpeg/jpeg file2']…*	List of Video files.
Command	**Function**	
--write	Write Super Video CD	
	Parameters	**Description**
	--drivename 'xxx'	Supply drive letter.
	--svideocd	Selection of Video CD type
	[--real]	Do not simulate burning process.
	[--TAO]	Track At Once.
	[--bup]	Burn with buffer underrun protection.
	[--writebuffersize 'x']	Set the size of the write buffer in KByte.
	[--speed 'x']	Select desired speed.

Command	Function	
	'mpeg/jpeg file1' ['mpeg/jpeg file2']…	List of Video files to burn.
Command	**Function**	
--read	Copy CD tracks to files	
	Parameters	**Description**
	-- drivename 'x'	Supply drive letter.
	-- 'xy' 'file1' [-- 'xy' 'file2']…	Read contents of track with number 'xy' into 'file 1'. The file name has to include the suffix. Only ".pcm" and ".wav" will be accepted.
Command	**Function**	
--erase	Erase a CD-RW	
	Parameters	**Description**
	[--entire]	By default a quick erase is done, where the actual content of the CD is not erased.The "—entire" option sweeps the whole CD, insuring that no data can be retrieved by any means afterwards.
	-- drivename 'x'	Supply drive letter.
--eject	Ejects a CD from the drive	
	Parameters	**Description**
	-- drivename 'x'	Supply drive letter.
Command	**Function**	
--load	Loads a CD into the drive	
	Parameters	**Description**
	-- drivename 'x'	Supply drive letter.

4.1.2. NeroAPITest Comand Line Examples

Simulate the burning of a mixed mode CD with one audio track and one file using the CD recorder with the drive letter "D":
NeroAPITest --write --drivename d c:\media\audio\police01.wav --iso mycd c:\data\file.dat

Burn the content of a folder:
NeroAPITest --write --drivename d --real --iso mycd c:\data

4.2. Compiling the Samples

- Start Visual C++.
- Select *Files* and *Open Workspace* from the menu.
  An *Open*-dialog will come up.
- Select *project files* as file type (“.dsp”-suffix)
- Navigate to the samples directory and select the project you want to open.
- Click the *OK*-button.
- Open the *Build*-menu and then select *Build All*.

4.3. Accessing the *NeroAPI* in Your Applications

- Make the *.h files of the *NeroAPI*-include-directory accessible from your program
- Link your project with the NeroAPIGlue library
- Use the *NeroAPI* functions in proper order. Take the samples for reference.
- Implement the callback functions.

4.4. Points of Consideration

Make sure the required DLLs can be found by the application, by installing *Nero*.

The NeroInit() and NeroDone() functions must not be called from the destructor of a global object or from a DllMain() function. If they are called like that, the result will be a deadlock.

To burn WMA files onto CD, the "Windows Media Format 7 SDK" must be installed. This can be found on

http://msdn.microsoft.com/workshop/imedia/windowsmedia/sdk/wmsdk.asp

5. Detailed Discussion of the NeroAPITest Sample

NeroAPITest,cpp is the source file that defines what the application actually does. It contains the main() function which is the application's entry point. Having a main() function identifies it as console application (as opposed to a true Windows application that has a winmain() function).

At the beginning of NeroAPITest.cpp, you will find a few include-directives. Those are references to other files that contain some required definitions:
NeroAPIGlue.h is responsible for attaching NeroAPITest to the NeroAPI.DLL.

The rest of the include files are required for communicating with your Operating System. They let you access device-Input/Output, handle special keystroke combinations like Ctrl-C (which will cause the current task to abort), display characters on your computer screen, and perform many other important background tasks.

Line 43: From line 43 on you will find function declarations. All listed functions are implemented within NeroAPITest.cpp. Most of them are callback-functions, and thus responsible for giving NeroAPI access to NeroAPITest whenever necessary, e.g. to display a progress bar or obtain user input.

Line 64: Beginning with line 64, types and variables are defined, and termination behaviour is implemented by a sequence of *NeroAPI* calls in the Exit() function, starting in line 115. The order of those API calls is very important, and if some API functions that free memory were not called, the application would allocate memory and not free it when terminating.

Line 133: The function Usage() will display a help-text, concerning the proper usage of program arguments or command line parameters.

Line 158: The function main() starts by initializing variables and then parses command line parameters to determine what the user wants to do. According to those parameters variables are set, indicating what kind of device is to be used, whether the application should read or write data, what kind of data, and where the data comes from.

Line 440: The function signal () tells Windows to call our SigCtrlC() function when Ctrl-C is pressed.

Then the application tries to load the NeroAPI.dll by calling NeroAPIGlueConnect(). If it cannot be found, an error message is displayed, and the application is terminated by calling our Exit() function, providing error code 10 as a parameter.
During NeroAPIGlueConnect() the Windows registry is queried for the shared *NeroAPI*.

Line 448: A call to NeroGetAPIVersionEx retrieves the API's version number. The version number, contained in four WORD values, is then printed on the screen.

Line 456: The *NeroAPI* is initialized. The *NeroAPI* will work in Demo mode if no Serial Number is found.

Line 478: If the write buffer size is not 0, NeroSetOption() is called, setting in the *NeroAPI*. The actual value is calculated by taking the user-provided parameter and multiplying it by 1024 (1 KByte), because the user is expected to give the buffer size in KBytes.

Line 483: The program then tries to get a list of all CD-ROM devices in the system by calling NeroGetAvailableDrivesEx(). If no device is present, the application will terminate.

Line 490: If a drive name has been specfied by the *–drivename* command, the program searches for that device in the list of available devices. The program then tries to open it for further use by calling NeroOpenDevice(). If no device handle is returned (e.g. if the drive letter given is not present in the system), the application will terminate with an error message.

If *–listdrives* was passed as a command line parameter, the list of available devices will be printed.

Line 544: If a device handle could not be retrieved, the application terminates.

Line 549: This code section retrieves and displays a list of allowed speeds for the selected drive.

Line 557: Loading a CD is handled after this line.

Line 561: Erase-behavior is implemented here. The application checks whether a CD is present in the drive, whether it is of the right type, and whether the CD should undergo a quick erase or a complete sweep, to make it ready for rewriting.

Line 579: The “Eject CD” part has been implemented here.

Line 583: Line 583 to 692 are dedicated to getting Information about a CD or reading tracks. The “read”-part after line 637 scans for all available tracks and tries to determine whether they are PCM or WAV. The respective format will be read from the CD.
If any error occurs, the application is stopped after displaying an error message.

Line 694: If the user requested to write a CD-ROM or DVD, the code after line 661 checks whether creation of ISO/Audio CD, ISO DVD, or VCD/SVCD was specified. The size of the CD/DVD directory is calculated, and the program tries to allocate memory for the NERO_WRITE_CD structure, that will be used for writing the information. If the free-memory-pool is not large enough, the application will terminate.

Then the application fills the NERO_WRITE_CD structure with the information the user provided.

Line 770: The burn process is started by calling NeroBurn(), passing a pointer to the NERO_WRITE_CD structure, which has just been filled with content.

Line 784: Burning of VCD/SVCD is handled here. Basically this means filling a NERO_WRITE_VIDEO_CD structure and passing it to the NeroBurn() function.

Line 840: This part of the code deals with burning an image. Allowed formats are NRG, ISO and CUE.

After the CD has been burnt, the log file is updated, the allocated memory is freed, and the application terminates without error.

Line 910: The implementation of the idle callback is empty, apart from an assertion that ensures that the NeroAPI actually returned the same pointer provided as user data. In a GUI application this function would probably do a little more than just to return control to the NeroAPI and passing the aborted-flag.

Line 931: Here the user can reply to a *NeroAPI* request by keyboard input. CharIO() will take an array of mappings from character to NeroUserDlgInOut constants, and return the proper NeroUserDlgInOut constant depending on the user input.

Line 953: The UserDialog callback implementation displays a number of options, depending on the value of “type”. It then calls CharIO() to get the user’s desired option. The corresponding NeroUserDlgInOut constant is then returned to the *NeroAPI*.

Line 1172: Here we can find the implementation of the various callback functions that the *NeroAPI* requires.

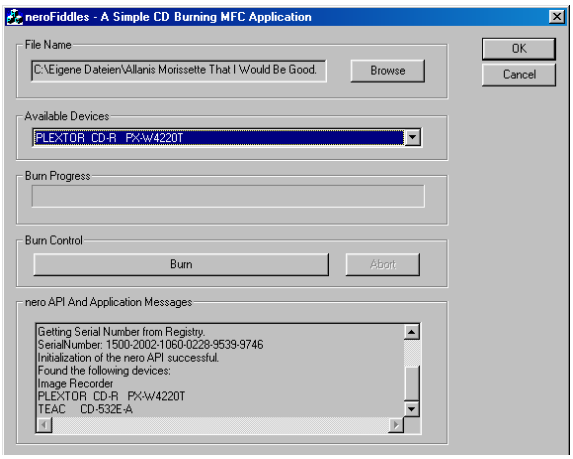
E. g. ProgressCallback(), whenever called by the *NeroAPI*, will display the current progress of the burn progress in percent.

Line 1271: The NeroError() function obtains the last error from the *NeroAPI*, prints some information about the action that failed, lets the *NeroAPI* free some memory and makes the application terminate with exit code 10.

6. Creating A Simple MFC Application

6.1. Nero Fiddled While Rome Burned!

It is quite obvious that the world has been waiting for an application that gives some tribute to important events of the past. So we will call this one “NeroFiddles”. Application names don’t come any better.



Our Nero - of course - has a lot more to do than just fiddle.

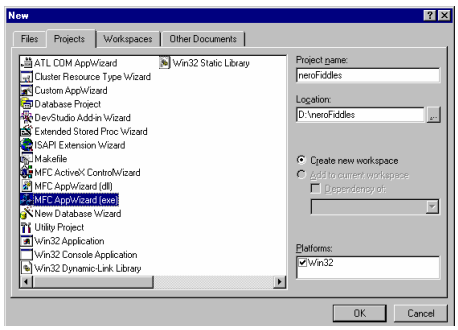
This simple application lists the available devices that can burn CDs. It lets the user choose a file and burns an ISO CD which contains this single file.
Nothing spectacular, and less powerful than the command line examples in the previous chapter.

But to prevent the application from becoming too cluttered and hard to understand we have to keep it simple. GUI applications have a much bigger overhead than console applications. So we’ll just provide minimum functionality to keep the program small and simple. Once it works it is not that complicated to provide additional functionality. Getting started is the hard part.

6.2. Creating The Framework

This tutorial is based on Visual C++ 6.0. Visual Studio .NET screens may differ!
Also Nero has to be installed on your system to run this sample!

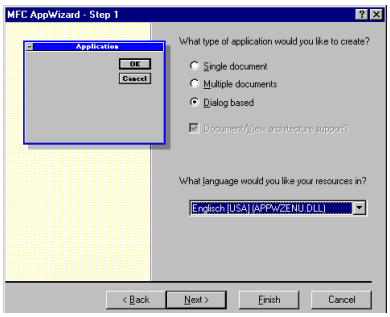
Open Microsoft Visual C++.



Select File/New from the menu. The “Projects” tab already should be selected when the “New” dialog opens.

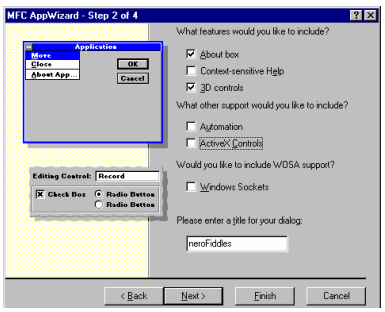
Select “MFC AppWizard (exe)” and type “NeroFiddles” into the “Project name” edit box. You may select your favorite project directory, but leave the rest of the settings as they are. Click on the “OK” button.

In the “MFC AppWizard Step 1” dialog select “Dialog based” and choose the preferred language for your resources. “English (USA) (APPWZENU.DLL)” should work fine, so let’s just pick this entry. Click “Next”.

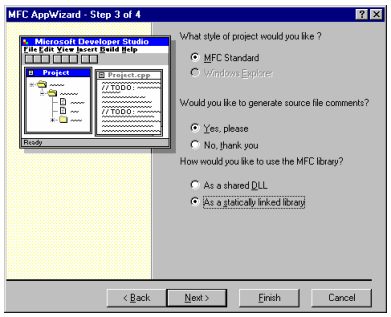


Uncheck “ActiveX Controls” in “MFC AppWizard Step 2” because we will not use any.

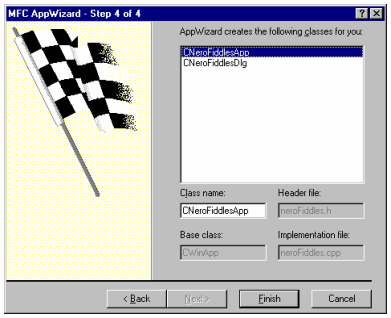
Click “Next”.



Select “As a statically linked library” in “MFC AppWizard Step 3 of 4”. The application becomes bigger, but we do not depend on the presence of the MFC DLLs on the target system. Leave the rest of the settings as they are. Click “Next”.

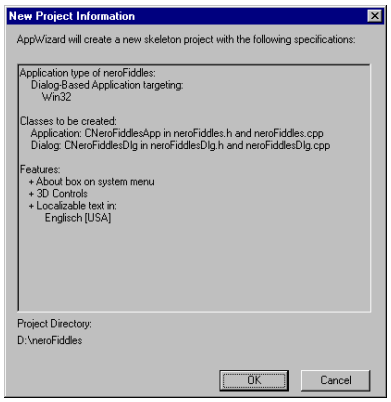


Click “Next”, do not make any changes and then click “Finish”.



A dialog pops up, telling you about what the AppWizard created for you. Click “OK”.

We now have a working MFC dialog based application that can be compiled and executed, though it doesn’t do much of what we need, yet.



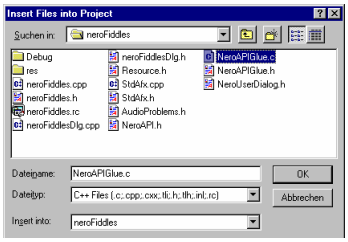
6.3. Adding NeroAPI files

Make sure that you can see the workspace window on the left side. If it is not there, activate it by selecting View/Workspace from the menu.

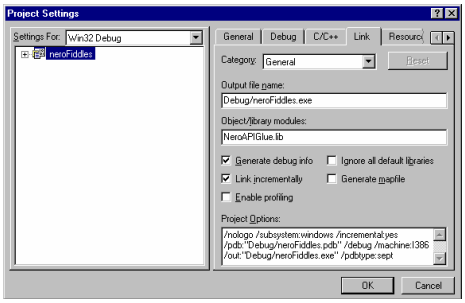
Before we do anything else we need to make sure that everything we need from the *NeroAPI* is in its place. Go to your *NeroAPI* folder and copy the contents of the “include” and “lib” subdirectories to your NeroFiddles-project directory. You will need to copy the following files:
NeroAPI.h, NeroAPIGlue.h, NeroUserDialog.h and NeroAPIGlue.lib.

The required *NeroAPI* files now are located as desired. Return to Visual Studio.

Select Project/Add To Project/Files from the menu. Select “NeroAPIGlue.h” from the “Insert Files Into Project” dialog that comes up. Hold down the <Ctrl> key and also select “NeroAPI.h” and “NeroUserDialog.h”. Click “OK”.



Go to Project/Settings. Select the “Link” tab, pick “Input” in the “Category” dropdown-listbox. Type “NeroAPIGlue.lib” in the “Object/library modules” edit field and “libcmt.lib” in the “Ignore libraries” edit field.



Select the FileView tab in the Workspace window. Open “NeroFiddles files”/ “Header Files”/ “StdAfx.h” by double clicking it. Within that file you should find something like this:

```
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdtctl.h>           // MFC support for Internet Explorer 4
Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
```

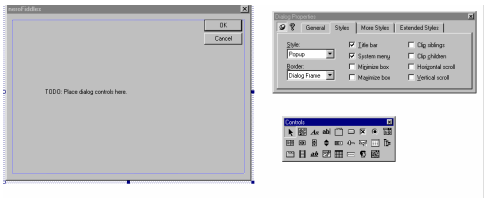
Right after that, add the following line:

```
#include "NeroAPIglue.h"
```

6.4. Adding Controls

Usually, the Resource View tab of the workspace window will already be selected and display a skeleton of our application. If you cannot see it, select the Resource View tab, and open NeroFiddles resources/Dialog/IDD_NEROFIDDLES_DIALOG.

You should see two buttons “OK”, “Cancel”, and a line of text that says “TODO: place your dialog controls here.”



That’s what we intended to do anyway. So click on that line of text, and delete it by pressing the “del”-key.

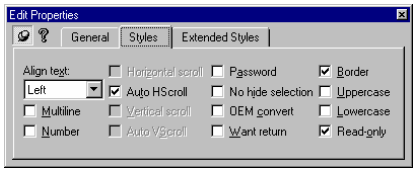
Resize the dialog window a little bit, so that it becomes bigger. There’s no need to squeeze everything into the small amount of space Visual Studio initially offers us.

Now click on the “Edit Box” icon in the little “Controls” window. You should place a Edit box control in the upper left corner of the NeroFiddles dialog. Resize the control so that it can display a little more text.

Right click over the Edit box and select “Properties” from the context menu that pops up. You will now see a “Edit properties” dialog. Click the “Keep Visible” pinboard-style pin icon in the upper left corner of the

property dialog – we will need this dialog more than once.

Rename IDC_EDIT1 to IDC_FILENAME. Select the “Styles” tab and make the control “Read only”, it should now have a gray color. We want it to be “read only” because the user should rather browse for a file than type its name, which is error prone.



We need a button that will later open a FileOpen dialog. Click on the button icon in the “Controls” window. Place the button to the right of the Edit control that you just inserted.

Change the name from “IDC_BUTTON1” to “IDC_BROWSE” in the “Push Button Properties” window. Change the “Caption” from “Button1” to “Browse”.

We also need a ComboBox that displays the available devices and lets the user select one for burning. Click on the “Combo Box” symbol of the “Control” window. Place the ComboBox under the Edit Box and and resize it. Rename it from “IDC_COMBO1” to “IDC_DEVICES”. Select the “Styles” tab and change the type to “droplist” – the user then cannot enter any information, but has to choose from the options our application gives him, which is exactly what we want.

Now add a Progress Control and another button. Rename that button from “IDC_BUTTON2” to “IDC_BURN”. Change the caption to “Burn”. Make the button “disabled”. It will be enabled after the user has selected a file for burning.

Place another button to the right of IDC_BURN. Rename that button to IDC_ABORT. Change the caption to “Abort”. Make the button “disabled”, this is the initial state when the application starts. The Abort button will be enabled when the user pushes the Burn button.

Add another Edit Box, Rename it from “IDC_EDIT2” to “IDC_MESSAGES”. Select the “Styles” tab and make it “multiline” and “read only”. Also check “Horizontal Scroll”, “AutoHScroll”, “Vertical Scroll” and “AutoVScroll”. Now resize the Edit Box so that it can display about ten lines of text.

That completes our work with the Resource Editor. You can enhance the appearance by using some group boxes if you want.

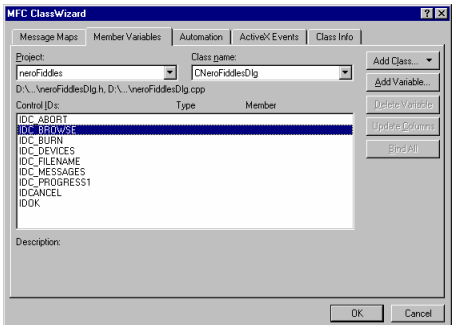
6.5. Adding Member Variables

If we build and run our application now, we see that it basically looks like what we wanted, but it doesn't do much so far. To include functionality we have to provide a few member variables, that map to controls and handle interchange with the NeroAPI.

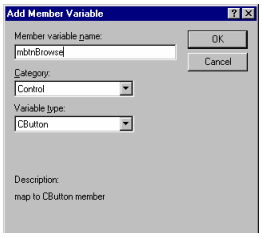
6.5.1. Variables For Controls

The controls we added need to be mapped to variables to provide easy access.

Open the ClassWizard (View/ClassWizard or Ctrl+W) and select the "Member Variables" tab. CNeroFiddlesDlg should already be selected as "Class name".



Click on "IDC_BROWSE" in the "Control IDs" list box. Click on "Add Variable", and in the following dialog provide the variable with the name "mbtnBrowse", make the Category "Control" and the variable type "CButton". Click "OK".



Now select "IDC_BURN", click on "Add Variable" and name it "mbtnBurn", category "Control", type "CButton".

IDC_ABORT gets a variable named "mbtnAbort", category Control, type CButton.

IDC_DEVICES becomes mcbxDevices, category Control (careful here: the default is value!), variable type CComboBox.

IDC_FILENAME is mapped to name medtFileName, Control, type CEdit.

IDC_MESSAGES becomes medtMessages, Control, CEdit.

IDC_PROGRESS1 maps to mpgsProgress, Control, CProgressCtrl.

IDCANCEL maps to mCancel, Control, CButton.

IDOK maps to mOK, Control, CButton.

The controls now have corresponding member variables and can be used quite easily.

6.5.2. Other Variables

We need to add numerous other variables to our dialog class.

Open the “ClassView” tab in the Workspace window. Right click on the “CNeroFiddlesDlg” class. Select “Add Member Variable” from the context menu. In the dialog that comes up enter “CString” for variable type and “mstrPathName” for variable name. Set “Access” to “private” and click “OK”.

Repeat this for CString mstrFileName, also private.

Enter “NERO_DEVICEHANDLE” as type and “ndhDeviceHandle” as variable name. Make it “private” and click on “OK”.

Repeat this with “NERO_SCSI_DEVICE_INFOS*” as type and “pndiDeviceInfos” as name. Make it “private”.

Do this for all of the following:

The name pncdCDInfo is of type NERO_CD_INFO*.

nsSettings is of type NERO_SETTINGS.

npProgress is of type NERO_PROGRESS.

writeCD is of type NERO_WRITE_CD.

mniiFile is of type NERO_ISO_ITEM.

dwVersion is of type DWORD.

pFile is of type FILE*.

pcDriveName [128] is of type char.

pcNeroFilesPath [128] is of type char.

pcVendor [128] is of type char.

pcSoftware [128] is of type char.

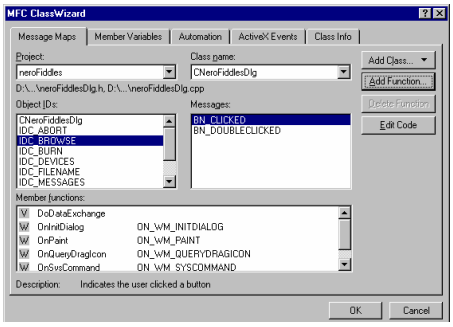
pcLanguageFile [128] is of type char.

mbAborted is of type bool.

6.6. Adding Message Handling Functions For Controls

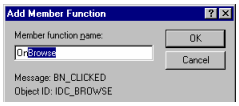
6.6.1. OnBrowse()

The first “real” functionality we add is the selection of a file and the display of its name in IDC_FILENAME.



We need the ClassWizard again. If you closed it, reopen it and select the “Message Maps” tab. “ClassName” should still be CNeroFiddlesDlg. Select IDC_BROWSE from “Object IDs” and “BN_CLICKED” from “Messages”. Click “Add Function”.

Accept the proposed function name, which is “OnBrowse”, by clicking “OK”.



We have now added a message handler that calls the OnBrowse() member function whenever the “Browse” button is clicked.

Click on “Edit Code”. The ClassWizard disappears and a source file window opens and displays the content of the OnBrowse function:

```
void CNeroFiddlesDlg::OnBrowse()
{
  // TODO: Add your control notification handler code here

}
```

Type the following after the line that starts with “TODO”. (To make things easier, you might as well copy it from here, if you obtained this document as a file.)

```
  static char BASED_CODE szFilter[] = "MP3 Files (*.mp3)|*.mp3|All Files
(*.*)|*.*||";
  CFileDialog dlgOpen(TRUE, NULL, NULL, OFN_FILEMUSTEXIST, szFilter,
this);
  if (dlgOpen.DoModal() == IDOK)
  {
    mstrPathName = dlgOpen.GetPathName();
    mstrFileName = dlgOpen.GetFileName();
    medtFileName.SetWindowText(mstrPathName);
    if (pndiDeviceInfos->nsdisNumDevInfos > 0)
```

```
    {
      mbtnBurn.EnableWindow(true);
    }
  }
```

This code defines a control string for our preferred file type, which is MP3. The string has to have a certain format so that it can be passed to the CFileDialog constructor. If the user clicks “OK” in the FileDialog, it will pass “IDOK” as return value. mstrPathName and mstrFileName now hold the selected file’s name, mstrPathName the full name, and mstrFileName the file name without path. Then the file name, including the path, is displayed in the Edit Box.

Afterwards, the functions checks whether any Devices have been enumerated during startup. If this is true, the Burn-button is enabled, otherwise it stays grayed.

(We could have mapped the Edit control to a string directly, but we are lazy and do not want to think too much. It is easier to keep track of everything, if we know that all controls map to control variables. Apart from that, we do not have to use the UpdateData() function. But that is a completely different issue.)

6.6.2. OnBurn()

This function is connected to the IDC_BURN button and is supposed to start the burn process later. We will now add it to our application.

Open the ClassWizard, select the “Message Maps” tab. “ClassName” should still be CNeroFiddlesDlg. Select IDC_BURN from “Object IDs” and “BN_CLICKED” from “Messages”. Click “Add Function” and accept the proposed function name, which is “OnBurn”, by clicking “OK”.

Click on “Edit Code”, and type the following after the line that starts with “TODO”. (The AppendString() method will be introduced later.)

```
  if (mstrFileName == "")
  {
    AppendString("You have to choose a file before you can start
burning!");
  }
  else
  {
    strcpy(mniiFile.fileName, mstrFileName);
    strcpy(mniiFile.sourceFilePath, mstrPathName);
    mniiFile.isDirectory=FALSE;
    mniiFile.isReference=FALSE;
    mniiFile.nextItem=NULL;

    memset(&writeCD,0,sizeof(writeCD));
    writeCD.nwcdpCDStamp=NULL;
    writeCD.nwcdArtist=NULL;
    writeCD.nwcdTitle=NULL;
```

```
    writeCD.nwcdCDExtra=FALSE;
    writeCD.nwcdNumTracks=0;

    writeCD.nwcdMediaType = MEDIA_CD;

    int i = mcbxDevices.GetCurSel();

    NERO_SCSI_DEVICE_INFO* nsdiDevice =
(NERO_SCSI_DEVICE_INFO*)mcbxDevices.GetItemDataPtr(i);

    ndhDeviceHandle = NeroOpenDevice(nsdiDevice);

    if (!ndhDeviceHandle)
    {
      AppendString("Device could not be opened: "+(CString)nsdiDevice-
>nsdiDeviceName);
    }
    else
    {
      mbtnAbort.EnableWindow(true);
      mCancel.EnableWindow(false);
      mOK.EnableWindow(false);
      mpgsProgress.SetRange(0,100);
      mcbxDevices.EnableWindow(false);
      mbtnBrowse.EnableWindow(false);
      mbtnBurn.EnableWindow(false);

      writeCD.nwcdIsoTrack = NeroCreateIsoTrackEx(&mniiFile,
"neroFiddles", NCITEF_CREATE_ISO_FS|NCITEF_USE_JOLIET);

      int iRes = NeroBurn(ndhDeviceHandle, NERO_ISO_AUDIO_CD, &writeCD,
NBF_WRITE, 0, &npProgress);

      NeroFreeIsoTrack(writeCD.nwcdIsoTrack);
      NeroCloseDevice(ndhDeviceHandle);

      mbtnAbort.EnableWindow(false);
      mCancel.EnableWindow(true);
      mOK.EnableWindow(true);
      mpgsProgress.SetPos(0);
      mcbxDevices.EnableWindow(true);
      mbtnBrowse.EnableWindow(true);
      mbtnBurn.EnableWindow(true);

      mbAborted = false;
      char* Log = NeroGetErrorLog();
      AppendString(Log);
      switch(iRes)
      {
        case NEROAPI_BURN_OK:
```

```
          AppendString ("BurnCD() : burn successful");
          break;
        case NEROAPI_BURN_UNKNOWN_CD_FORMAT:
          AppendString ("BurnCD() : unknown CD format");
          break;
        case NEROAPI_BURN_INVALID_DRIVE:
          AppendString ("BurnCD() : invalid drive");
          break;
        case NEROAPI_BURN_FAILED:
          AppendString ("BurnCD() : burn failed");
          break;
        case NEROAPI_BURN_FUNCTION_NOT_ALLOWED:
          AppendString ("BurnCD() : function not allowed");
          break;
        case NEROAPI_BURN_DRIVE_NOT_ALLOWED:
          AppendString ("BurnCD() : drive not allowed");
          break;
        default:
          AppendString ("BurnCD() : unknown error");
      }
    }
  }
```

You might have noticed that this code has few comments, to say the least. The author was not driven by laziness here, but rather wanted to prevent this tutorial from becoming monstrous. You will find source code comments in the NeroFiddles files that come with the *NeroAPI*. Here we will briefly explain what the code is supposed to do (and hopefully does).

First, the function checks whether or not the user had selected a file name by using the Browse button. If he did not, the function returns, doing nothing but adding an admonishing line to the message log.

If a file was selected, the NERO_ISO_ITEM structure mniiFile is filled and the NERO_WRITE_CD structure is initialized.

The index of the selected ComboBox entry is retrieved and used for getting a pointer to the respective NERO_SCSI_DEVICE_INFO, which is stored as a void-pointer. Therefore it needs to be casted.

Then the NERO_WRITE_CD structure is filled with the required information.

The function tries to open this device and store a handle in NeroDeviceHandle.

If the device handle is 0, meaning that the device could not be opened, a log line is added and the function returns.

If the device handle is valid, the nwcdIsoTrack member of NERO_WRITE_CD is assigned to a CNeroIsoTrack pointer.

The burn process is started. Burning is actually done and **not** simulated.

When the NeroBurn() function returns, the ISO track is freed, and the device is closed.

The return value of the NeroBurn() function is evaluated, and a corresponding line is added to the message log.

6.6.3. OnOK()

When the application closes, we also need to properly disconnect from the NeroAPI.DLL. This means that we need to have handling-functions that intercept when the user clicks “OK” or “Cancel.

Open the ClassWizard and select the “Message Maps” tab. Select IDOK from “Object IDs” and “BN_CLICKED” from “Messages”. Click “Add Function” and accept the proposed function name, which is “OnOK”, by clicking “OK”.

Click on “Edit Code” and you will see this:

```
void CNeroFiddlesDlg::OnOK()
{
  // TODO: Add extra validation here

  CDialog::OnOK();
}
```

Type the following after the line that contains “TODO”.

```
NeroAPIFree();
```

6.6.4. OnCancel()

Open the ClassWizard and select the “Message Maps” tab. Select IDCANCEL from “Object IDs” and “BN_CLICKED” from “Messages”. Click “Add Function” and accept the proposed function name, which is “OnCancel”, by clicking “OK”.

Change the function to look like this:

```
void CNeroFiddlesDlg::OnCancel()
{
  // TODO: Add extra cleanup here
  NeroAPIFree();

  CDialog::OnCancel();
}
```

6.6.5. OnAbort()

If the “Abort” button is pressed while burning, the member variable “mbAborted” will be set to true. The value of mbAborted will be returned to the *NeroAPI* during

Process- and Idle-callbacks. If it becomes “true”, the *NeroAPI* will stop the burn process.

Open the ClassWizard and select the “Message Maps” tab. Select IDC_ABORT from “Object IDs” and “BN_CLICKED” from “Messages”. Click “Add Function” and accept the proposed function name, which is “OnAbort”, by clicking “OK”.

Click on “Edit Code” and change the function to look like this:

```
void CNeroFiddlesDlg::OnAbort()
{
  // TODO: Add your control notification handler code here
  mbAborted = true;
}
```

6.7. Adding Utility Functions

6.7.1. NeroAPIInit()

The initialization of the NeroAPI will be performed during OnInitDialog(). However, there is a lot to do, so we will not add the code there, but rather create a function that initializes the API.

This function will be named NeroAPIInit(). The first thing we need to do is adding a line in OnInitDialog to calls this function.

Open the CNeroFiddlesDlg tree in ClassView and locate “OnInitDialog”. Double click “OnInitDialog”.

You will now see the body of this function. Go right to the end of it, where you should find some code that looks like this:

```
// Set the icon for this dialog.  The framework does this automatically
//  when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);            // Set big icon
SetIcon(m_hIcon, FALSE);           // Set small icon

// TODO: Add extra initialization here

return TRUE;  // return TRUE  unless you set the focus to a control
```

In the line after “TODO” type “NeroAPIInit();”. That will call the - still non existing - function NeroAPIInit() during initialization.

We will provide this function now.

Right click on CNeroFiddlesDlg and choose “Add Member Function”. Set the Function Type to “void” and the Function Declaration to “NeroAPIInit”. Click “OK”.

Visual Studio will add the function declaration, create the function body, and get you right into the function body. Fill the function with the following code:

```
  mbAborted = false;

  AppendString("Opening NeroAPI.DLL");

  if (!NeroAPIGlueConnect (NULL)) {
    AppendString("Cannot open NeroAPI.DLL");
    return;
  }

  AppendString("Retrieving version information.");

  WORD majhi, majlo, minhi, minlo;
```

```
NeroGetAPIVersionEx(&majhi, &majlo, &minhi, &minlo, NULL);

CString strVersion;
strVersion.Format("Nero API version %d.%d.%d.%d",
                  majhi, majlo, minhi, minlo);

AppendString(strVersion);


AppendString("Filling NERO_SETTINGS structure");

strcpy(pcNeroFilesPath, "NeroFiles");
strcpy(pcVendor, "ahead");
strcpy(pcSoftware, "Nero - Burning Rom");
strcpy(pcLanguageFile, "Nero.txt");

memset(&nsSettings, 0, sizeof(nsSettings));
nsSettings.nstNeroFilesPath = pcNeroFilesPath;
nsSettings.nstVendor = pcVendor;
nsSettings.nstIdle.ncCallbackFunction = IdleCallback;
nsSettings.nstIdle.ncUserData = this;
nsSettings.nstSoftware = pcSoftware;
nsSettings.nstUserDialog.ncCallbackFunction = UserDialog;
nsSettings.nstUserDialog.ncUserData = this;
nsSettings.nstLanguageFile =pcLanguageFile;

memset(&npProgress, 0, sizeof(npProgress));
npProgress.npAbortedCallback = AbortedCallback;
npProgress.npAddLogLineCallback = AddLogLine;
npProgress.npDisableAbortCallback = NULL;
npProgress.npProgressCallback = ProgressCallback;
npProgress.npSetPhaseCallback = SetPhaseCallback;
npProgress.npSetMajorPhaseCallback=NULL;
npProgress.npSubTaskProgressCallback=NULL;
npProgress.npUserData = this;

pndiDeviceInfos = NULL;

NEROAPI_INIT_ERROR initErr;
initErr = NeroInit (&nsSettings, NULL);

switch (initErr)
{
  case NEROAPI_INIT_OK:
    AppendString("Initialization of the NeroAPI successful.");
    break;
  case NEROAPI_INIT_INVALID_ARGS:
    AppendString("The arguments are not valid.");
    break;
```

```
    case NEROAPI_INIT_INVALID_SERIAL_NUM:
      AppendString("The Serial Number is not valid.");
      break;
    default:
      AppendString("An error occured. The type of error cannot be
determined.");
      break;
  }

  pndiDeviceInfos = NeroGetAvailableDrivesEx (MEDIA_CD, NULL);

  if (!pndiDeviceInfos) {
    AppendString("NeroGetAvailableDrives() returned no available
devices.");
  }
  else
  {
    if (pndiDeviceInfos->nsdisNumDevInfos > 0)
    {
      AppendString("Found the following devices:");
      for (DWORD dDeviceCounter = 0; dDeviceCounter < pndiDeviceInfos-
>nsdisNumDevInfos; dDeviceCounter++)
      {
        AppendString(pndiDeviceInfos-
>nsdisDevInfos[dDeviceCounter].nsdiDeviceName);

        int i = mcbxDevices.AddString(pndiDeviceInfos->
nsdisDevInfos[dDeviceCounter].nsdiDeviceName);

        mcbxDevices.SetItemDataPtr(i, &pndiDeviceInfos-
>nsdisDevInfos[dDeviceCounter]);
      }

      mcbxDevices.SelectString(-1, pndiDeviceInfos-
>nsdisDevInfos[0].nsdiDeviceName);
    }
    else
    {
      AppendString("The number of available devices is 0.");
    }
  }
```

The NERO_SETTINGS and NERO_PROGRESS structures are initialized and then filled with pointers to callback functions and the this-pointer.

The result of the call to NeroInit() is evaluated and added to the message log.

Then the available drives are added to the Devices-ComboBox, linking each entry with a pointer to a NERO_SCSI_DEVICE_INFO.

6.7.2. NeroAPIFree()

This function disconnects NeroFiddles from the NeroAPI and is called when our application closes.

Right click on CNeroFiddlesDlg and choose “Add Member Function”. Set the Function Type to “void” and the Function Declaration to “NeroAPIFree”. Click “OK”.

Visual Studio will add the function declaration, create the function body and set the cursor to the function body. Fill the function with the following code:

```
if (pndiDeviceInfos)
{
  NeroFreeMem(pndiDeviceInfos);
}

NeroClearErrors();
NeroDone();

NeroAPIGlueDone();

return;
```

6.7.3. AppendString()

Right click on CNeroFiddlesDlg and choose “Add Member Function”. Set the Function Type to “void” and the Function Declaration to “AppendString (CString str)”. Click “OK”.

Fill the function with the following code:

```
CString   strBuffer;

medtMessages.GetWindowText (strBuffer);
if (!strBuffer.IsEmpty())
{
   strBuffer += "\r\n";
}
strBuffer += str;
medtMessages.SetWindowText (strBuffer);

medtMessages.LineScroll (medtMessages.GetLineCount(), 0);
```

6.8. Adding Callback Functions

The one remarkable thing about the use of the callback functions is the this-pointer.

The ncUserData member of the NERO_CALLBACK structure is supposed to hold a pointer to the calling object in a C++ environment. We filled that pointer with a dummy value in the console applications, because there is no such pointer when you are not using classes and objects.

For NeroFiddles it is vital, though. If we do not hand over that pointer to the NeroAPI and retrieve it in our callback functions, we will not be able to access any non-static member of our CNeroFiddlesDlg class. This would mean that we could not update the progress bar or print messages, which is a must.

We set the this-pointer during NeroAPIInit():

```
nsSettings.nstUserDialog.ncUserData = this;
```

The NeroAPI stores the pointer, and what we need to do is retrieve it. It is handed over to our callback functions as void* pUserData. We have to cast it to a CNeroFiddlesDlg pointer. The usage looks like this:

```
bool bSomeBooleanVariable = ((CNeroFiddlesDlg*)pUserData)->mbAborted;
```

6.8.1. IdleCallback()

IdleCallback will be called continuously during a burn process. If the user clicked the “Abort” button, mbAborted becomes true, and the API will be told to stop burning.

Use the known ClassView approach to add a member function. Set the Function Type to “BOOL NERO_CALLBACK_ATTR” and the Function Declaration to “ IdleCallback (void *pUserData)”. Activate the “static” Checkbox.

Now change the function body to this:

```
BOOL NERO_CALLBACK_ATTR CNeroFiddlesDlg::IdleCallback(void *pUserData)
{
  static MSG msg;
  while (!(((CNeroFiddlesDlg*)pUserData)->mbAborted) &&
::PeekMessage(&msg,NULL,NULL,NULL,PM_NOREMOVE))
  {
    if (!AfxGetThread()->PumpMessage())
        {
      break;
```

```
        }
      }
  return ((CNeroFiddlesDlg*)pUserData)->mbAborted;
}
```

The first part of the function ensures that, while Nero is burning, the application still can process messages.

6.8.2. UserDialog()

The UserDialog() callback function is designed to let the user make a choice, or tell the NeroAPI that the user had finished a task, which the NeroAPI required him to perform. To keep the sample as small as possible, we will only provide user-controlled handling where absolutely required.

Add a member function of Function Type “NeroUserDlgInOut NERO_CALLBACK_ATTR” and make the Function Declaration “ UserDialog (void *pUserData, NeroUserDlgInOut type, void *data)”. Activate the “static” Checkbox.

Make the function body look like this:

```
NeroUserDlgInOut NERO_CALLBACK_ATTR CNeroFiddlesDlg::UserDialog(void
*pUserData, NeroUserDlgInOut type, void *data)
{
  switch (type)
  {
    case DLG_AUTO_INSERT:
      return DLG_RETURN_CONTINUE;
      break;
    case DLG_DISCONNECT_RESTART:
      return DLG_RETURN_ON_RESTART;
      break;
    case DLG_DISCONNECT:
      return DLG_RETURN_CONTINUE;
      break;
    case DLG_AUTO_INSERT_RESTART:
      return DLG_RETURN_EXIT;
      break;
    case DLG_RESTART:
      return DLG_RETURN_EXIT;
      break;
    case DLG_SETTINGS_RESTART:
      return DLG_RETURN_CONTINUE;
      break;
    case DLG_OVERBURN:
      return DLG_RETURN_TRUE;
      break;
    case DLG_AUDIO_PROBLEMS:
```

```
      return DLG_RETURN_EXIT;
      break;
    case DLG_WAITCD:
      AfxMessageBox("Please insert a CDR or just click OK if you are
using the Image Recorder.");
      return DLG_RETURN_EXIT;
      break;
    case DLG_WAITCD_REMINDER:
      break;
    case DLG_WAITCD_DONE:
      return DLG_RETURN_EXIT;
      break;
    case DLG_FILESEL_IMAGE:
      {
        static char BASED_CODE szFilter[] = "Image Files
(*.nrg)|*.nrg|All Files (*.*)|*.*||";
        CFileDialog dlgOpen(TRUE, NULL, "test.nrg", OFN_OVERWRITEPROMPT,
szFilter, ((CNeroFiddlesDlg*)pUserData));
        if (dlgOpen.DoModal() == IDOK)
        {
          strcpy((char*)data,dlgOpen.GetPathName());
          return DLG_RETURN_TRUE;
        }
        else
        {
          return DLG_BURNIMAGE_CANCEL;
        }
      }
      break;
    default:
      break;
  }

  return DLG_RETURN_EXIT;
}
```

6.8.3. ProgressCallback()

The ProgressCallback() function will provide information on how much of the current process has been completed. We use this information for display in a progress bar.

Add a member function of Function Type “BOOL NERO_CALLBACK_ATTR” and make the Function Declaration “ProgressCallback (void *pUserData, DWORD dwProgressInPercent)”. Activate the “static” Checkbox.

Make the function body look like this:

```
BOOL NERO_CALLBACK_ATTR CNeroFiddlesDlg::ProgressCallback(void
*pUserData, DWORD dwProgressInPercent)
{
```

```
  ((CNeroFiddlesDlg*)pUserData)->
mpgsProgress.SetPos(dwProgressInPercent);
  return ((CNeroFiddlesDlg*)pUserData)->mbAborted;
}
```

6.8.4. AbortedCallback()

This function is used by the *NeroAPI* to check whether the current process is supposed to be terminated.

Add a member function of Function Type “BOOL NERO_CALLBACK_ATTR” and make the Function Declaration “AbortedCallback(void *pUserData)”. Activate the “static” Checkbox.

Make the function body look like this:

```
BOOL NERO_CALLBACK_ATTR CNeroFiddlesDlg::AbortedCallback(void
*pUserData)
{
  return ((CNeroFiddlesDlg*)pUserData)->mbAborted;
}
```

6.8.5. AddLogLine()

This function provides textual information about certain states that might be important for the application.

Add a member function of Function Type “void NERO_CALLBACK_ATTR” and make the Function Declaration “AddLogLine(void *pUserData, NERO_TEXT_TYPE type, const char *text)”. Activate the “static” Checkbox.

Make the function body look like this:

```
void NERO_CALLBACK_ATTR CNeroFiddlesDlg::AddLogLine(void *pUserData,
NERO_TEXT_TYPE type, const char *text)
{
  CString csTemp(text);
  ((CNeroFiddlesDlg*)pUserData)->AppendString("Log line:" + csTemp);
  return;
}
```

6.8.6. SetPhaseCallback()

This function provides textual information about the current phase of the burning process.

Add a member function of Function Type “void NERO_CALLBACK_ATTR ” and make the Function Declaration “ SetPhaseCallback(void *pUserData, const char *text)”. Activate the “static” Checkbox.

Make the function body look like this:

```
void NERO_CALLBACK_ATTR CNeroFiddlesDlg::SetPhaseCallback(void
*pUserData, const char *text)
```

```
{
  CString csTemp(text);
  ((CNeroFiddlesDlg*)pUserData)->AppendString("Phase: " + csTemp);
  return;
}
```

6.9. Build And Run NeroFiddles

We’re almost done. We have added everything that is required; now choose “Build/Rebuild All” from the menu and then “Build/Execute NeroFiddles.exe”.

If we did everything right, NeroFiddles should now be running.

NeroFiddles is almost screaming for additional functionality. You should check the command line examples, and get ideas there. E.g. you could enable it to burn more than one file or complete folders. You could complete the user-interaction part and provide burning of different formats.

You could provide RadioButtons to toggle between simulation of the burn process and actual burning. Also, you could add the “continue session” feature.

7. API Types and Functions

This paragraph describes the interface to the NeroAPI DLL.

7.1. Types

7.1.1. NERO_DEVICEHANDLE

is defined as a class pointer for C++ or a void pointer for standard C.

```
#ifdef  __cplusplus
class CRecorderInfo;
typedef CRecorderInfo *NERO_DEVICEHANDLE;
#else
typedef void *NERO_DEVICEHANDLE;
#endif
```

The __cplusplus preprocessor macro determines whether C++ or C is being compiled. This macro is predefined and gives the programmer the opportunity to use more sophisticated C++ constructs where possible, or substitute them with standard C where not.

Identifier	Introduced in *NeroAPI* version
NERO_DEVICEHANDLE	5.0.3.9

7.1.2. NERO_CALLBACK

Actually, this is a pointer to one of several different callback functions defined below. The this-pointer will be passed to the function as first parameter when it is called by the NeroAPI.

A callback function is an interface to another software to notify your application of changes. Windows makes extensive use of callback functions.

Data exchange between an application and NeroAPI is done with a function that gets a pointer to its own structure, a buffer pointer, and the amount of bytes to be read or written. It shall return the actual amount of bytes transferred. Other functions indicate that the end of the file has been reached (EOF) when reading, or that a serious error occurred.

```
typedef struct tag_NERO_CALLBACK
{
  void *ncCallbackFunction;
  void *ncUserData
} NERO_CALLBACK;
```

Identifier	Introduced in *NeroAPI* version
NERO_CALLBACK	5.0.3.9

7.1.3. NERO_IDLE_CALLBACK

During writing, or in several long running functions, control is transferred to the DLL. The application has to provide services and interact with the user via callback functions.

NERO_CALLBACK_ATTR is defined in "NeroUserDialog.h" and ensures that the same conventions are used for passing of parameters. NERO_IDLE_CALLBACK is called regularly during long running activities. Return TRUE if this activity shall be aborted.

```
typedef BOOL (NERO_CALLBACK_ATTR *NERO_IDLE_CALLBACK) (void *pUserData);
```

Identifier	Introduced in *NeroAPI* version
NERO_IDLE_CALLBACK	5.0.3.9

7.1.4. NERO_PROGRESS_CALLBACK

This function needs to return TRUE if the user wants to abort.

The application may provide callback functions to set the different parts of this display. All of them may be NULL.

```
typedef BOOL  (NERO_CALLBACK_ATTR *NERO_PROGRESS_CALLBACK)(void
*pUserData, DWORD dwProgressInPercent);
```

Identifier	Introduced in *NeroAPI* version
NERO_PROGRESS_CALLBACK	5.0.3.9

7.1.5. NERO_ABORTED_CALLBACK

```
typedef BOOL  (NERO CALLBACK ATTR *NERO ABORTED CALLBACK)
(void *pUserData);
```

Identifier	Introduced in *NeroAPI* version
NERO_ABORTED_CALLBACK	5.0.3.9

7.1.6. NERO_MAJOR_PHASE_CALLBACK

Let the application know which phase of the burn process NeroAPI is currently in.

```
typedef void (NERO_CALLBACK_ATTR *NERO_SET_MAJOR_PHASE_CALLBACK)(void
*pUserData,NERO_MAJOR_PHASE phase,void *reserved);
```

Identifier	Introduced in *NeroAPI* version
NERO_MAJOR_PHASE_CALLBACK	5.0.3.9

7.1.7. NERO_IO_CALLBACK

```
typedef DWORD (NERO_CALLBACK_ATTR *NERO_IO_CALLBACK)
(void *pUserData, BYTE *pBuffer, DWORD dwLen);
```

Identifier	Introduced in *NeroAPI* version
NERO_IO_CALLBACK	5.0.3.9

7.1.8. NERO_STATUS_CALLBACK

```
typedef BOOL (NERO_CALLBACK_ATTR *NERO_STATUS_CALLBACK)
(void *pUserData);
```

Identifier	Introduced in *NeroAPI* version
NERO_STATUS_CALLBACK	5.0.3.9

7.1.9. NERO_ADD_LOG_LINE_CALLBACK

A one-line text to be displayed. The text pointer becomes invalid after returning from this function.

```
typedef void  (NERO_CALLBACK_ATTR *NERO_ADD_LOG_LINE_CALLBACK)
(void *pUserData, NERO_TEXT_TYPE type, const char *text);
```

Identifier	Introduced in *NeroAPI* version
NERO_ADD_LOG_LINE_CALLBACK	5.0.3.9

7.1.10. NERO_SET_PHASE_CALLBACK

Set the phase line. Text pointer becomes invalid after returning from this function.

```
typedef void  (NERO_CALLBACK_ATTR *NERO_SET_PHASE_CALLBACK)(void
*pUserData, const char *text);
```

Identifier	Introduced in *NeroAPI* version
NERO_SET_PHASE_CALLBACK	5.0.3.9

7.1.11. NERO_DISABLE_ABORT_CALLBACK

Tells the main program whether the burn process can be interrupted or not.

```
typedef void (NERO_CALLBACK_ATTR *NERO_DISABLE_ABORT_CALLBACK)(void
*pUserData,BOOL abortEnabled);
```

Identifier	Introduced in *NeroAPI* version
NERO_DISABLE_ABORT_CALLBACK	5.0.3.9

7.1.12. NERO_PROGRESS

Is used for passing required callback function pointers to the NeroBurn() function. npDisableAbortCallback will be called only if the NBF_DISABLE_ABORT flag is given to the NeroBurn function.
npSubTaskProgressCallback provides the write buffer fill level.

```
typedef struct tag_NERO_PROGRESS
{
  NERO_PROGRESS_CALLBACK  npProgressCallback;
  NERO_ABORTED_CALLBACK  npAbortedCallback;
  NERO_ADD_LOG_LINE_CALLBACK  npAddLogLineCallback;
  NERO_SET_PHASE_CALLBACK  npSetPhaseCallback;
  NERO_DISABLE_ABORT_CALLBACK npDisableAbortCallback;
  NERO_SET_MAJOR_PHASE_CALLBACK npSetMajorPhaseCallback;
  NERO_PROGRESS_CALLBACK npSubTaskProgressCallback;
} NERO_PROGRESS;
```

Identifier	Introduced in *NeroAPI* version
NERO_PROGRESS	5.0.3.9
npSetMajorPhaseCallback	5.5.5.8
npSubTaskProgressCallback	5.5.6.6

7.1.13. NERO_DLG_WAITCD_MEDIA_INFO

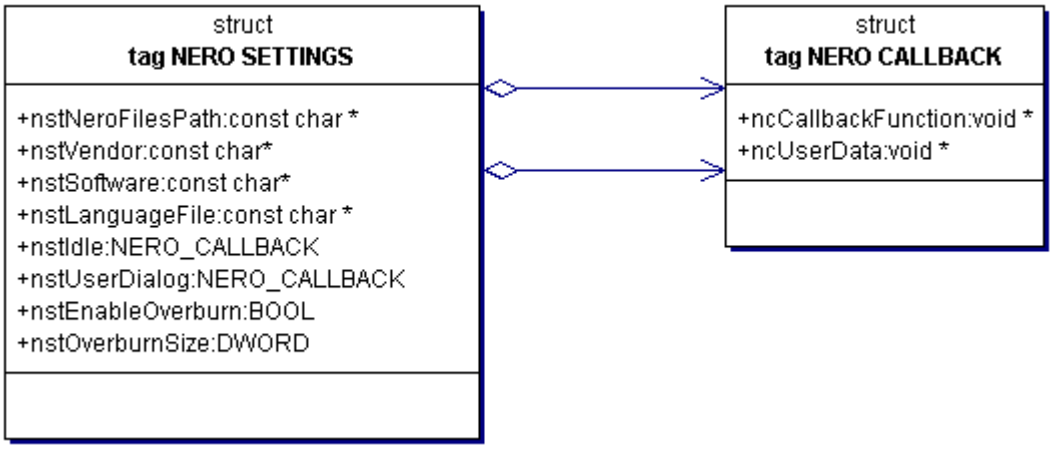
A pointer to this structure will be passed with the DLG_WAITCD_MEDIA_INFO user dialog callback.

```
typedef struct
{
      DWORD ndwmiSize;
      NERO_MEDIA_TYPE   ndwmiLastDetectedMedia;
      NERO_MEDIA_SET    ndwmiRequestedMedia;
      const char *ndwmiLastDetectedMediaName;
      const char *ndwmiRequestedMediaName;
} NERO_DLG_WAITCD_MEDIA_INFO;
```

Identifier	Introduced in *NeroAPI* version
NERO_DLG_WAITCD_MEDIA_INFO	5.5.9.4

7.1.14. NERO_SETTINGS

Has to be passed when calling the NeroInit() function.




```
typedef struct tag_NERO_SETTINGS
{
  const char  *nstNeroFilesPath;
  const char  *nstVendor, *nstSoftware;
  const char   *nstLanguageFile;
  NERO_CALLBACK   nstIdle;
  NERO_CALLBACK   nstUserDialog;
  BOOL    nstEnableOverburn;
  DWORD   nstOverburnSize;
} NERO_SETTINGS;
```

Description of structure members	
nstNeroFilesPath	Directory name with trailing '\' of where to find the additional Nero DLL and text files. No trailing '\' or '/' is necessary for NeroAPI Portable
nstVendor	Path for registry setting. Use “ahead”.
nstSoftware	Path for registry settings. Use "Nero - Burning Rom" for Nero application's settings.
nstLanguageFile	Name of the Nero language file; relative to nstNeroFilesPath (e.g. "Nero.txt")
nstIdle	NERO_IDLE_CALLBACK, may be NULL
nstUserDialog	NERO_USER_DIALOG, must not be NULL, see "NeroUserDialog.h" for details
nstEnableOverburn	Overburn settings: Overburning (writing more than the nominal capacity of a disc) is allowed if all of this is true: NstEnableOverburn == TRUE NstOverburnSize >= amount of required blocks for compilation The drive supports it DAO is used. Even then, overburning has to be acknowledged via callback (see DLG_OVERBURN in "NeroUserDialog.h").
nstOverburnSize	In blocks

Identifier	Introduced in *NeroAPI* version
NERO_SETTINGS	5.0.3.9

7.1.15. NEROAPI_INIT_ERROR

Used when informing the user about whether a call to NeroInit() was successful or not. Provides some additional information.

```
typedef enum
{
  NEROAPI_INIT_OK=0,
  NEROAPI_INIT_INVALID_ARGS,
  NEROAPI_INIT_UNSPECIFIED_ERROR,
  NEROAPI_INIT_INVALID_SERIAL_NUM,
  NEROAPI_INIT_DEMOVERSION_EXPIRED,
  NEROAPI_INIT_ALREADY_INITIALISED,
  NEROAPI_INIT_CANNOT_LOCK
} NEROAPI_INIT_ERROR;
```

Identifier	Introduced in *NeroAPI* version
NERO_INIT_ERROR	5.0.3.9
NEROAPI_INIT_DEMOVERSION_EXPIRED	5.5.1.1
NEROAPI_INIT_ALREADY_INITIALISED	5.5.2.4
NEROAPI_INIT_CANNOT_LOCK	5.5.5.2

7.1.16. NEROAPI_OPTION

Possible global *Nero* options. Used when calling NeroSetOption().

```
typedef enum
{
  NEROAPI_OPTION_MSG_FILE_NAME,
  NEROAPI_OPTION_WRITE_BUFFER_SIZE
} NEROAPI_OPTION;
```

Description of enumerators	
NEROAPI_OPTION_MSG_FILE_NAME	Used for changing the file name for the Nero error messages.
NEROAPI_OPTION_WRITE_BUFFER_SIZE	Set write buffer size. Value points onto an integer containing the size in byte.

Identifier	Introduced in *NeroAPI* version
NEROAPI_OPTION	5.0.3.9
NEROAPI_OPTION_WRITE_BUFFER_SIZE	5.5.5.0

7.1.17. NEROAPI_SCSI_DEVTYPE

Code to scan the SCSI/IDE bus and get information about the available WORM/CD-ROM drives.

```
typedef enum
{
  NEA_SCSI_DEVTYPE_UNKNOWN,
  NEA_SCSI_DEVTYPE_WORM,
  NEA_SCSI_DEVTYPE_CDROM,
  NEA_SCSI_DEVTYPE_UNSUPPORTED_WORM
} NEROAPI_SCSI_DEVTYPE;
```

Description of enumerators	
NEA_SCSI_DEVTYPE_UNKNOWN	Type information not available
NEA_SCSI_DEVTYPE_WORM	Write once. A CD-burner.
NEA_SCSI_DEVTYPE_CDROM	Read only. A CD-ROM drive.
NEA_SCSI_DEVTYPE_ UNSUPPORTED_WORM	Can write but is not supported by NeroAPI

Identifier	Introduced in *NeroAPI* version
NEROAPI_SCSI_DEVTYPE	5.0.3.9
NEA_SCSI_DEVTYPE_UNSUPPORTED_WORM	5.5.6.5

7.1.18. NERO_SPEED_INFOS

```
typedef struct tag_NERO_SPEED_INFOS
{
  DWORD  nsiNumSupportedSpeeds;
  DWORD nsiBaseSpeedKBs;
  DWORD nsiSupportedSpeedsKBs[64];
  DWORD nsiSupportedSpeeds[64];
  DWORD nsiReserved[32];
} NERO_SPEED_INFOS;
```

Description of structure members	
nsiNumSupportedSpeeds	1 if the speed cannot be changed.
nsiBaseSpeedKBs	Speed corresponding to 1X for the selected media in KB/s.
nsiSupportedSpeedsKBs	List of possible speeds in KB/s
nsiSupportedSpeeds	List of possible speeds in multiple of 150KB/s (1X for CD) (present for compatibility)
nsiReserved	

Identifier	Introduced in *NeroAPI* version
NERO_SPEED_INFOS	5.0.3.9

7.1.19. NERO_ACCESSTYPE

```
typedef enum
{
   ACCESSTYPE_WRITE,
   ACCESSTYPE_READ
} NERO_ACCESSTYPE;
```

Identifier	Introduced in *NeroAPI* version
NERO_ACCESSTYPE	5.5.9.14

7.1.20. NERO_MEDIA_TYPE

```
typedef enum tag_NERO_MEDIA_TYPE
{
      MEDIA_NONE                    =     0,
      MEDIA_CD                      =     0x00001,
      MEDIA_DDCD                    =     0x00002,
      MEDIA_DVD_M                   =     0x00004,
      MEDIA_DVD_P                   =     0x00008,
      MEDIA_DVD_ANY                 =     MEDIA_DVD_M|MEDIA_DVD_P,
      MEDIA_DVD_RAM                 =     0x00010,
      MEDIA_ML                      =     0x00020,
      MEDIA_MRW                     =     0x00040,

      //NeroAPI>=5.5.9.4:
      MEDIA_NO_CDR                  =     0x00080,
      MEDIA_NO_CDRW                 =     0x00100,
      MEDIA_CDRW                    =     MEDIA_CD|MEDIA_NO_CDR,
      MEDIA_CDR                     =     MEDIA_CD|MEDIA_NO_CDRW,
      MEDIA_DVD_ROM                 =     0x00200,
      MEDIA_CDROM                   =     0x00400,

      //NeroAPI>=5.5.9.10
      MEDIA_NO_DVD_M_RW             =     0x00800,
      MEDIA_NO_DVD_M_R              =     0x01000,
      MEDIA_NO_DVD_P_RW             =     0x02000,
      MEDIA_NO_DVD_P_R              =     0x04000,
      MEDIA_DVD_M_R                 =     MEDIA_DVD_M|MEDIA_NO_DVD_M_RW,
      MEDIA_DVD_M_RW                =     MEDIA_DVD_M|MEDIA_NO_DVD_M_R,
      MEDIA_DVD_P_R                 =     MEDIA_DVD_P|MEDIA_NO_DVD_P_RW,
      MEDIA_DVD_P_RW                =     MEDIA_DVD_P|MEDIA_NO_DVD_P_R,
      MEDIA_FPACKET                 =     0x08000,
      MEDIA_VPACKET                 =     0x10000,
      MEDIA_PACKETW                 =     MEDIA_MRW|MEDIA_FPACKET
                                                |MEDIA_VPACKET
} NERO_MEDIA_TYPE;
```

Description of enumerators	
MEDIA_NONE	No media present.
MEDIA_CD	CD-R/RW
MEDIA_DDCD	DDCD-R/RW
MEDIA_DVD_M	DVD-R/RW
MEDIA_DVD_P	DVD+RW
MEDIA_DVD_ANY	Any DVD-Recorder
MEDIA_DVD_RAM	DVD-RAM
MEDIA_ML	ML (Multi Level disc)
MEDIA_MRW	Mt. Rainier
MEDIA_NO_CDR	Exclude CD-R
MEDIA_NO_CDRW	Exclude CD-RW
MEDIA_CDRW	CD-RW
MEDIA_CDR	CD-R
MEDIA_DVD_ROM	DVD-ROM (non writable)
MEDIA_CDROM	CD-ROM (non writable)
MEDIA_NO_DVD_M_RW	Exclude DVD-RW
MEDIA_NO_DVD_M_R	Exclude DVD-R
MEDIA_NO_DVD_P_RW	Exclude DVD+RW
MEDIA_NO_DVD_P_R	Exclude DVD+R
MEDIA_DVD_M_R	DVD-R
MEDIA_DVD_M_RW	DVD-RW
MEDIA_DVD_P_R	DVD+R
MEDIA_DVD_P_RW	DVD+RW
MEDIA_FPACKET	Fixed Packetwriting
MEDIA_VPACKET	Variable Packetwriting
MEDIA_PACKETW	A bit mask for packetwriting

Identifier	Introduced in *NeroAPI* version
NERO_MEDIA_TYPE	5.5.4.3
MEDIA_NONE	5.5.9.4
MEDIA_NO_CDR	5.5.9.4
MEDIA_NO_CDRW	5.5.9.4
MEDIA_CDRW	5.5.9.4
MEDIA_CDR	5.5.9.4
MEDIA_DVD_ROM	5.5.9.4
MEDIA_CDROM	5.5.9.4
MEDIA_NO_DVD_M_RW	5.5.9.10
MEDIA_NO_DVD_M_R	5.5.9.10
MEDIA_NO_DVD_P_RW	5.5.9.10
MEDIA_NO_DVD_P_R	5.5.9.10
MEDIA_DVD_M_R	5.5.9.10
MEDIA_DVD_M_RW	5.5.9.10
MEDIA_DVD_P_R	5.5.9.10
MEDIA_DVD_P_RW	5.5.9.10
MEDIA_FPACKET	5.5.9.10

Identifier	Introduced in *NeroAPI* version
MEDIA_VPACKET	5.5.9.10
MEDIA_PACKETW	5.5.9.10

7.1.21. NERO_MEDIA_SET

NERO_MEDIA_SET represents a set of several media.

```
typedef DWORD NERO_MEDIA_SET;
```

Identifier	Introduced in *NeroAPI* version
NERO_MEDIA_SET	5.5.8.0

7.1.22. NERO_SCSI_DEVICE_INFO

```
typedef struct tag_NERO_SCSI_DEVICE_INFO
{
  char    nsdiDeviceName[32];
  char    nsdiHostAdapterName[8];
  DWORD   nsdiHostAdapterNo;
  DWORD   nsdiDeviceID;
  NEROAPI_SCSI_DEVTYPE   nsdiDevType;
  char    nsdiDriveLetter;
  DWORD   nsdiCapabilities;
  NERO_SPEED_INFOS  nsdiReadSpeeds;
  NERO_SPEED_INFOS  nsdiWriteSpeeds;
  const void        *nsdiDriver;
  char*   NsdiBufUnderrunProtName[64];
  DWORD   nsdiMandatoryBUPSpeed;
  NERO_MEDIA_SET    nsdiMediaSupport;
  DWORD   nsdiDriveBufferSize;
  DWORD   nsdiReserved[63];
 } NERO_SCSI_DEVICE_INFO;
```

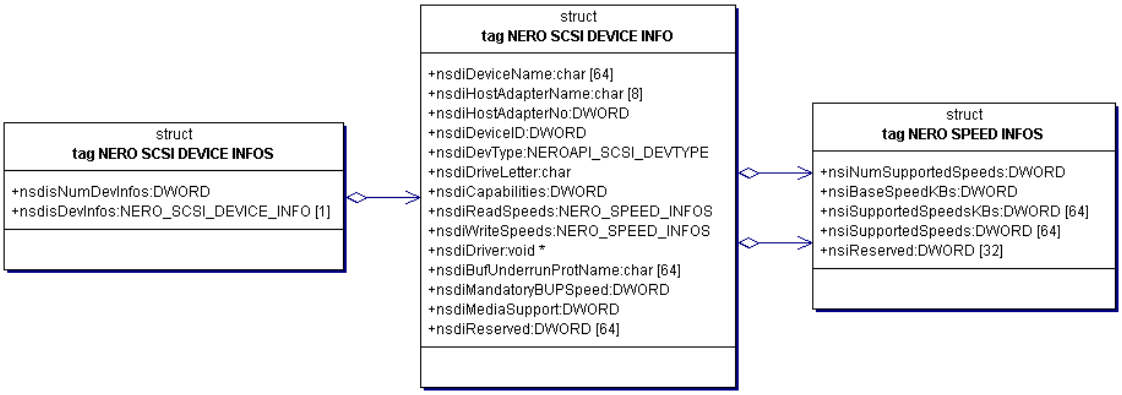
Description of structure members		
nsdiDeviceName		
nsdiHostAdapterName		
nsdiHostAdapterNo		
nsdiDeviceID		
nsdiDevType		
nsdiDriveLetter	Windows drive letter or 0 if not available.	
nsdiCapabilities	drive capabilities:	
	NSDI_ALLOWED	The drive can only be used if this bit is set.
	NSDI_DAO	Can write in DAO.
	NSDI_READ_CD_TEXT	Can read CD text.

Description of structure members		
	NSDI_VARIABLE_PAUSES_IN_TAO	See natPauseInBlksBeforeThisTrack below.
	NSDI_DAO_WRITE_CD_TEXT	Writes CD text in DAO (see natArtist/Title); never supported in TAO.
	NSDI_BURN_PROOF	Drive can use the burn proof mode. This flag is present for compatibility, better use the NSDI_BUF_UNDERRUN _PROT flag to support other technologies too
	NSDI_IMAGE_RECORDER	Drive is the image recorder.
	NSDI_UNDETECTED	
	NSDI_IDE_BUS	
	NSDI_SCSI_BUS	
	NSDI_BUF_UNDERRUN_PROT	Drive has a buffer underrun protection feature (not necessary Burn Proof)
	NSDI_RESERVED	Must not be used.
nsdiReadSpeeds	See declaration of NERO_SPEED_INFOS.	
nsdiWriteSpeeds	See declaration of NERO_SPEED_INFOS.	
nsdiDriver	Opaque identifier of the internal driver, required by NeroOpenDevice().	
nsdiBufUnderrunProt Name	Buffer underrun protection technology name The string will be empty if the technology has no name	
nsdiMandatoryBUPSp eed	It is highly recommended to enable buffer underrun protection when burning at this speed or faster. Contains 0 if there is no recommendation.	
nsdiMediaSupport	Bit field of supported media (constructed with the NERO_MEDIA_TYPE enum).	
nsdiDriveBufferSize	Drive buffer size (internal) in KB.	
nsdiReserved	Should be zero.	

Identifier	Introduced in *NeroAPI* version
NERO_SCSI_DEVICE_INFO	5.0.3.9
NSDI_BUF_UNDERRUN_PROT	5.5.0.6
nsdiBufUnderrunProtName	5.5.0.6
nsdiMandatoryBUPSpeed	5.5.3.2
nsdiMediaSupport	5.5.4.1 5.5.8.0: changed type from DWORD to NERO_MEDIA_SET
nsdiDriver	5.5.9.4 changed from void* to const void*
nsdiDriveBufferSize	5.5.9.4

7.1.23. NERO_SCSI_DEVICE_INFOS

Used to create a list of NERO_SCSI_DEVICE_INFO structures.




```
typedef struct tag_NERO_SCSI_DEVICE_INFOS
{
  DWORD   nsdisNumDevInfos;
  NERO_SCSI_DEVICE_INFO   nsdisDevInfos[1];
} NERO_SCSI_DEVICE_INFOS;
```

Description of structure members	
nsdisNumDevInfos	Number of entries in nsdisDevInfos.
nsdisDevInfos	See declaration of NERO_SCSI_DEVICE_INFO.

Identifier	Introduced in *NeroAPI* version
NERO_SCSI_DEVICE_INFOS	5.0.3.9

7.1.24. NERO_TRACK_TYPE

```
typedef enum
{
  NTT_UNKNOWN,
  NTT_DATA,
  NTT_AUDIO
} NERO_TRACK_TYPE;
```

Identifier	Introduced in *NeroAPI* version
NERO_TRACK_TYPE	5.0.3.9

7.1.25. NERO_TRACK_INFO

```
typedef struct tag_NERO_TRACK_INFO
{
  DWORD  ntiSessionNumber;
  DWORD   ntiTrackNumber;
  NERO_TRACK_TYPE  ntiTrackType;
```

```
  DWORD  ntiTrackStartBlk;
  DWORD  ntiTrackLengthInBlks;
  char  ntiArtist[65];
  char  ntiTitle[65];
  char  ntiISRC[13];
  DWORD  ntiReserved[29];
} NERO_TRACK_INFO;
```

Description of structure members	
ntiSessionNumber	Session Number
ntiTrackNumber	Track Number
ntiTrackType	Track Type (Audio, Data, Unknown)
ntiTrackStartBlk	Start Block of Track
ntiTrackLengthInBlks	Length of Track in Blocks
ntiArtist[65]	Name of Artist for Audio Tracks
ntiTitle[65]	Title of Song for Audio Tracks
ntiISRC[13]	If NGCDI_READ_ISRC is present: 12 chars ISRC (International Standard Recording Code) + terminator.
ntiReserved[29]	Should be zero

Identifier	Introduced in *NeroAPI* version
NERO_TRACK_INFO	5.5.8.3
ntiISRC	5.5.9.14
ntiReserved	5.5.9.14: Size decreased from 32 to 29.

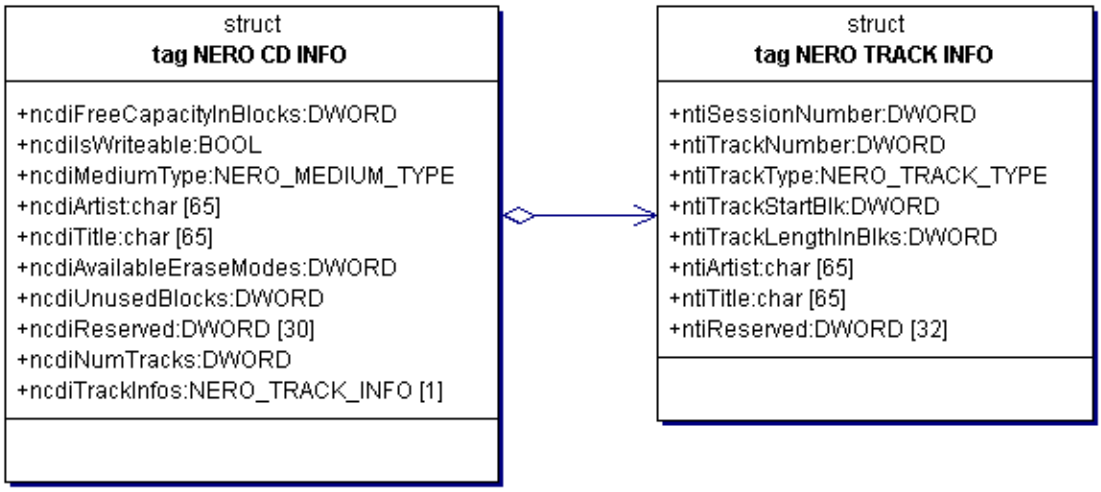
7.1.26. NERO_MEDIUM_TYPE

```
typedef enum
{
  NMT_UNKNOWN,
  NMT_CD_ROM,
  NMT_CD_RECORDABLE,
  NMT_CD_REWRITEABLE
} NERO_MEDIUM_TYPE;
```

Description of enumerators	
NMT_UNKNOWN	Unknown medium
NMT_CD_ROM	CD ROM
NMT_CD_RECORDABLE	CD Recordable (CDR)
NMT_CD_REWRITEABLE	CD Rewritable (CDRW)

Identifier	Introduced in *NeroAPI* version
NERO_MEDIUM_TYPE	5.0.3.9

7.1.27. NERO_CD_INFO




```
typedef struct tag_NERO_CD_INFO
{
  DWORD ncdiFreeCapacityInBlocks;
  BOOL  ncdiIsWriteable;
  NERO_MEDIUM_TYPE  ncdiMediumType;
  char  ncdiArtist[65];
  char  ncdiTitle[65];
  DWORD ncdiAvailableEraseModes;
  DWORD ncdiUnusedBlocks;

  DWORD ncdiReserved[29];
  DWORD ncdiNumTracks;
  NERO_TRACK_INFO  ncdiTrackInfos[1];
} NERO_CD_INFO;
```

Description of structure members	
ncdiFreeCapacityInBlocks	Number of unused blocks on CD.
ncdiIsWriteable	A disc can be non-writeable.
ncdiMediumType	Old media type description, ncdiMediaType should be used instead.
ncdiArtist	Artist name.
ncdiTitle	CD Title.
ncdiAvailableEraseModes	This bitfield can be decoded using the NCDI_IS_ERASE_MODE_AVAILABLE macro.
ncdiUnusedBlocks	Difference beetween Lead-Out position and last possible Lead-Out position.
ncdiMediaType	Type of media.
ncdiReserved	Should be zero.
ncdiNumTracks	Number of tracks.
ncdiTrackInfos	A List of NERO_TRACK_INFO structures.

Identifier	Introduced in *NeroAPI* version
NERO_CD_INFO	5.0.3.9
ncdiAvailableEraseModes	5.5.4.7
ncdiUnusedBlocks	5.5.5.8
ncdiMediaType	5.5.9.4
ncdiReserved	5.5.9.4: Reduced size from 30 to 29.

7.1.28. NERO_TEXT_TYPE

```
typedef enum
{
  NERO_TEXT_INFO,
  NERO_TEXT_STOP,
  NERO_TEXT_EXCLAMATION,
  NERO_TEXT_QUESTION,
  NERO_TEXT_DRIVE,
  NERO_TEXT_FILE,
  NERO_TEXT_UNSPECIFIED
} NERO_TEXT_TYPE;
```

Description of enumerators	
NERO_TEXT_INFO	Informative text.
NERO_TEXT_STOP	Some operation stopped prematurely
NERO_TEXT_EXCLAMATION	Important information.
NERO_TEXT_QUESTION	A question which requires an answer.
NERO_TEXT_DRIVE	A message concerning a CD-ROM drive or recorder.
NERO_TEXT_FILE	A message concerning a file.
NERO_TEXT_UNSPECIFIED	No type specified.

Identifier	Introduced in *NeroAPI* version
NERO_TEXT_TYPE	5.0.3.9

7.1.29. NERO_MAJOR_PHASE

```
typedef enum
{
      NERO_PHASE_UNSPECIFIED                  =-1,
      NERO_PHASE_START_CACHE                  =24,
      NERO_PHASE_DONE_CACHE                   =25,
      NERO_PHASE_START_TEST                   =28,
      NERO_PHASE_DONE_TEST                    =29,
      NERO_PHASE_START_SIMULATE               =32,
      NERO_PHASE_DONE_SIMULATE                =33,
      NERO_PHASE_START_WRITE                  =36,
      NERO_PHASE_DONE_WRITE                   =37,
      NERO_PHASE_START_SIMULATE_NOSPD         =61,
```

```
      NERO_PHASE_DONE_SIMULATE_NOSPD           =62,
      NERO_PHASE_START_WRITE_NOSPD             =65,
      NERO_PHASE_DONE_WRITE_NOSPD              =66,
      NERO_PHASE_ENCODE_VIDEO                  =82,
      NERO_PHASE_SEAMLESSLINK_ACTIVATED        =87,
      NERO_PHASE_BUP_ACTIVATED                 =90,
      NERO_PHASE_CONTINUE_FORMATTING           =99,
      NERO_PHASE_FORMATTING_SUCCESSFUL         =100,
      NERO_PHASE_DVDVIDEO_DETECTED             =111,
      NERO_PHASE_DVDVIDEO_REALLOC_STARTED      =112,
      NERO_PHASE_DVDVIDEO_REALLOC_COMPLETED    =113
      NERO_PHASE_DVDVIDEO_REALLOC_NOTNEEDED    =114,
      NERO_PHASE_DVDVIDEO_REALLOC_FAILED       =115
} NERO_MAJOR_PHASE;
```

Identifier	Introduced in *NeroAPI* version
NERO_MAJOR_PHASE	5.0.3.9
NERO_PHASE_BUP_ACTIVATED	5.5.7.8
NERO_PHASE_DVDVIDEO_DETECTED	5.5.7.8
NERO_PHASE_DVDVIDEO_REALLOC_STARTED	5.5.7.8
NERO_PHASE_DVDVIDEO_REALLOC_COMPLETED	5.5.7.8
NERO_PHASE_CONTINUE_FORMATTING	5.5.8.0
NERO_PHASE_SEAMLESSLINK_ACTIVATED	5.5.8.2
NERO_PHASE_FORMATTING_SUCCESSFUL	5.5.8.2
NERO_PHASE_DVDVIDEO_REALLOC_NOTNEEDED	5.5.9.3
NERO_PHASE_DVDVIDEO_REALLOC_FAILED	5.5.9.3

7.1.30. NERO_IO

```
typedef struct tag_NERO_IO
{
  void  *nioUserData;
  NERO_IO_CALLBACK  nioIOCallback;
  NERO_STATUS_CALLBACK  nioEOFCallback;
  NERO_STATUS_CALLBACK  nioErrorCallback;
} NERO_IO;
```

Description of structure members	
nioUserData	Provide the this-pointer here.
nioIOCallback	See declaration of NERO_IO_CALLBACK.
nioEOFCallback	Shall return TRUE if further IO calls will always fail to transfer any data, i.e. EOF reached.
nioErrorCallback	Shall return TRUE if an error occurred during an IO call.

Identifier	Introduced in *NeroAPI* version
NERO_IO	5.0.3.9

7.1.31. NERO_ISO_ITEM

```
typedef struct tag_NERO_ISO_ITEM
{
  char fileName[256];
  BOOL isDirectory;
  BOOL isReference;
  char sourceFilePath[256];
  struct tag_NERO_ISO_ITEM *subDirFirstItem;
  struct tag_NERO_ISO_ITEM *nextItem;
  void *userData;
  long dataStartSec;
  __int64 dataLength;
  struct tm entryTime;
  int itemSize;
  struct CImportInfo *importinfo;
} NERO_ISO_ITEM;
```

Description of structure members	
fileName	File name on the burnt CD
isDirectory	Is this item a directory?
isReference	Is this item a reference to a file/directory of a previous session.
sourceFilePath	Path to the file, including file name (ignored for a directory). When recording RockRidge, you can set the name of a directory to be used for retrieving rockridge informations here.
subDirFirstItem	Point on the first item of the sub directory if the item is a directory. Can be NULL if the directory is empty. (ignored for a file)
nextItem	Next item in the current directory
userData	Can be used to store additional information
dataStartSec	Used to reference a file from a previous session
dataLength	Used to reference a file from a previous session
entryTime	Used to reference a file from a previous session
itemSize	Size of the structure
importinfo	Optional pointer to an object with import information.

Identifier	Introduced in *NeroAPI* version
NERO_ISO_ITEM	5.0.3.9
itemSize	5.5.0.6
importinfo	5.5.0.6 5.5.7.5: “rockridge” is renamed to “importinfo”

7.1.32. NERO_AUDIO_ITEM_HANDLE

```
typedef void * NERO_AUDIO_ITEM_HANDLE;
```

Identifier	Introduced in *NeroAPI* version
NERO_AUDIO_ITEM_HANDLE	5.5.9.14

7.1.33. NERO_AUDIO_ITEM_INFO

```
typedef struct tagNERO_AUDIO_ITEM_INFO
{
  NERO_AUDIO_ITEM_HANDLE     naiiAudioItem;
  const char                 *naiiFileName;
} NERO_AUDIO_ITEM_INFO;
```

Identifier	Introduced in *NeroAPI* version
NERO_AUDIO_ITEM_INFO	5.5.9.14

7.1.34. NERO_DATA_EXCHANGE_TYPE

```
typedef enum
{
  NERO_ET_FILE,
  NERO_ET_IO_CALLBACK,
  NERO_ET_MP3,
  NERO_ET_FILE_RAW,
  NERO_ET_AUDIO_FILE
} NERO_DATA_EXCHANGE_TYPE;
```

Description of enumerators	
NERO_ET_FILE	Read/write to/from WAV file.
NERO_ET_IO_CALLBACK	Exchange data with application directly.
NERO_ET_MP3	Read from MP3 file (not for DAE).
NERO_ET_WMA	Read from MS audio file (not for DAE).
NERO_ET_FILE_RAW	For a Freestyle compilation, this and NERO_ET_IO_CALLBACK are the only types allowed at the moment. It will expect files to be in the format as to be written to the disc. This exchange type is valid for freestyle compilations only.
NERO_ET_AUDIO_FILE	Audio file created with the plugin manager.

Identifier	Introduced in *NeroAPI* version
NERO_DATA_EXCHANGE_TYPE	5.0.3.9
NERO_ET_AUDIO_FILE	5.5.9.8

7.1.35. NERO_TRACKMODE_TYPE

```
typedef enum
{
  NERO_TRACKMODE_MODE1,
  NERO_TRACKMODE_MODE2_FORM1,
  NERO_TRACKMODE_AUDIO
} NERO_TRACKMODE_TYPE;
```

Description of enumerators	
NERO_TRACKMODE_MODE1	2048 Bytes per sector data track
NERO_TRACKMODE_MODE2_ FORM1	2048 Bytes per sector, used for multisession
NERO_TRACKMODE_AUDIO	2352 Bytes per sector, standard audio track

Identifier	Introduced in *NeroAPI* version
NERO_TRACKMODE_TYPE	5.0.3.9

7.1.36. NERO_FREESTYLE_TRACK

```
typedef struct tag_NERO_FREESTYLE_TRACK
{
  DWORD nftStructureSize;
  DWORD nftPauseInBlksBeforeThisTrack;
  DWORD nftNumIndexPositions;
  DWORD nftRelativeIndexBlkPositions[98];
  const char *nftTitle, *nftArtist;
  NERO_DATA_EXCHANGE nftSourceDataExchg;
  DWORD nftLengthInBlocks;
  NERO_TRACKMODE_TYPE nftTracktype;
} NERO_FREESTYLE_TRACK;
```

Description of structure members	
nftStructureSize	size of this structure, to ensure binary compatibility
nftPauseInBlksBeforeThisTrack	
nftNumIndexPositions	
nftRelativeIndexBlkPositions[98]	offsets between one index position and the next one
nftTitle	set to NULL if unknown or to be taken from source
nftArtist	set to NULL if unknown or to be taken from source
nftSourceDataExchg	source for raw track data
nftLengthInBlocks	only used for NERO_IO_CALLBACK
nftTracktype	specifies track type to be written

Identifier	Introduced in *NeroAPI* version
NERO_FREESTYLE_TRACK	5.0.3.9

7.1.37. NERO_DATA_EXCHANGE

Use PCM, 44.1kHz, Stereo (left channel first), 16 bits per channel, Little Endian Word (LSB first), when exchanging data with the NeroAPI.

```
typedef struct tag_NERO_DATA_EXCHANGE
{
  NERO_DATA_EXCHANGE_TYPE ndeType;
  union
  {
    char      ndeFileName[256];
    NERO_IO   ndeIO;
    NERO_AUDIO_ITEM_INFO     ndeAudioItemInfo;
  } ndeData;
} NERO_DATA_EXCHANGE;
```

Description of structure members	
ndeFileName	NERO_WAV_FILE
ndeIO	NERO_IO/EOF/ERROR_CALLBACK

Identifier	Introduced in *NeroAPI* version
NERO_DATA_EXCHANGE	5.0.3.9
ndeAudioItemInfo	5.5.9.8

7.1.38. NERO_AUDIO_TRACK

```
typedef struct tag_NERO_AUDIO_TRACK
{
  DWORD natPauseInBlksBeforeThisTrack;
  DWORD natNumIndexPositions;
  DWORD natRelativeIndexBlkPositions[98];
  const char *natTitle, *natArtist;
  NERO_DATA_EXCHANGE natSourceDataExchg;
  DWORD natLengthInBlocks;
  BOOL  natIndex0ContainsData;
  DWORD natReserved[31];
} NERO_AUDIO_TRACK;
```

Description of structure members	
natPauseInBlksBeforeThisTrack	
natNumIndexPositions	
natRelativeIndexBlkPositions	Offsets between one index position and the next one.
natTitle	Set to NULL if unknown or to be taken from source.
natArtist	Set to NULL if unknown or to be taken from source.
natSourceDataExchg	
natLengthInBlocks	Only used for NERO_IO_CALLBACK.
natReserved	Should be zero.

Description of structure members	
natIndex0ContainsData	TRUE, if audio data shall be written into index 0. Data for index 0 must be provided. This can be used to prevent silent pauses between tracks.

Identifier	Introduced in *NeroAPI* version
NERO_AUDIO_TRACK	5.0.3.9
natIndex0ContainsData	5.5.9.8
natReserved	5.5.9.8: Size decreased from 32 to 31

7.1.39. NERO_VIDEO_ITEM_TYPE

```
typedef enum
{
  NERO_MPEG_ITEM,
  NERO_JPEG_ITEM,
  NERO_NONENCODED_VIDEO_ITEM,
  NERO_DIB_ITEM
} NERO_VIDEO_ITEM_TYPE;
```

Description of enumerators	
NERO_MPEG_ITEM	Item is of MPEG type.
NERO_JPEG_ITEM	Item is of JPEG type.
NERO_NONENCODED_VIDEO_ITEM	The source file name will be an AVI file which will be encoded into MPG by NeroAPI.
NERO_DIB_ITEM	The source is a DIB picture. Informations about it must be given in nviData.nviDIB.

Identifier	Introduced in *NeroAPI* version
NERO_VIDEO_ITEM_TYPE	5.0.3.9
NERO_NONENCODED_VIDEO_ITEM	5.5.7.8
NERO_DIB_ITEM	5.5.7.6

7.1.40. NERO_VIDEO_ITEM

```
typedef struct tag_NERO_VIDEO_ITEM
{
  DWORD nviPauseAfterItem;
  char nviSourceFileName[236];
  union
  {
      struct
      {
        BITMAPINFO *pDIB;
        size_t size;
        double pixelRatio;
```

```
      } nviDIB;
  } nviData;

  DWORD reserved;
  NERO_VIDEO_ITEM_TYPE nviItemType;
} NERO_VIDEO_ITEM;
```

Description of structure members	
nviPauseAfterItem	
nviSourceFileName	MPG or JPG or AVI file name.
pDIB	Points to the DIB header followed by data.
size	DIB size.
pixelRatio	Pixel ratio of the given picture (height/width). 1 means square pixels.
nviItemType	Callback functions can only be used for MPG files.

Identifier	Introduced in *NeroAPI* version
NERO_VIDEO_ITEM	5.0.3.9
nviData	5.5.7.6

7.1.41. NERO_CD_FORMAT

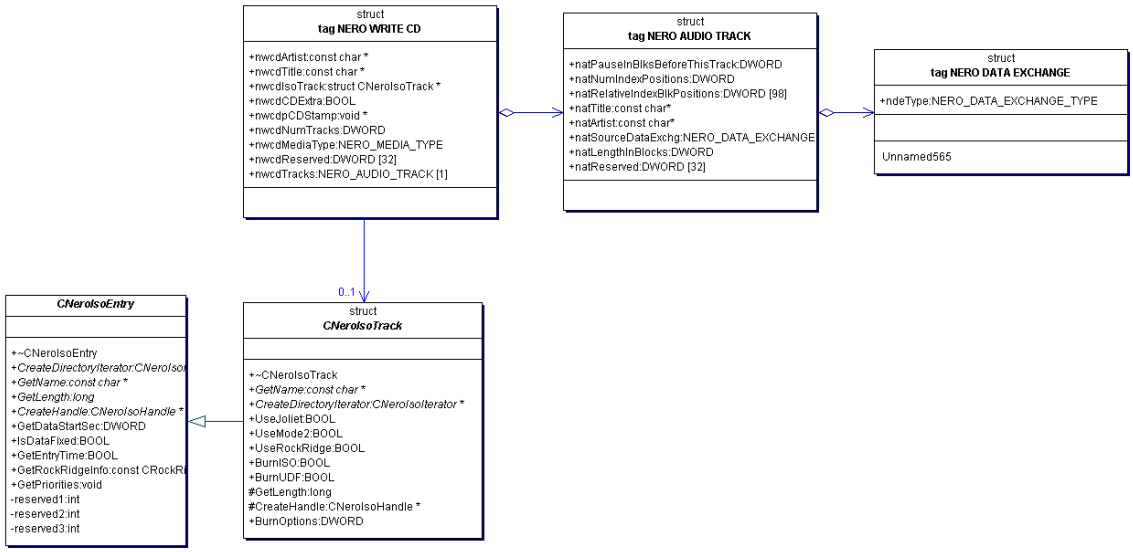
```
typedef enum
{
  NERO_ISO_AUDIO_MEDIA              =0,
  NERO_VIDEO_CD                     =1,
  NERO_BURN_IMAGE_MEDIA             =2,
  NERO_FREESTYLE_CD                 =3,
  NERO_FILE_SYSTEM_CONTAINER_MEDIA  =4,

  NERO_ISO_AUDIO_CD                 =0,
  NERO_BURN_IMAGE_CD                =2
} NERO_CD_FORMAT;
```

Description of enumerators	
NERO_ISO_AUDIO_MEDIA	Burn either a CD or a DVD, depending on the nwcdMediaType member.
NERO_VIDEO_CD	
NERO_BURN_IMAGE_MEDIA	Burn either a CD or a DVD from an image.
NERO_FREESTYLE_CD	For a Freestyle compilation.
NERO_FILE_SYSTEM_ CONTAINER_MEDIA	Burn an IfileSystemDescContainer.
NERO_ISO_AUDIO_CD	Audio or ISO CD. Available only for compatibility reasons.
NERO_BURN_IMAGE_CD	CD Type determined by content of CD Image. Available only for compatibility reasons.

Identifier	Introduced in *NeroAPI* version
NERO_CD_FORMAT	5.0.3.9
NERO_FILE_SYSTEM_CONTAINER_MEDIA	5.5.6.0

7.1.42. NERO_WRITE_CD




```
typedef struct tag_NERO_WRITE_CD
{
  const char *nwcdArtist;
  const char *nwcdTitle;
  struct CNeroIsoTrack *nwcdIsoTrack;
  BOOL nwcdCDExtra;
  void *nwcdpCDStamp;
  DWORD nwcdNumTracks;
  NERO_MEDIA_TYPE nwcdMediaType;
  BOOL nwcdAudioMaster;
  DWORD nwcdReserved[31];
  NERO_AUDIO_TRACK nwcdTracks[1];
} NERO_WRITE_CD;
```

Description of structure members	
nwcdArtist	May be NULL.
nwcdTitle	May be NULL.
nwcdIsoTrack	If not NULL, then the disc will have an ISO track - please refer to "NeroIsoTrack.h".
nwCDExtra	If TRUE and nwcdIsoTrack not NULL, then the resulting CD will have audio in the first session and the data track in the second, however, currently the NeroAPI does not add any of

Description of structure members	
	the special CD Extra files to the data track.
nwcdpCDStamp	Point on a CDStamp object if a particular CD is requested, otherwise NULL.
nwcdNumTracks	Number of Tracks.
nwcdMediaType	Media on which the data should be written.
nwcdAudioMaster	Create an Audio Master CD (if the recorder supports it).
nwcdReserved[31]	
nwcdTracks	See declaration of NERO_AUDIO_TRACK.

Identifier	Introduced in *NeroAPI* version
NERO_WRITE_CD	5.0.3.9
nwcdMediaType	5.5.4.3

7.1.43. NERO_WRITE_IMAGE

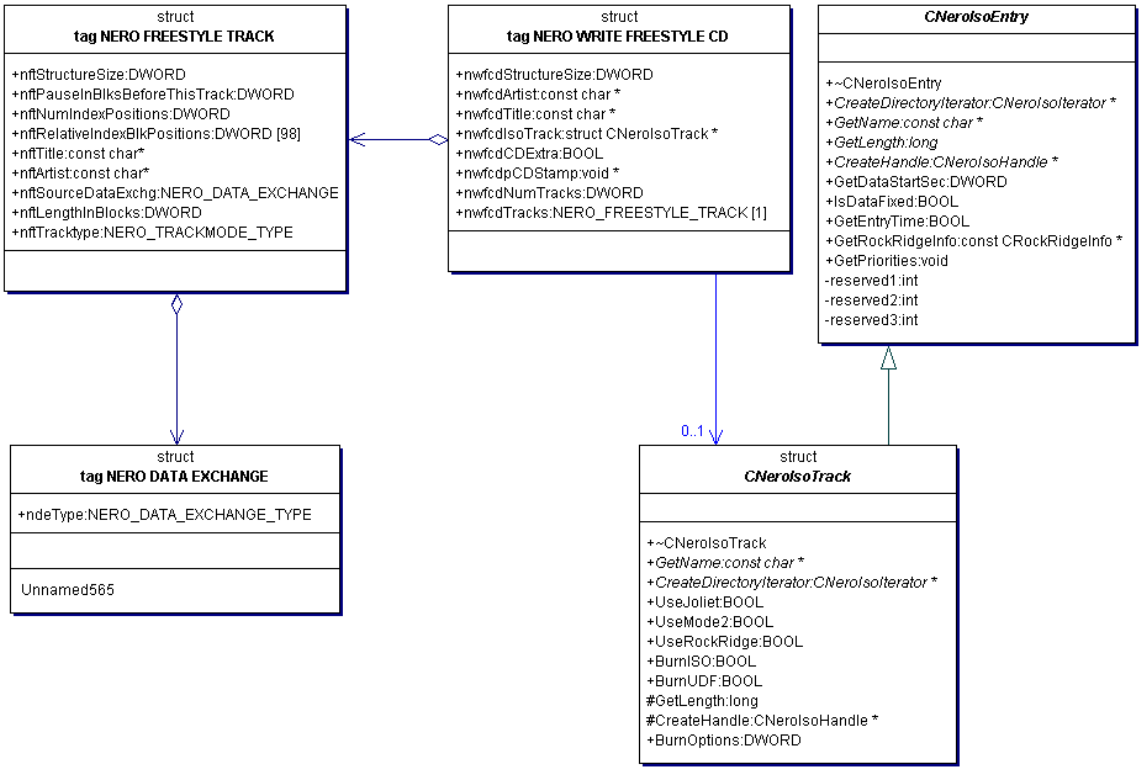
Used when burning an image.
nwiImageFileName contains the name of the image file to burn. Supported formats are NRG, ISO and CUE.

```
typedef struct tag_NERO_WRITE_IMAGE
{
  char nwiImageFileName[256];
} NERO_WRITE_IMAGE;
```

Identifier	Introduced in *NeroAPI* version
NERO_WRITE_IMAGE	5.0.3.9
nwiImageFileName	5.5.6.8: ISO and CUE possible

7.1.44. NERO_WRITE_FREESTYLE_CD

This structure will allow you to write any type of CD Layout, e.g. containing a raw data track at the beginning of the disc instead of a self-made ISO/UDF filesystem. This is good for writing .iso images as can be downloaded everywhere on the net.




```
typedef struct
{
  DWORD nwfcdStructureSize;
  const char *nwfcdArtist;
  const char *nwfcdTitle;
  struct CNeroIsoTrack *nwfcdIsoTrack;
  BOOL nwfcdCDExtra;
  void *nwfcdpCDStamp;
  DWORD nwfcdNumTracks;
  DWORD nwfcdBurnOptions;
#ifdef __cplusplus
FileSystemContent::
#else //__cplusplus
struct
#endif//__cplusplus
  IFileSystemDescContainer *nwfcdFSContainer
  NERO_MEDIA_TYPE nwfcdMediaType;
  DWORD nwfcdReserved[32];
  NERO_FREESTYLE_TRACK nwfcdTracks[1];
```

```
} NERO_WRITE_FREESTYLE_CD;
```

Description of structure members	
nwfcdStructureSize	Fill this with sizeof(NERO_WRITE_FREESTYLE_CD)
nwfcdArtist	may be NULL
nwfcdTitle	may be NULL
nwfcdIsoTrack	If not NULL, then the disc will have an ISO track - please refer to the “ISO Track Classes” description.
nwfcdCDExtra	If TRUE and nwfcdIsoTrack not NULL, then the resulting CD will have audio in the first session and the data track in the second, however, currently the NeroAPI does not add any of the special CD Extra files to the data track.
	Point to a CDStamp object if a particular CD is requested, otherwise NULL
nwfcdNumTracks	Number of tracks.
nwfcdBurnOptions	Combination of NCITEF flags.
nwfcdFSContainer	If not NULL, then the disc will have an ISO track described by this container. nwfcdIsoTrack must be NULL, otherwise the container will be ignored.
nwfcdMediaType	Media on which the data should be written.
nwfcdTracks[1]	List of NERO_FREESTYLE_TRACKs.
nwfcdReserved[32]	Should be zero.

Identifier	Introduced in *NeroAPI* version
NERO_WRITE_FREESTYLE_CD	5.0.3.9
nwfcdBurnOptions	5.5.9.1
nwfcdFSContainer	5.5.9.1
nwfcdMediaType	5.5.9.1
nwfcdReserved	5.5.9.1

7.1.45. NEROAPI_BURN_ERROR

```
typedef enum
{
      NEROAPI_BURN_OK=0,
      NEROAPI_BURN_UNKNOWN_CD_FORMAT,
      NEROAPI_BURN_INVALID_DRIVE,
      NEROAPI_BURN_FAILED,
      NEROAPI_BURN_FUNCTION_NOT_ALLOWED,
      NEROAPI_BURN_DRIVE_NOT_ALLOWED,
      NEROAPI_BURN_USER_ABORT
} NEROAPI_BURN_ERROR;
```

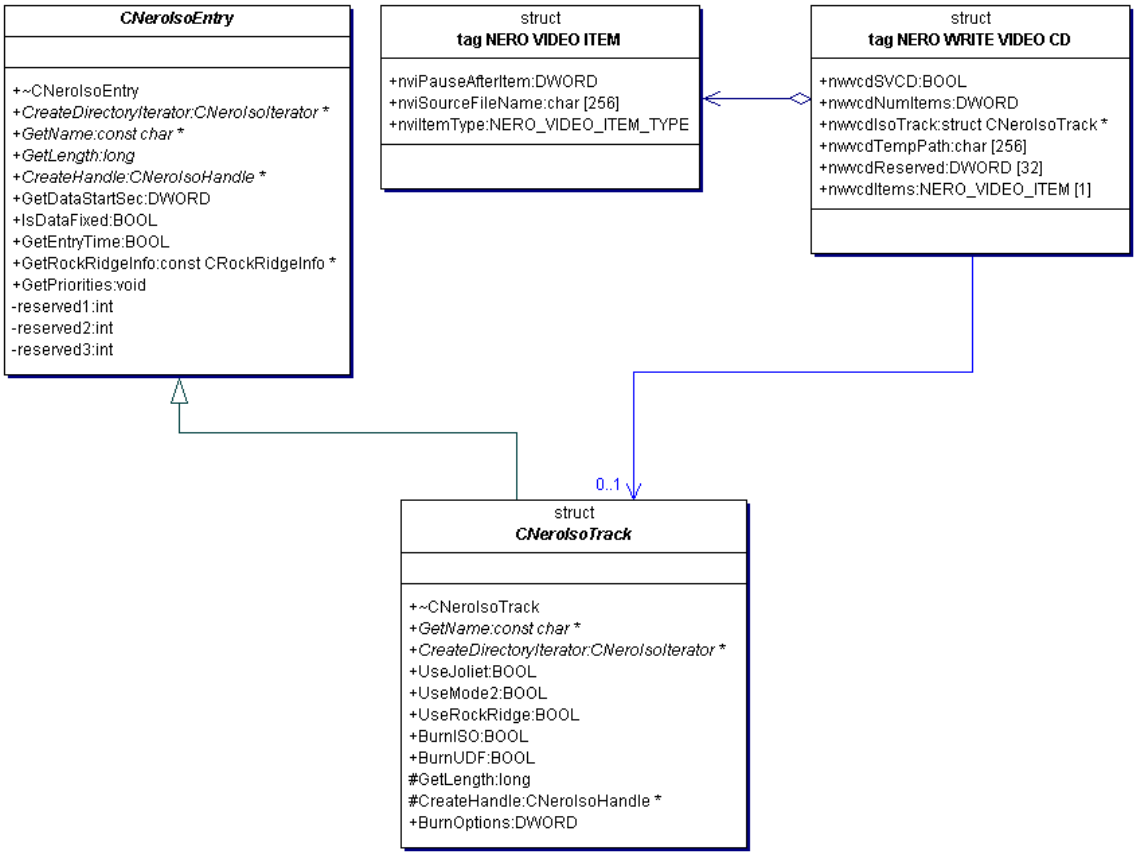
Identifier	Introduced in *NeroAPI* version
NEROAPI_BURN_ERROR	5.0.3.9

7.1.46. NeroCITEArgs

```
typedef struct {
      int size;
      NERO_ISO_ITEM *firstRootItem;
      NERO_ISO_ITEM *firstRootItem_wrapper;
      const char    *name;
      DWORD          dwBurnOptions;
} NeroCITEArgs;
```

Identifier	Introduced in *NeroAPI* version
NeroCITEArgs	5.5.9.0

7.1.47. NERO_WRITE_VIDEO_CD




```
typedef struct tag_NERO_WRITE_VIDEO_CD
{
  BOOL  nwvcdSVCD;
  DWORD  nwvcdNumItems;
  struct CNeroIsoTrack  *nwvcdIsoTrack;
  char nwvcdTempPath[256];
#ifdef __cplusplus
```

```
  VCDEngine::IVCDFSContentGenerator* (*nwvcdCustomVCDEngine)
      (VCDEngine::IVCDMediaDescription *desc,
      FileSystemContent::IFileSystemDescContainer *pFSDC);
#else
  void *nwvcdCustomVCDEngine;
#endif
  DWORD nwvcdReserved[31];
  NERO_VIDEO_ITEM nwvcdItems[1];
} NERO_WRITE_VIDEO_CD;
```

Description of structure members	
nvcdSVCD	If TRUE, write a SVCD.
nwvcdNumItems	Number of Video/Super Video Items.
nwvcdIsoTrack	Pointer to an ISO Track.
nwvcdTempPath	where the encoded files will be temporary stored.
nwvcdCustomVCDEngine	
nwvcdReserved	Should be zero.
nwvcdItems	List of Video/Super Video Items.

Identifier	Introduced in *NeroAPI* version
NERO_WRITE_VIDEO_CD	5.0.3.9
nwvcdTempPath	5.5.5.3
nwvcdCustomVCDEngine	5.5.7.6

7.1.48. NERO_WRITE_FILE_SYSTEM_CONTENT

```
typedef struct tag_NERO_WRITE_FILE_SYSTEM_CONTAINER
{
      DWORD nwfscSize;
#ifdef __cplusplus
FileSystemContent::
#else
struct
#endif
    IFileSystemDescContainer *nwfscFSContainer;
      NERO_MEDIA_TYPE   nwfscMediaType;
      DWORD nwfscBurnOptions;
      DWORD nwfscReserved[32];
} NERO_WRITE_FILE_SYSTEM_CONTENT;
```

Description of structure members	
nwfscSize	fill this with sizeof(NERO_WRITE_FILE_SYSTEM_CONTENT)
nwfscFSContainer	
nwfscMediaType	Media on which the data should be written
nwfscBurnOptions	Combination of NCITEF flags
nwfscReserved	Should be zero

Identifier	Introduced in *NeroAPI* version
NERO_WRITE_FILE_SYSTEM_CONTENT	5.5.6.0

7.1.49. NERO_CONFIG_RESULT

```
typedef enum
{
  NCR_CANNOT_CONFIGURE,
  NCR_CHANGED,
  NCR_NOT_CHANGED
} NERO_CONFIG_RESULT;
```

Identifier	Introduced in *NeroAPI* version
NERO_CONFIG_RESULT	5.5.9.8

7.1.50. NERO_AUDIO_FORMAT_INFO

```
typedef struct tagNERO_AUDIO_FORMAT_INFO
{
      char  nafiDescription[256],
            nafiExtList[256];
      BOOL  nafiTgt,
            nafiConfigurable;
}     NERO_AUDIO_FORMAT_INFO;
```

Description of structure members	
nafiDescription[256]	A description, e.g. "RIFF PCM WAV format".
nafiExtList[256];	A list, e.g. "wav,wave,riff"
nafiTgt	Contains TRUE if this is a target plugin.
nafiConfigurable	ConfigureItem will fail on items of this type if this member equals to false.

Identifier	Introduced in *NeroAPI* version
NERO_AUDIO_FORMAT_INFO	5.5.9.8

7.2. Functions

7.2.1. NeroSetExpectedAPIVersion();

```
NEROAPI_API void NADLL_ATTR NeroSetExpectedAPIVersion(DWORD);
```

Using this function, an application can tell *NeroAPI* for which version of *NeroAPI* it was designed to work. *NeroAPI* then tries to behave like this version as much as possible. This ensures the binary compatibility with future versions of *NeroAPI*. If this function is not called, *NeroAPI* will behave as *NeroAPI 5.0.3.9*. If your application uses the NeroAPIGlue, this function will be called automatically.

Note: This function is obsolete since NeroAPI 5.5.9.9. Use NeroSetExpectedAPIVersionEx instead!

Identifier	Introduced in *NeroAPI* version
NeroSetExpectedAPIVersion	5.0.3.9

7.2.2. NeroSetExpectedAPIVersionEx()

```
NEROAPI_API BOOL NADLL_ATTR NeroSetExpectedAPIVersionEx( WORD majhi
                                                         ,WORD majlo
                                                         ,WORD minhi
                                                         ,WORD minlo
                                                         ,void reserved);
```

Using this function, an application can tell *NeroAPI* for which version of *NeroAPI* it was designed to work. *NeroAPI* then tries to behave like this version as much as possible. This ensures the binary compatibility with future versions of *NeroAPI*. If this function is not called, *NeroAPI* will behave as *NeroAPI 5.0.3.9*. If your application uses the NeroAPIGlue, this function will be called automatically.

It returns true for success. Provide NULL for the “reserved” parameter!

NeroSetExpectedAPIVersion was extended in NeroAPI 5.5.9.9 to support multiple digits.

Identifier	Introduced in *NeroAPI* version
NeroSetExpectedAPIVersionEx	5.5.9.9

7.2.3. NeroFreeMem()

```
NEROAPI_API void NADLL_ATTR NeroFreeMem (void *pMem);
```

The *NeroAPI* never uses static memory. Instead, memory is allocated dynamically on behalf of the application, e.g. for strings. This memory has to be freed with this function. Passing NULL is allowed.

Identifier	Introduced in *NeroAPI* version
NeroFreeMem	5.0.3.9

7.2.4. NeroGetLastError()

```
NEROAPI_API char * NADLL_ATTR NeroGetLastError ();
```

If an error occurred, NeroGetLastError() will return additional information.

Identifier	Introduced in *NeroAPI* version
NeroGetLastError	5.0.3.9

7.2.5. NeroGetErrorLog()

```
NEROAPI_API char * NADLL_ATTR NeroGetErrorLog ();
```

All functions returning a DWORD will return 0 for success and a error number otherwise. These error numbers are opaque and neither can nor should be interpreted by the application. Instead, localized strings are provided for errors and informative displays. The NeroAPI keeps a log of such informative messages or errors.

In case of an error, NeroGetLastError() will return more information about the last error and NeroGetErrorLog() will show all recorded events.

Both functions return NULL if no error is available. Memory is allocated for the string, which has to be freed with NeroFreeMem().

Note: NeroCloseDrive() has to throw away all errors, because they might be bound to the driver. Handle errors before calling it!

Identifier	Introduced in *NeroAPI* version
NeroGetErrorLog	5.0.3.9

7.2.6. NeroClearErrors()

```
NEROAPI_API void NADLL_ATTR NeroClearErrors ();
```

Clear errors and log (done automatically for every read or write function, but can be used to avoid false memory leak warnings).

Identifier	Introduced in *NeroAPI* version
NeroClearErrors	5.0.3.9

7.2.7. NeroGetAPIVersion()

```
NEROAPI_API DWORD NADLL_ATTR NeroGetAPIVersion(void);
```

Version management for this API: Returns 1000 for 1.0.0.0

Note: This function is obsolete since NeroAPI 5.5.9.9. Use NeroGetAPIVersionEx instead!

Identifier	Introduced in *NeroAPI* version
NeroGetAPIVersion	5.0.3.9

7.2.8. NeroGetAPIVersionEx()

```
NEROAPI_API BOOL NADLL_ATTR NeroGetAPIVersionEx(   WORD *majhi
                                                  ,WORD *majlo
                                                  ,WORD *minhi
                                                  ,WORD *minlo
                                                  ,void *reserved);
```

Fills the pointed numbers (major version high and low, minor version high and low) with the version number and returns true for success. The NeroGetAPIVersion function was extended in NeroAPI 5.5.9.9 to support multiple digits. Provide NULL for the “reserved” parameter!

Identifier	Introduced in *NeroAPI* version
NeroGetAPIVersionEx	5.5.9.9

7.2.9. NeroInit()

```
NEROAPI_API NEROAPI_INIT_ERROR NADLL_ATTR NeroInit (const NERO_SETTINGS
*pNeroSettings, const char *reserved);
```

Initialize the DLL. Must be successful before any of the remaining functions can be called. Settings structure and strings it points to are not copied and function callbacks must be available all the time.
Provide NULL for the “reserved” parameter!

Make sure to keep **all** the data including the strings valid as long as you are using *NeroAPI*, since Nero will only store a pointer to the NERO_SETTINGS structure, not make a copy.

Identifier	Introduced in *NeroAPI* version
NeroInit	5.0.3.9

7.2.10. NeroSetOption()

```
NEROAPI_API int NADLL_ATTR NeroSetOption(NEROAPI_OPTION option,void
*value);
```

Used to set global Nero options such as the name of the message text file.

Identifier	Introduced in *NeroAPI* version
NeroSetOption	5.0.3.9

7.2.11. NeroDone()

```
NEROAPI_API void NADLL_ATTR NeroDone ();
```

Call this function before closing the DLL. This is necessary because some clean-up actions like stopping threads cannot be done in the close function of the DLL.

Identifier	Introduced in *NeroAPI* version
NeroDone	5.0.3.9

7.2.12. NeroGetAvailableDrivesEx()

```
NEROAPI_API NERO_SCSI_DEVICE_INFOS * NADLL_ATTR
NeroGetAvailableDrivesEx(NERO_MEDIA_TYPE mediaType, void *reserved);
```

Gets a list of available WORM and CDROM devices. This list will be freed when calling NeroFreeMem(). NeroGetAvailableDrivesEx() will return NULL if errors occurred.

Identifier	Introduced in *NeroAPI* version
NeroGetAvailableDrivesEx	5.0.3.9

7.2.13. NeroGetTypeNameOfMedia()

```
EROAPI_API char *NADLL_ATTR NeroGetTypeNameOfMedia(NERO_MEDIA_SET media,
                                                   void *reserved);
```

Get a string describing the given bit field of supported media. Free with NeroFreeMem();

Identifier	Introduced in *NeroAPI* version
NeroGetTypeNameOfMedia	5.0.3.9
NeroGetTypeNameOfMedia	5.5.9.4: Changed signature and behavior. Old version: NeroGetTypeNameOfMedia (DWORD media,const char *separator); With the current version, the separator of the current system language is used. This might cause problems if old code relies on the use of a special separator.

7.2.14. NeroOpenDevice()

```
NEROAPI_API NERO_DEVICEHANDLE NADLL_ATTR NeroOpenDevice(const
NERO_SCSI_DEVICE_INFO* pDevInfo);
```

Open a device. This function returns NULL if errors have occurred.

Identifier	Introduced in *NeroAPI* version
NeroOpenDevice	5.0.3.9

7.2.15. NeroCloseDevice()

```
NEROAPI_API void NADLL_ATTR NeroCloseDevice(NERO_DEVICEHANDLE
aDeviceHandle);
```

Close a device. This function returns NULL if errors have occurred.

Identifier	Introduced in *NeroAPI* version
NeroCloseDevice	5.0.3.9

7.2.16. NeroCreateIsoItem()

```
#define NeroCreateIsoItem()  NeroCreateIsoItemOfSize(sizeof(struct
NERO_ISO_ITEM))
```

Macro for automatically filling the size_t member of NeroCreateIsoItemOfSize().

Identifier	Introduced in *NeroAPI* version
NeroCreateIsoItem	5.0.3.9

7.2.17. NeroCreateIsoItemOfSize()

```
NEROAPI_API struct NERO_ISO_ITEM * NADLL_ATTR
NeroCreateIsoItemOfSize(size_t);
```

Allocates an instance of the NERO_ISO_ITEM structure of size size_t.

Identifier	Introduced in *NeroAPI* version
NeroCreateIsoItemOfSize	5.0.3.9

7.2.18. NeroFreeIsoItem()

```
NEROAPI_API void NADLL_ATTR NeroFreeIsoItem(struct NERO_ISO_ITEM *);
```

Free memory that is used by an instance of the NERO_ISO_ITEM structure.

Identifier	Introduced in *NeroAPI* version
NeroFreeIsoItem	5.0.3.9

7.2.19. NeroCopyIsoItem

```
NEROAPI_API NERO_ISO_ITEM * NADLL_ATTR
NeroCopyIsoItem (const NERO_ISO_ITEM *iso_item);
```

Create a copy of an existing NERO_ISO_ITEM object.

This is a safe way to obtain an exact copy of NERO_ISO_ITEM objects imported from a previous session. Note that the new NERO_ISO_ITEM's extItem, userData and subDirFirstItem members are set to NULL.

Identifier	Introduced in *NeroAPI* version
NeroCopyIsoItem	5.5.9.9

7.2.20. NeroCreateIsoTrackEx()

```
NEROAPI_API struct CNeroIsoTrack * NADLL_ATTR
NeroCreateIsoTrackEx(struct NERO_ISO_ITEM *root, const char *name, DWORD
flags);
```

Create an ISO track from a NERO_ISO_ITEM tree.

Description of parameters

Description of parameters		
Root	First item of the root directory.	
Name	Name of the CD	
Flags	available constants :	
	NCITEF_USE_JOLIET	Create a Joliet Track
	NCITEF_USE_MODE2	Create a Mode 2 Track
	NCITEF_USE_ROCKRIDGE	Create a RockRidge Track
	NCITEF_CREATE_ISO_FS	Create an ISO File System Track
	NCITEF_CREATE_UDF_FS	Create an Universal Disk Format File System Track
	NCITEF_CREATE_HFS_FS	Not yet available.
	NCITEF_DVDVIDEO_ REALLOC	Perform reallocation of files in the VIDEO_TS directory.
	NCITEF_USE_STRUCT	'name' points to an argument struct instead of name. If set, 'root' and other 'flags' are ignored.

Identifier	Introduced in *NeroAPI* version
NeroCreateIsoTrackEx	5.0.3.9
NCITEF_DVDVIDEO_REALLOC	5.5.7.8
NCITEF_USE_STRUCT	5.5.9.0

7.2.21. NeroFreeIsoTrack()

```
NEROAPI_API void NADLL_ATTR NeroFreeIsoTrack(struct CNeroIsoTrack
*track);
```

Free an ISO track previously allocated with NeroCreateIsoTrackEx().

Identifier	Introduced in *NeroAPI* version
NeroFreeIsoTrack	5.0.3.9

7.2.22. NeroImportIsoTrackEx()

```
NEROAPI_API NERO_ISO_ITEM *NeroImportIsoTrackEx
(
  NERO_DEVICEHANDLE pRecorder,
  DWORD trackNumber,
  void **ppCDStamp,
  DWORD flags
);
```

Create a NERO_ISO_ITEM tree from an already existing ISO track in order to create a new session with reference to files from older sessions.

Description of parameters		
pRecorder	First item of the root directory.	
trackNumber	Name of the CD	
ppCDStamp	*ppCDStamp will be filled with a pointer on a CDStamp object which will have to be freed later	
flags	available constants :	
	NIITEF_IMPORT_ROCKRIDGE	Will be ignored, RockRidge is now always imported if present.
	NIITEF_IMPORT_ISO_ONLY	
	NIITEF_PREFER_ROCKRIDGE	Will be ignored.
	NIITEF_IMPORT_UDF	Import UDF Session.

Identifier	Introduced in *NeroAPI* version
NeroImportIsoTrackEx	5.0.3.9

7.2.23. NeroFreeCDStamp()

```
NEROAPI_API void NADLL_ATTR NeroFreeCDStamp(void *pCDStamp);
```

Free a CD stamp allocated by NeroImportIsoTrackEx().

Identifier	Introduced in *NeroAPI* version
NeroFreeCDStamp	5.0.3.9

7.2.24. NeroGetCDInfo()

```
NEROAPI_API NERO_CD_INFO * NADLL_ATTR NeroGetCDInfo
(
  NERO_DEVICEHANDLE aDeviceHandle,
  DWORD dwFlags
);
```

Retrieve a pointer to a NERO_CD_INFO structure for the specified device. The allocated memory for the structure has to be freed by using NeroFreeMem(). NULL will be returned if an error occurred.

Description of parameters		
aDeviceHandle	Device Handle	
dwFlags	available constants :	
	NGCDI_READ_CD_TEXT	(1<<0)
	NGCDI_READ_ISRC	(1<<1) International Standard Recording Code

Identifier	Introduced in *NeroAPI* version
NeroGetCDInfo	5.0.3.9
NGCDI_READ_ISRC	5.5.8.4

7.2.25. NeroGetAvailableSpeeds

```
NEROAPI_API NERO_SPEED_INFOS * NADLL_ATTR
NeroGetAvailableSpeeds  (NERO_DEVICEHANDLE aDeviceHandle,
                         NERO_ACCESSTYPE accessType,
                         NERO_MEDIA_TYPE mediaType,
                         void *reserved);
```

Get available write speeds depending on medium type, free with NeroFreeMem(). Returns NULL for error.

Identifier	Introduced in *NeroAPI* version
NeroGetAvailableSpeeds	5.5.9.10

7.2.26. NeroBurn()

```
NEROAPI_BURN_ERROR NADLL_ATTR NeroBurn
(
 NERO_DEVICEHANDLE  aDeviceHandle,
 NERO_CD_FORMAT  CDFormat,
 const void*  pWriteCD,
 DWORD  dwFlags,
 DWORD  dwSpeed,
 NERO_PROGRESS*  pNeroProgress
);
```

Burns a CD-ROM.

Description of parameters		
pwriteCD	Must point on a NERO_WRITE_CD or a NERO_WRITE_VIDEO_CD structure.	
dwFlags	Some options for burning:	
	NBF_SPEED_TEST	Test speed of source first.
	NBF_SIMULATE	Simulate writing before actually writing.
	NBF_WRITE	Really write at the end.
	NBF_DAO	Write in DAO.
	NBF_CLOSE_SESSION	Only close the session and not the whole disc.
	NBF_CD_TEXT	Write CD text - will be ignored if not supported by drive.

	NBF_BURN_PROOF	Present for compatibility: will enable any buffer underrun protection feature even if it is not "burn proof"
	NBF_BUF_UNDERRUN_PROT	Enable safer burn mode.
	NBF_DISABLE_ABORT	The abort callback will be called.
	NBF_DETECT_NON_EMPTY_CDRW	The DLG_NON_EMPTY_CDRW user callback will be called when trying to burn onto a non empty CDRW.
	NBF_DISABLE_EJECT	CD will not be ejected at the end of the burn process.
	NBF_VERIFY	Verify Filesystem after writing. Works for ISO only.
	NBF_SPEED_IN_KBS	Interpret the dwSpeed as KB/s instead of multiple of 150 KB/s.
	NBF_DVDP_BURN_30MM_AT_LEAST	DVD+R/RW high compability mode (at least 1GB will be written)
dwSpeed	In KB/s if NBF_SPEED_IN_KBS is present, in multiple of 150 KB/s otherwise.	

Identifier	Introduced in *NeroAPI* version
NeroBurn	5.0.3.9
NBF_DISABLE_EJECT	5.5.1.1
NBF_SPEED_IN_KBS	5.5.5.5
NBF_DVDP_BURN_30MM_AT_LEAST	5.5.8.0

7.2.27. NeroDAE()

```
int NADLL_ATTR NeroDAE
(
 NERO_DEVICEHANDLE  aDeviceHandle,
 DWORD  dwTrackStartBlk,
 DWORD  dwTrackLengthInBlks,
 const  NERO_DATA_EXCHANGE  *pDestDataExchg,
 DWORD  iSpeedInX,
 NERO_CALLBACK*  pNeroProgressCallback
);
```

Digital Audio Extraction. Aborting will not be reported by NeroGetLastError(). Incomplete target files are not deleted.

Description of parameters	
iSpeedInX	Speed of extraction, 0 means maximum speed
pNeroProgressCallback	Has to be a NERO_PROGRESS_CALLBACK.

Identifier	Introduced in *NeroAPI* version

NeroDAE	5.0.3.9

7.2.28. NeroIsDeviceReady()

```
NEROAPI_API int NADLL_ATTR NeroIsDeviceReady(NERO_DEVICEHANDLE
aDeviceHandle);
```

Returns 0 if ready or an error code if not.

Identifier	Introduced in *NeroAPI* version
NeroIsDeviceReady	5.0.3.9

7.2.29. NeroEjectLoadCD()

```
NEROAPI_API int NADLL_ATTR NeroEjectLoadCD(NERO_DEVICEHANDLE
aDeviceHandle,BOOL eject);
```

Returns 0 if successful or an error code if not. FALSE in parameter “eject” loads a CD, TRUE ejects.

Identifier	Introduced in *NeroAPI* version
NeroEjectLoadCD	5.0.3.9

7.2.30. NeroWaitForMedia

```
NEROAPI_API BOOL NADLL_ATTR
      NeroWaitForMedia (NERO_DEVICEHANDLE aDeviceHandle,
                        NERO_MEDIA_SET nms,
                        DWORD dwFlags,
                        void *pCDStamp);
```

Use the nstUserDialog callback functions to request a CD. Returns FALSE if the burn process should be aborted.

Description of parameters	
nms	Media types requested.
dwFlags	Set of NBF_ flags.
pCDStamp	Optional stamp of requested media.

Identifier	Introduced in *NeroAPI* version
NeroWaitForMedia	5.5.9.4

7.2.31. NeroGetLocalizedWaitCDTexts

```
NEROAPI_API char * NADLL_ATTR NeroGetLocalizedWaitCDTexts
(NERO_WAITCD_TYPE type);
```

Returned string must be released using NeroFreeMem(). Function may return NULL if type is out of range.

Identifier	Introduced in *NeroAPI* version
NeroGetLocalizedWaitCDTexts	5.5.9.10

7.2.32. NeroGetCDRWErasingTime ()

```
NEROAPI_API int NADLL_ATTR NeroGetCDRWErasingTime(NERO_DEVICEHANDLE
aDeviceHandle,int mode);
```

Returns estimated blanking time for loaded CD-RW in seconds.

Description of return values	
-1	No CD inserted.
-2	Recorder does not support CDRW.
-3	The inserted media is not rewriteable.

Identifier	Introduced in *NeroAPI* version
NeroGetCDRWErasingTime	5.0.3.9
Return value “–3”	5.5.7.4

7.2.33. NeroEraseCDRW()

```
NEROAPI_API int NADLL_ATTR NeroEraseCDRW(NERO_DEVICEHANDLE
aDeviceHandle,int mode);
```

Erase the loaded CD. With parameter “mode” set to 0 the function will erase the entire CD. If “mode” is set to “1” a quick erase routine will be performed.

Identifier	Introduced in *NeroAPI* version
NeroEraseCDRW	5.0.3.9

8. ISO Track Creation

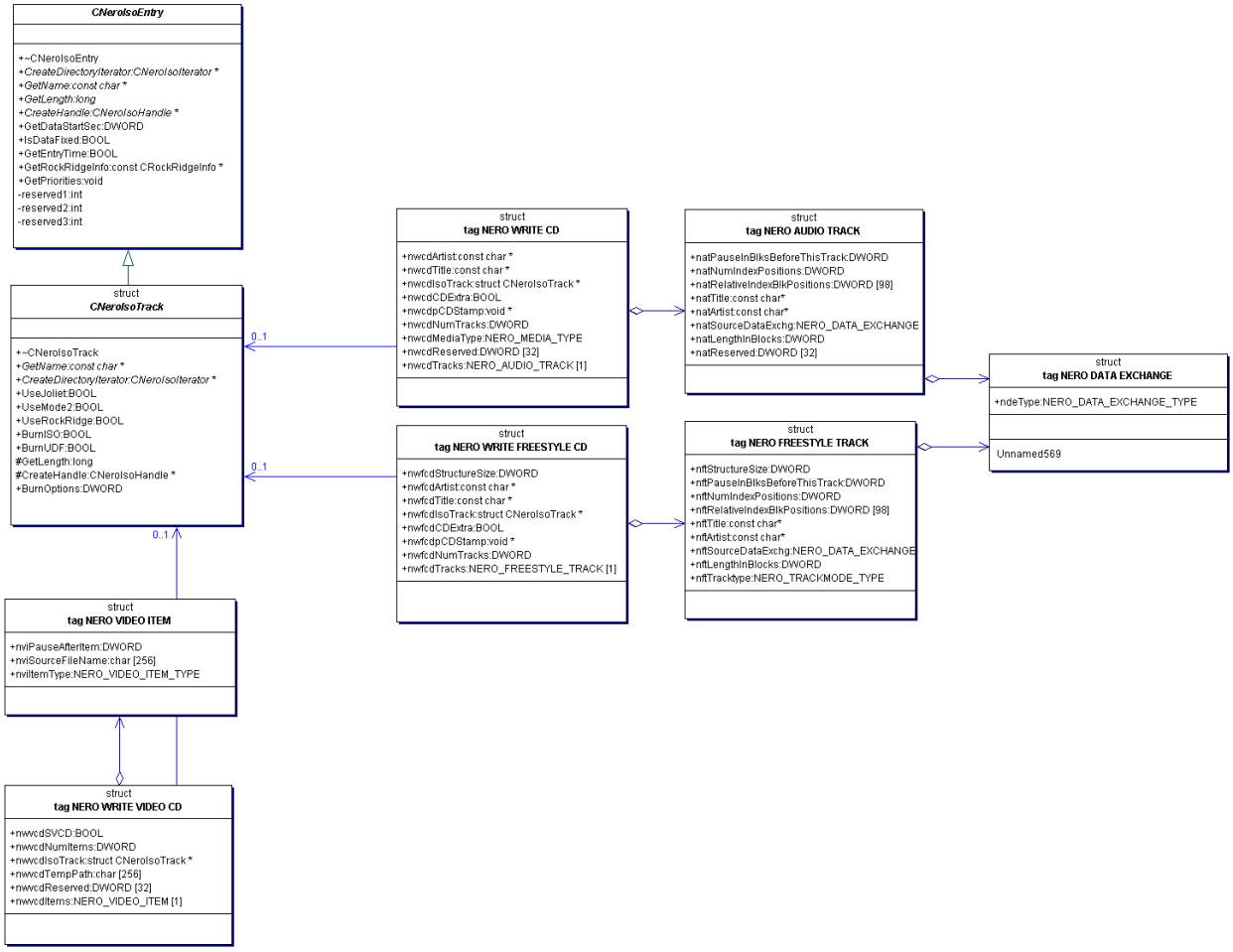
When working with the *NeroAPI*, there are three ways for creating ISO tracks:

1. By creating classes derived from those declared in NeroIsoTrack.h, and putting a pointer to an instance into the NERO_WRITE_CD structure.
2. By creating a tree of NERO_ISO_ITEMs, and creating an CNeroIsoTrack object, using the NeroCreateIsoTrackEx function
3. By creating an instance of an IFileSystemDescContainer object, using the NeroCreateFileSystemContainer function, adding files to this object through the interfaces declared in NeroFileSystemContent.h, and then using the NERO_WRITE_FILE_SYSTEM_CONTENT structure.

These three interfaces have been created for different needs and coexist in *NeroAPI*.

9. ISO Track Classes

9.1. Overview



The following classes are used to write an ISO 9660/Joliet track. In contrast to most of the rest of the *NeroAPI*, the ISO Track interface is not written in pure C, but C++. Thus, the Nero ISO Track feature can only be used by C++ code.

9.2. class CNeroDataCallback

```
class CNeroDataCallback
{
public:
   virtual ~CNeroDataCallback () {}
   virtual DWORD IOCallback(BYTE *pBuffer, DWORD dwLen) = 0;
   virtual BOOL EOFCallback ()                          = 0;
   virtual BOOL ErrorCallback ()                        = 0;
};
```

The application has to specify the complete layout of the ISO track. The NeroAPI does not care at all where the data for the files comes from. This also means that the application has to provide access to the filename, or the data itself, when the API needs it. Data can be fed into the API directly (i.e. without intermediate files) with CNeroDataCallback.

Description of class member functions	
IOCallback()	same semantic as NERO_IO_CALLBACK in "NeroAPI.h"
EOFCallback()	same semantic as NERO_IO.nioEOFCallback
ErrorCallback()	same semantic as NERO_IO.nioErrorCallback

9.3. class CNeroIsoHandle

```
class CNeroIsoHandle
{
public:
   virtual ~CNeroIsoHandle () {}
   virtual CNeroIsoHandle * Clone ()                                = 0;
   virtual int GetFileName (char *strBuffer, UINT nBufferSize)      = 0;
   virtual CNeroDataCallback * Open ()                              = 0;
};
```

The API builds an internal representation of the complete ISO tree and uses a CNeroIsoHandle, acquired from the application for each file, to access the data later.

Description of class member functions	
Clone()	Creates a copy of the CNeroIsoHandle object.
GetFileName()	If the application wants the API to read files, it has to fill the buffer of size nBufferSize with a null-terminated string and return the length of the full name, even if the given buffer was too small. The API will try again with a larger buffer then. Return 0 in GetFileName() if you want to provide the data via a CNeroDataCallback

Open()	Return instance ready to read the data associated with this handle or NULL for error; this instance will be deleted by NeroAPI; usually only one file at once will be left open

9.4. class CNeroIsoIterator

```
class CNeroIsoEntry;
class CNeroIsoIterator
{
public:
   virtual ~CNeroIsoIterator () {}
   virtual CNeroIsoEntry * GetCurrentEntry ()      = 0;
   virtual void Next ()                            = 0;
};
```

Iterators are used to walk through directories while the API builds its internal copy of the tree. Iterators point to an entry or to NULL, if the last entry was passed, and can only be incremented.

Description of class member functions	
GetCurrentEntry()	Get pointer to current entry or NULL if last one passed; entry not deleted by API, so the iterator may point to itself and implement the required interface (as in the NeroAPI demo), or to some permanent entry.
Next()	Go to next entry.

9.5. struct CNeroIsoEntry

```
struct CImportInfo;
class CNeroIsoEntry
{
public:
   virtual ~CNeroIsoEntry () {}

   virtual CNeroIsoIterator * CreateDirectoryIterator() = 0;
   virtual const char *       GetName ()                = 0;
   virtual __int64            GetLength ()              = 0;
   virtual CNeroIsoHandle *   CreateHandle ()           = 0;
   virtual DWORD GetDataStartSec() { return 0;}
   virtual BOOL IsDataFixed() { return FALSE;}
   virtual BOOL GetEntryTime(struct tm *tm) {return FALSE;}
   virtual const CImportInfo *GetImportInfo() const
    {return NULL;};
   virtual void GetPriorities(int &iPriority,int &iDirPriority)
   {
     iPriority       =0;
     iDirPriority    =0;
   };
```

```
   virtual CNeroIsoIterator * CreateDirectoryIteratorWrapper()
                                { return NULL;    }
   virtual CNeroIsoHandle *   CreateResourceHandle ()
                                { return NULL; };
private:
   virtual int reserved1()       {return 0;}
};
```

Description of class member functions	
CreateDirectoryIterator()	NULL if no directory, otherwise an iterator to step through all child entries; iterator will be deleted by NeroAPI.
GetName()	The name for this entry; will be copied by API.
GetLength()	The size of this entry in bytes, or -1 if a directory.
CreateHandle()	Creates a handle stored by the API to open a file later, NULL for directory; handle will be deleted by NeroAPI when deleting the internal ISO tree.
GetDataStartSec()	Can be used to reference files from previous session. Not fully implemented yet.
IsDataFixed()	Can be used to reference files from previous session.
GetEntryTime()	Can be used to reference files from previous session.
GetImportInfo()	This method was formerly known as GetRockRidgeInfo. The object returned is a bit different internally now. Since it is a private structure of NeroAPI this change does not matter. No ImportInfo by default
GetPriorities()	
CreateDirectoryIteratorWrapper()	This function is equivalent to CreateDirectoryIterator() but reurns an iterator for the wraper file system of a CD, e.g. when creating HFS+ CDs with an HFS wrapper file system.
CreateResourceHandle()	See CreateHandle(). Creates rsc fork handle for HFS filesystems. Will be preferred to reading the resource fork from the file specified by GetName() if !=NULL.
reserved1()	Reserved for future use.

Identifier	Introduced in *NeroAPI* version
CreateDirectoryIteratorWrapper()	5.5.9.0
CreateHandle()	5.5.9.0

9.6. struct CNeroIsoTrack

```
struct CNeroIsoTrack : public CNeroIsoEntry
{
   friend class CNeroIsoTrackProxy5039;
public:
   ~CNeroIsoTrack () {}

   virtual const char *       GetName ()                      = 0;
   virtual CNeroIsoIterator * CreateDirectoryIterator ()      = 0;
   virtual BOOL               UseJoliet () { return TRUE; }
```

```
   virtual BOOL                UseMode2 () { return FALSE; }
   virtual BOOL                UseRockRidge () { return FALSE; }
   virtual BOOL                BurnISO() { return TRUE; }
   virtual BOOL                BurnUDF() { return FALSE; }
protected:
   virtual __int64             GetLength () { return -1; }
   virtual CNeroIsoHandle *    CreateHandle () { return NULL; }
public:
    virtual DWORD               BurnOptions()
     {
         return (UseJoliet() ? NCITEF_USE_JOLIET : 0)
                 |(UseMode2() ? NCITEF_USE_MODE2 : 0)
                 |(UseRockRidge() ? NCITEF_USE_ROCKRIDGE : 0)
                 |(BurnISO() ? NCITEF_CREATE_ISO_FS : 0)
                 |(BurnUDF() ? NCITEF_CREATE_UDF_FS : 0);
     };
   virtual CNeroIsoIterator *CreateDirectoryIteratorWrapper()
   {
         return NULL;
   }

   virtual BOOL HasWrapper(void)
   {
         return FALSE;
   }

protected:
   virtual CNeroIsoHandle *    CreateResourceHandle () { return NULL; }
};
```

An ISO track is a special directory entry.

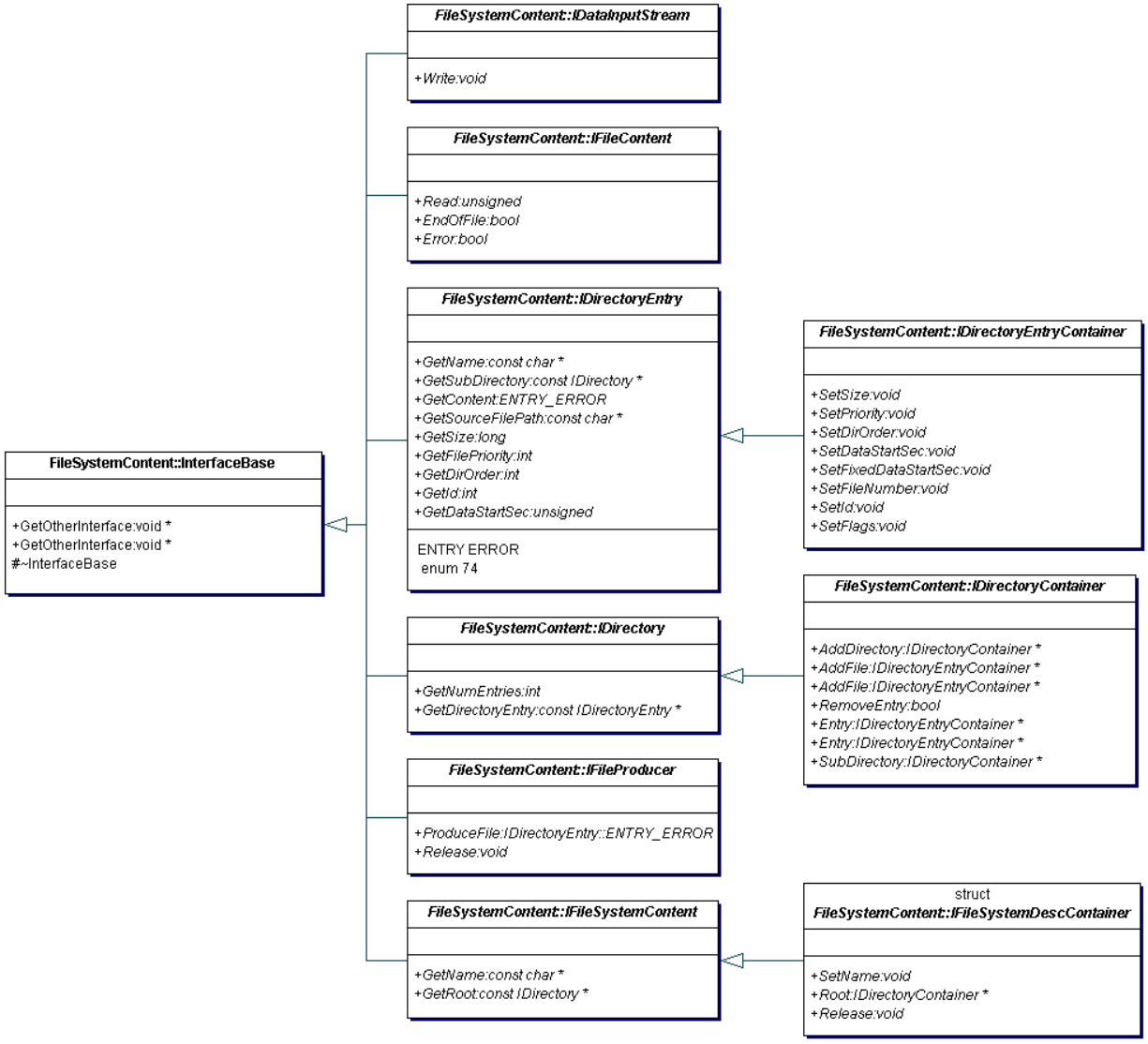
Description of class member functions	
GetName()	Essential function. ISO volume name, copied by API.
CreateDirectoryIterator()	Essential function. Iterator for root directory; will be deleted by API.
UseJoliet()	Function returns resonable default. TRUE if track shall contain Joliet names in addition to ISO.
UseMode2()	Function returns resonable default. TRUE if track shall be written as mode 2/XA.
UseRockRidge()	Function returns resonable default. RockRidge requires additional informations, so it is off by default.
BurnISO()	Function exists from NeroAPI version 5.5.0.0. TRUE if ISO should be created.
BurnUDF()	Function exists from NeroAPI version 5.5.0.0. TRUE if UDF should be created.
GetLength()	Function exists from NeroAPI version 5.5.0.0. Returns value for directory.
CreateHandle()	Return NULL, so object cannot be read.

BurnOptions()	From NeroAPI version 5.5.1.2 you can set your burn options simply by redefining this function instead of UseJoliet(), UseMode2(), UseRockRidge(), BurnISO() and BurnUDF(). See 0 NeroCreateIsoTrackEx() for the significance of the NCITEF flags.

Identifier	**Introduced in *NeroAPI* version**
CreateDirectoryIteratorWrapper()	5.5.9.0
HasWrapper()	5.5.9.0
CreateResourceHandle()	5.5.9.0

10. The FileSystemContent Interface

10.1. Overview



This is the third NeroAPI interface for preparing data CDs/DVDs. Unlike NeroIsoTrack.h, it is not much "callback based", thus most of the process will be driven by the application, making it easier to write. This interface is closely connected to the internal engine of *NeroAPI*; this improves the cooperation of *NeroAPI* and the application.

This set of classes describes the content of the file system of a disc. The application will build a file structure using the IFileSystemContent object.

During the burn process, *NeroAPI* will request the content of files using the IFileContent interface.

Use the NeroCreateFileSystemContainer function of NeroAPI.h to get an instance of an IFileSystemDescContainer object. Then, use the NERO_WRITE_FILE_SYSTEM_CONTAINER structure to burn the file structure created.

10.2. Namespace setting

To make sure we do not interfere with other classes we use the namespace "FileSystemContent" for this group of interfaces.

```
namespace FileSystemContent
```

10.3. InterfaceBase

GetOtherInterface returns a different interface for the same object. This will be used to extend the DLL interface without loosing binary compatibility. The function returns NULL if no interface with this ID was found. This is inspired from the COM QueryInterface function.
A different interface can be requested by providing an ID number or a string. Currently no other interfaces are available by default.

```
class InterfaceBase
{
public:
  virtual void *GetOtherInterface(int interfaceId)            const
  {
         return 0;
  }
  virtual void *GetOtherInterface(const char *interfaceName)  const
  {
         return 0;
  }

protected:
  virtual ~InterfaceBase() {}
};
```

10.4. File System Reading Interfaces

This first set of interfaces will be used by the burn engine to read the content of the file system.

10.4.1. IFileContent

Release will be called by the application when the object is not needed anymore

```
class IFileContent : public InterfaceBase
{
public:
      virtual unsigned Read(void *pBuffer,unsigned length) =0;
      virtual bool EndOfFile()                             =0;
      virtual bool Error()                                 =0;

      virtual void Release()                               =0;
};
```

10.4.2. IDirectoryEntry

GetName returns a file or directory name.

GetSourceFilePath returns the source file path, NULL if the file is generated.

GetId returns an Id number that can be used to find the file again later.

```
class IDirectoryEntry : public InterfaceBase
{
public:
      enum ENTRY_ERROR
      {
            ENTRY_NO_ERROR,
            SEQUENCING_ERROR,
            ERROR_NOT_A_FILE,
            NOT_AVAILABLE,
            INTERFACE_ERROR
      };

      enum
      {
            MODE2_FORM2                   =1<<0,
            FIXED_INSIDE_VOLUME_SPACE     =1<<1,
            FIXED_OUTSIDE_VOLUME_SPACE    =1<<2,
            NO_OWN_CONTENT                =1<<3
      };

      virtual const char *GetName()                   const =0;

      virtual const IDirectory *GetSubDirectory()     const =0;
      virtual ENTRY_ERROR GetContent(IFileContent **) const =0;
      virtual const char *GetSourceFilePath()         const =0;
      virtual __int64   GetSize()                     const =0;
      virtual int GetFilePriority()                   const =0;
      virtual int GetDirOrder()                       const =0;
      virtual int GetId()                             const =0;
      virtual unsigned GetDataStartSec()              const =0;
```

```
};
```

Description of enumerators	
SEQUENCING_ERROR	Indicates that the content for this file may not be requested at this moment.
ERROR_NOT_A_FILE	This entry is not a file
NOT_AVAILABLE	The content of this file cannot be requested at all.
INTERFACE_ERROR	The overriden function has tried to get an other interface for one object and has failed.
FEATURE_NOT_AVAILABLE	This feature is not available for this file system type.

Identifier	Introduced in *NeroAPI* version
FEATURE_NOT_AVAILABLE	5.5.8.2
NO_OWN_CONTENT	5.5.9.4

10.4.3. IDirectory

```
class IDirectory : public InterfaceBase
{
public:
      virtual int GetNumEntries()                              const =0;
      virtual const IDirectoryEntry *GetDirectoryEntry(int i) const =0;
};
```

10.4.4. IFileSystemContent

GetName returns the volume name.

```
class IFileSystemContent : public InterfaceBase
{
public:
      virtual const char *GetName()       const =0;
      virtual const IDirectory *GetRoot() const =0;
};
```

10.5. File System Content Creation Interfaces

This second set of interfaces will be used by the application to produce the content of the file system.

10.5.1. IDataInputStream

This interface allows the file producer to return the data.

```
class IDataInputStream : public InterfaceBase
{
public:
      virtual void Write(const void *buffer,int size)= 0;
};
```

10.5.2. IFileProducer

Produce the content of a file. This interface must be derived and its implementation must create the content of the file in the ProduceFile function.

Calling ProduceFile will automatically update the file size to the amount of data delivered by the producer.

Release will be called by the *NeroAPI* when the object is not needed anymore

```
class IFileProducer : public InterfaceBase
{
public:
      virtual IDirectoryEntry::ENTRY_ERROR ProduceFile(
              IDataInputStream *str) const = 0;

      virtual void Release()         const = 0;
};
```

10.5.3. IDirectoryEntryContainer

This interface provides the means of describing a file. Using the SetSize function, the file size can be changed after having added the entry to the directory.

If the file entry was created using an IFileProducer object, this one can be retrieved using GetOtherInterface.

SetDataStartSec sets the sector number that will be saved into the directory structure.

SetFixedDataStartSec sets the physical position of the file in the filesystem.

Setpriority and SetDirOrder can be used to re-adjust the directory priority. Priority numbers will be used in upward order: the file with smaller values first.

SetFlags enables or disables the given flag.

```
class IDirectoryEntryContainer : public IDirectoryEntry
{
public:
      enum
      {
        IID_IDirectoryEntryContainer,
        IID_IFileProducer,
        IID_IDirectoryEntryContainer2,
        IID_IDirectoryEntry2
      };
      virtual void SetSize(__int64 size)                    =0;

      virtual void SetPriority(int priority)                =0;
      virtual void SetDirOrder(int directoryPriority)       =0;
```

```
      virtual void SetDataStartSec(unsigned)                =0;
      virtual void SetFixedDataStartSec(unsigned)           =0;
      virtual void SetFileNumber(int)                       =0;
      virtual void SetId(int)                               =0;
      virtual void SetFlags(bool enable,unsigned f)         =0;
};
```

10.5.4. IDirectoryContainer

This interface represents the content of a directory. AddDirectory returns a pointer to the directory, directoryPriority specifies the position in the directory. AddFile adds a file to the directory. The fp object will be automatically deleted when the directory container will be deleted.

The filesize passed here does **not** need to be correct, it will be used by the filesystem generator to preallocate space so it must be the **maximum** space the final version of the file may need (worst-case).

Priority specifies some user-defined ordinal defining the order in which the files are being written to the disc physically (like .ifo comes before .vob).

Priorities are valid across directories. The fileentry order in a directory is defined by the directoryPriority parameter which is the primary sort criterium when arranging the files in a directory (Note that this is only true for filesystems that do not require files to be sorted in the directory, e.g. UDF).

If any of the priority specifiers is -1, the producer does not care about the priority and the *NeroAPI* will put the file where it thinks it fits. AddFile will add a file which is present in the real file system, and return NULL if a file with the same name already exists.

RemoveEntry removes an entry from the directory.

```
class IDirectoryContainer : public IDirectory
{
public:
      virtual IDirectoryContainer *AddDirectory(const char *name,
              int directoryPriority)                            =0;

      virtual IDirectoryEntryContainer *AddFile(const char *name,
              const IFileProducer *fp,__int64 size,int priority,
              int directoryPriority)                            =0;


      virtual IDirectoryEntryContainer *AddFile(const char *name,
              const char *sourcePath, int priority,
              int directoryPriority)                            =0;

      virtual bool RemoveEntry(const char *name)                =0;
```

```
      virtual IDirectoryEntryContainer *Entry(const char *name)  =0;
      virtual IDirectoryEntryContainer *Entry(int i)             =0;
      virtual IDirectoryContainer *SubDirectory(const char *name =0;
};
```

10.5.5. IFileSystemDescContainer

This interface represents the content of a file system.

SetName sets the volume name of the file system.

The Root function provides access to the root directory for changing it.

Release is called by the application when the object is not needed anymore.

```
struct IFileSystemDescContainer : public IFileSystemContent
{
      virtual void SetName(const char *)  =0;
      virtual IDirectoryContainer *Root() =0;

      virtual void Release()        const =0;
};
```

11. Media Type Formats

11.1. Audio

The *NeroAPI* requires the use of PCM (Pulse Code Modulation), 44.1kHz, Stereo (left channel first), 16 bits per channel, Little Endian Word (Least Significant Byte first).

WAV and MP3 files can also be burnt on Audio-CD by passing their path.

11.2. Video

11.2.1. SVCD Creation with *Nero*

There has been some confusion about what kind of input files are accepted by *Nero* for VCD and SVCD. The general answer is:

MPEG files that have already been prepared for VCD or SVCD. If the files conform to the VCD or SVCD specs, *Nero* is able to write a VCD or SVCD on-the-fly without re-encoding the files.

It's important to realize that there are different types of MPEG files. What makes a MPEG suitable for a (S)VCD is way beyond the scope of this documentation text and has to be dealt with by the makers of MPEG encoders. It involves details and settings that simply cannot be chosen via the user interface of existing encoders, unless they have a button dedicated to "(S)VCD encoding".

Having said that, there is a way to at least make *Nero* happy with the source MPEG files. However, this is far from producing a standard compliant CD, because *Nero* cannot test all the relevant aspects.

The most obvious (and most easily met) requirement is picture size:

Format	PAL Resolution	NTSC Resolution
VCD, normal	352x288	352x240
VCD, high	704x576	704x480
SVCD, normal	480x576	480x480
SVCD, high	704x576	704x480

The "high" resolutions are only available for still images, not for movies.

For still images, *Nero* will do the encoding by itself, so the picture sizes may differ. *Nero* will automatically fit the picture into the available space (in a future update, this will be user-configurable).

The frequency for the video is 25Hz for PAL and 29.97Hz for NTSC. The VCD format also allows a "MOVIE" resolution of 352x240 at 23.976Hz, but although this is legal, it is said to cause problems with some players.

Audio must be MPEG-1, layer 2, at 44.1kHz, stereo. SVCD also allows a second music channel and MPEG-2 multi-channel. The second channel is usually used for another language or – in case of Karaoke - for the music without the vocal track.

Apart from these obvious aspects, *Nero* also requires the MPEG-2 file to have a pack size that fits directly into a mode 2, form 2 block, i.e. it must be 2324 bytes large. If this (and for VCD also some other minor aspects) is not met, then *Nero* will list the file as having an "invalid stream encoding".

Nero does not test if scan information is stored in the user data of a stream. Scan information is required by the SVCD standard and might be required by certain players for seeking functions. *Nero* also accepts streams that contain invalid stream IDs.

12. Error Codes

12.1. Verification Errors

Error Code	Error Info
0	No error
1	Files are different
2	CD File is missing
3	File on harddisk is missing
4	CD File is inaccessible
5	HD File is inaccessible
6	Read error on CD file
7	Read error on harddisk file
8	Directory missing on CD
8	Directory missing on harddisk
10	Cannot mount the written CD
11	Don't know the drive letter for the recorder!
12	Unknown verify error
13	Status summary
14	Out of memory
15	I/O error while reading the filesystem structure
16	Error caused by corrupt filesystem structure
17	Error caused by supplying wrong parameters to the fs
18	Too many locks of the filesystem buffers

12.2. WMA Item Errors

Error Code	Error Info
-1	Don't know, what went wrong
0	No Error
1	The MS Audio Codec is not installed
2	Nero does not have the right to open this MS Audio item
3	Error during Unicode character conversion
4	Cannot find the Nero MS Audio DLL 'NeWMAEnc.DLL'
5	Cannot open the MS Audio Codec
6	Cannot determine the format of the MS Audio file
7	Unsupported audio format. Need 44.1kHz, 16 bit, stereo
8	Cannot read the file's attributes
9	Cannot initialize the audio queue
10	Cannot start the MS Audio decoding process
11	Cannot stop the MS Audio decoding process
12	Seek error within the MS Audio file
13	Cannot open the audio output device

Error Code	Error Info
14	Cannot open or write the temporary cache file for the MS Audio file. Disk full!?!
15	Out of memory
16	Cannot calculate the filter effect

12.3. MPEG Audio Item Errors

Error Code	Error Info
1	Lost sync within the MP3 stream. Bad MP3 file?
2	Cannot open the MP3 file.
3	Unexpected MP3 file format.
4	Read error within the MP3 file.
5	Out of memory.
6	**Seek error within the MP3 file.**

12.4. Wave Audio Item Errors

Error Code	Error Info
-1	Out of memory
2	Can't open wave file!
3	Unexpected file format!
4	Read error in wave file!
5	Nero supports only PCM wave files!
6	Unsupported wave sample format!
7	Internal error: Init is missing!
8	Internal error!

12.5. SCSI Hard-disk Page File Errors

Error Code	Error Info
-1	Read error on SCSI device!
-2	Out of memory!
-3	Seek error on SCSI device!
-4	Illegal call (internal error)!

12.6. AVI VCD File Item Errors

Error Code	Error Info
-100	Video MPEG encoder not available
-101	Invalid Video MPEG encoder DLL
-102	This version of the Video MPEG encoder DLL is not compatible

Error Code	Error Info
-103	Cannot create a temporary filename
-104	MPEG2 plugin not installed
-105	Unknown error in the Video MPEG encoder
1	Not enough memory
2	Not enough encoding parameters given to the encoder
3	Unable to open the file
4	Encoding interrupted by the user
5	Cannot start video encoder
6	The parameters were not accepted by the encoder
7	Invalid video format
8	Error creating video format
9	Cannot open output file
10	Error during encoding
11	Decoder not ready
12	Cannot get bitmap
13	Disk full
14	Error retrieving information
15	The encoder cannot produce this output format

12.7. ISO DOC Errors

Error Code	Error Info
-1	Error writing multisession info file!
-2	Error inserting multisession info file into compilation!
-3	Error recreating multisession CD stamp!
-4	Fatal error while trying to read HFS data from SCSI harddisk!
-5	Fatal error while trying to read boot image data!

12.8. Transfer Errors

Error Code	Error Info
1	User aborted
2	Error opening file
3	Error closing file
4	Error reading file
5	Error reading cache file
-1	General Error
-2	function not implemented
-3	no user interface
-4	phase error
-5	not enough memory
-6	Error writing file
-7	Error opening cache file

Error Code	Error Info
-8	Error writing cache file
-9	no driver
-10	Wrong Version
-11	Error accessing cache file
-12	Capability cannot be changed
-13	Error reading cache file
-14	Error writing cache file
-15	Cache file too small
-16	Could not perform StartTrack
-17	Could not perform ReadBuffer
-18	Could not perform EndTrack
-19	Could not perform Fixation
-20	Could not perform Write
-21	Error reading Audio
-22	Cannot cache with Audio-CD
-23	Currently ISO images may only be written on empty CDs!
-24	Could not perform end of Disc At Once
-25	Error reading Data
-26	Recorder doesn´t support Disc At Once
-27	Could not perform start of Disc At Once
-28	Could not generate CD Text information
-30	Cannot write track at correct position with this recorder
-31	Cannot write compilation in DAO to this recorder
-32	Cannot write compilation in TAO to this recorder
-33	Error building internal DAO structures
-34	Cannot write compilation to selected recorders simultaneously
-35	Invalid track info
-36	Could not perform StartDisk
-37	Could not perform StartSession
-38	Could not perform StartTAO
-39	Could not perform EndSession
-40	Could not perform EndDisk

12.9. Image Compression Errors

Error Code	Error Info
-1	Not enough memory
-2	Read error in image file
-3	Image file has illegal format
-4	Illegal function/method call
-5	Error building internal DAO structures
-6	Strange obscure internal error of unknown kind

12.10. CD Recording Errors

Error Code	Error Info
1	Unspecified Recorder Error
2	Bad DLL version
3	Recorder function not implemented
4	Recorder function not supported
-14	Capability read only
-15	Capability unknown
-20	Queuing failed
-21	Waitqueue failed
-22	Abortqueue failed
-23	Out of memory
-24	Waitqueue timeout
-100	Parameter error
-101	Invalid trackinfo
-102	Invalid fixinfo
-103	Invalid trackmode
-104	Invalid pause length
-200	Internal state error
-201	Invalid writestate
-999	No SCSI/IDE command
-1000	SCSI/IDE error
-1001	Absorption control error
-1001	Application code conflict
-1003	ATIP time miscompare
-1004	Attempt to cross track boundary
-1005	Audio play aborted
-1006	Audio play stopped due to error
-1007	Barcode reading error
-1008	Blocksize conflict
-1009	Blocksize format conflict
-1010	Buffer data size error
-1011	Buffer error during recording
-1012	Buffer overflow
-1013	Buffer underrun
-1014	Calibration area allmost full
-1015	Calibration area full
-1016	Cannot read ATIP
-1017	Cannot recover from leadin
-1018	Cannot recover from leadout
-1019	Cannot recover from PMA
-1020	Cannot recover from callibration area
-1021	Cannot recover from track
-1022	Command currently not valid
-1023	Command sequence error

Error Code	Error Info
-1024	Communication failure
-1025	Current command aborted
-1026	Current program area empty
-1027	Damaged track present
-1028	Damaged track detected
-1029	Data track length error
-1030	Diagnostic failure
-1031	Disc is write protected
-1032	Disc style mismatch
-1033	Drive not ready
-1034	Dummy blocks added
-1035	EEPROM failure
-1036	End of medium reached
-1037	End of user area of this track
-1038	Exist packet write area
-1039	Focus or tracking error
-1040	Illegal block size for command
-1041	Illegal cue sheet
-1042	Illegal cue sheet length
-1043	Illegal cue sheet parameter
-1044	Illegal field in command
-1045	Illegal form for reserved track
-1046	This CD is not a CD-Recordable
-1047	Illegal mode for this track
-1048	Illegal packet mode
-1049	Illegal reserve length
-1050	Illegal reserve packet length
-1051	Illegal track
-1052	Illegal track descriptor table
-1053	Illegal track descriptor unit
-1054	Illegal track mode
-1055	Illegal track number
-1056	Illegal track status
-1057	Illegal transfer length
-1058	Inappropriate command
-1059	Initiator detected error message received
-1060	Internal controller error
-1061	Internal target failure
-1062	Invalid audio address
-1063	Invalid bits in identify message
-1064	Invalid block address
-1065	Invalid command
-1066	Invalid field in command
-1067	Invalid field in parameter
-1068	Invalid LUN

Error Code	Error Info
-1069	Invalid message error
-1070	Invalid start address
-1071	Laser current over
-1072	Laser adjustment error
-1073	Link area encountered
-1074	Link blocks encountered
-1075	Logical unit is reserved
-1076	Medium changed
-1077	Medium format corrupted
-1078	Medium load mechanism failed
-1079	Medium load or eject failure
-1080	Medium not present
-1081	Medium removal prevented
-1082	Message error
-1083	Mode parameters changed
-1084	Monitor ATIP error
-1085	Non existent block encountered
-1086	Not enough space
-1087	Not ready to ready transition
-1088	No additional sense information
-1089	No barcode available
-1090	No EFM at search address
-1091	No packet
-1092	No pregap
-1093	No seek complete
-1094	No TOC
-1095	No track for finalize
-1096	Operator medium removal requested
-1097	Overlapped command attempted
-1098	Packet size mismatch
-1099	Parameter list length error
-1100	Parameter list too long
-1101	Parity error
-1102	PCA area full
-1103	PMA area full
-1104	Positioning error
-1105	Power calibration error
-1106	Power calibration error
-1107	Program area not empty
-1108	Recovered data with ECC
-1110	Recovered data with retries
-1111	Recovery needed
-1112	Request for fixation failed
-1113	Reserved track present
-1114	Reset occurred

Error Code	Error Info
-1115	Select or reselect failure
-1116	Selftest failure
-1117	Stopped on non data block
-1118	Track at once not in PMA
-1119	Track following error
-1120	Tray out
-1121	Unable to read TOC, PMA or subcode
-1122	Unable to recover TOC
-1123	Unable to reserve track, because track mode has been changed
-1124	Unknown medium format
-1125	Unrecordable disc
-1126	Unrecoverable damaged track cause too small writable area
-1127	Unrecoverable track descriptor encountered
-1128	Unrecoverable read error
-1129	Unwritten area encountered
-1130	Verify failed
-1131	Write append error
-1132	Write busy
-1133	Write complete
-1134	Write data error with CU
-1135	Write error
-1136	Write illegal block length
-1137	Write operation in progress
-1138	Writing leadin or leadout in progress
-1139	Audio play in progress
-1140	Audio play completed
-1141	Invalid parameter list length
-1142	Synchronous data transfer error
-1143	Blank CD
-1144	No PMA
-1145	OPC initialization error
-1146	OPC execution error
-1147	OPC ALPC error
-1148	OPC timeout
-1149	Read address is ATIP
-1150	Write address is EFM
-1151	Write emergency occurred
-1152	Medium speed error
-1153	Medium no identification
-1154	Medium not supported
-1155	Write emergency
-1156	Servo error
-1157	PLL error
-1158	Address error
-1159	Header monitor error

Error Code	Error Info
-1160	Header search error
-1161	EDC error
-1162	Link/RI/RO error
-1163	CDBD overrun error
-1164	Write audio on reserved track
-1165	Hardware error
-1166	Invisible track present
-1167	Incomplete track present (maybe a CD-RFS-CD?)
-1168	Unknown track present

12.11. Audio Generation Errors

Error Code	Error Info
0	No error
1	User aborted
2	File missing
3	File hat a wrong format
-1	Unspecified Audiogenerator Error
-2	Could not perform StartDisk
-3	Could not perform StartSession
-4	Could not perform StartTrack
-5	Could not perform Write
-6	Could not perform EndTrack
-7	Could not perform EndSession
-8	Could not perform EndDisk
-9	Could not generate DAO information
-10	Error preparing audio track

12.12. Image Generation Errors

Error Code	Error Info
0	No error
1	User aborted
2	File missing
3	File hat a wrong format
-1	Unspecified Imagegenerator Error
-2	Could not perform StartDisk
-3	Could not perform StartSession
-4	Could not perform StartTrack
-5	Could not perform Write
-6	Could not perform EndTrack
-7	Could not perform EndSession
-8	Could not perform EndDisk

-9	Error during ISO relocation
-10	The image chunks:
-11	Error during generation of DAO information
-12	Cannot write track at correct position with this recorder

12.13. ISO 9660 File System Generation Errors

Error Code	Error Info
-3	Joliet names of these files (listed here in ISO-L3) are equal:
-4	The ISO9660 Generator ran out of memory
-5	The following fixed file has an illegal position
-6	Prepartion of ISO9660 structures failed
-7	Too many files in one group of interleaved files
-8	IO error using the boot image
-9	Cannot allocate space on the CD-R
-10	Boot sector not recognized

12.14. Joliet Name Creation Errors

Error Code	Error Info
-3	The listed files have no unique Joliet name:

12.15. Phase Errors

Error Code	Error Info
20	Start of read CD
21	Read CD done
22	Read CD failed
23	Read CD aborted
24	Start of caching files
25	Caching files completed
26	Caching of files failed
27	Caching of files aborted
28	Start of speed measurement
29	Speed measurement done: %dx (%d KB/s)
30	Speed measurement failed
31	Speed measurement aborted
32	Start of simulation at %dx (%d KB/s)
33	Simulation done successfully at %dx (%d KB/s)
34	Simulation failed at %dx (%d KB/s)
35	Simulation aborted
36	Start of burnprocess at %dx (%d KB/s)
37	Burnprocess done successfully at %dx (%d KB/s)
38	Burnprocess failed at %dx (%d KB/s)
39	Burnprocess aborted
40	User aborted
41	Compilation will not fit onto a blank CD
42	Writing Lead In
43	Writing Lead Out
44	Writing Lead In and Lead Out
45	Start generating CD identification
46	Finished generation of CD identification
47	Generating relocation information
48	Analyzing CD
49	Read track info
50	Read media catalog number
51	Read ISRC
52	Find end of track
54	Copy CD
55	Finishing
56	Unspecified Error. Burnprocess failed
57	DAO Oversize burning option activated
59	Writing harddisk backup track
60	Finishing
61	Start of simulation
62	Simulation done successfully

Error Code	Error Info
63	Simulation failed
64	Simulation aborted
65	Start of burnprocess
66	Burnprocess done successfully
67	Burnprocess failed
68	Burnprocess aborted
69	Index scanning
70	Only simulation of first session possible
71	Write several sessions at once; result unreadable in some drives
72	BURN-Proof activated
73	Preparing items
74	Checking discs
75	JustLink activated
76	Change of write speed due to media
77	Write tracks
78	Verify compilation
79	Verify aborted
80	Verify compilation done successfully
81	Verify compilation failed
82	Encoding video
83	Start checking for virus infected files
84	End of virus checking phase
85	End of virus checking. A virus was found and burning has to be aborted!
86	CD-Text not supported
87	Seamless Link activated
88	ExacLink activated
89	Super Link activated
90	Buffer underrun protection activated
91	poor performance on your DMA-chipset-driver (look for updates)
92	recording quality may be affected not using adaptive writespeed option

12.16. MMC Errors

Error Code	Error Info
0000	No additional sense information
0006	I/O process terminated
0011	Audio play in progress
0017	Cleaning requested
0200	No seek complete
0400	Logical unit not ready, cause not reportable
0401	Logical unit in process of becoming ready
0402	Logical unit not ready, initialization command required
0403	Logical unit not ready, manual intervention required
0404	Logical unit not ready, format in progress

Error Code	Error Info
0407	Logical unit not ready, operation in progress
0408	Logical unit not ready, long write in progress
0500	Logical unit does not respond to selection
0600	No reference position found
0700	Multiple peripherial devices selected
0800	Logical unit communication failure
0801	Logical unit communication time-out
0802	Logical unit communication parity error
0900	Track following error
0901	Track servo error
0902	Focus servo failure
0903	Spindle servo failure
0904	Head select fault
0A00	Error log overflow
0B00	Warning
0B01	Warning, specified temperaturee exceeded
0B02	Warning, enclosure degraded
0C00	Write error
0C07	Write error, recovery needed
0C08	Write error, recovery failed
0C09	Write error, loss of streaming
0C0A	Write error, padding blocks added
1000	Invalid block size for command
1001	Packet does not fit into available space
1002	Buffer underrun
1100	Unrecovered read error
1101	Read retries exhausted
1102	Error too long to correct
1105	L-EC uncorrectable error
1106	CIRC unrecovered error
110F	Error reading UPC/EAN number
1110	Error reading ISRC number
1111	Read error, loss of streaming
1500	Random positioning error
1501	Mechanical positioning error
1502	Positioning error detected by read of medium
1700	Recovered data with no error correction applied
1702	Recovered data with retries
1702	Recovered data with positive head offset
1703	Recovered data with negative head offset
1704	Recovered data with retries and/or CIRC applied
1705	Recovered data using previous secotr id
1707	Recovered data without ECC - recommend reassignment
1708	Recovered data without ECC - recommend rewrite
1709	Recovered data without ECC - data rewritten

Error Code	Error Info
1800	Recovered data with error correction applied
1801	Recovered data with error correction & retries applied
1802	Recovered data - data auto-reallocated
1803	Recovered data with CIRC
1804	Recovered data with L-EC
1805	Recoverred data - recommen reassignment
1806	Recoverred data - recommend rewrite
1A00	Parameter list length error
1B00	Synchronous data transfer error
1D00	Miscompare during verify operation
2000	Invalid command operation code
2100	Logical block out of range
2101	Invalid element address
2102	Invalid address for write
2200	Illegal function
2400	Invalid field in CDB
2500	Logical unit not supported
2600	Invalid field in parameter list
2601	Parameter not supported
2602	Parameter value invalid
2603	Threshold parameters not supported
2604	Invalid release of active persistent reservation
2700	Write protected
2701	Hardware write protected
2702	Logical unit software write protected
2703	Associated write protect
2704	Persistant write protect
2705	Permanent write protect
2800	Not ready to ready change, medium may have changed
2801	Import or export element accessed
2900	Power on, reset or bus device reset occurred
2901	Power on occurred
2902	SCSI bus reset occurred
2903	Bus device reset funtion occurred
2904	Device internal reset
2A00	Parameters changed
2A01	Mode parameters changed
2A02	Log parameters changed
2A03	Reservations preempted
2B00	Copy cannot execute since host cannot disconnect
2C00	Command sequence error
2C03	Current program area is not empty
2C04	Current program area is empty
2C05	Persistent prevent conflict
2E00	Insufficient time for operation

Error Code	Error Info
2F00	Commands cleared by another initator
3000	Incompatible medium installed
3001	Cannot read medium - unknown format
3002	Cannot read medium - incompatible format
3003	Cleaning cartridge installed
3004	Cannot write medium - unknown format
3005	Cannot write medium - incompatible format
3006	Cannot format medium - incompatible medium
3007	Cleaning failure
3008	Cannot write - application code mismatch
3009	Current session not fixated for append
3100	Medium format corrupted
3101	Format command failed
3400	Enclosure failure
3500	Enclosure services failure
3501	Unsupported enclosure failure
3502	Enclosure services unavailable
3503	Enclosure services transfer failure
3504	Enclosure services transfer refused
3700	Rounded parameter
3900	Saving parameters not supported
3A00	Medium not present
3A01	Medium not present - tray closed
3A02	Medium not present - tray open
3B0D	Medium destination element full
3B0E	Medium source element empty
3B0F	End of medium reached
3B11	Medium magazine not accessible
3B12	Medium magazine removed
3B13	Medium magazine inserted
3B14	Medium magazine locked
3B15	Medium magazine unlocked
3D00	Invalid bits in identify message
3E00	Logical unit has not self-configured yet
3E01	Logical unit failure
3E02	Timeout on logical unit
3F00	Target operating conditions have changed
3F01	Microcode has been changed
3F02	Changed operating definition
3F03	Inquiry data has changed
4000	Diagnostic failure on component
4300	Message error
4400	Internal target failure
4500	Select or reselect failure
4600	Unsuccessful soft reset

Error Code	Error Info
4700	SCSI parity error
4800	Initiator detected error message received
4900	Invalid message error
4A00	Command phase error
4B00	Data phase error
4C00	Logical unit failed self-configuration
4D00	Tagged overlapped commands
4E00	Overlapped commands attempted
5100	Erase failure
5300	Media load or eject failed
5302	Medium removal prevented
5380	Eject button is pressed in prevent condition
5700	Unable to recover table of contents
5A00	Operator request or state change input
5A01	Operator medium removal request
5A02	Operator selected write protect
5A03	Operator selected write permit
5B00	Log exception
5B01	Threshold condition met
5B02	Log counter at maximum
5B03	Log list codes exhausted
5D00	Failure prediction threshold exceeded
5DFF	Failure prediction threshold exceeded (FALSE)
5E00	Low power condition on
5E01	Idle condition activated by timer
5E02	Standby condition activated by timer
5E03	Idle condition activated by command
5E04	Standby condition activated by command
6000	Lamp failure
6100	Video acquisition error
6102	Out of focus
6200	Unable to acquire video
6300	End of user area encountered on this track
6301	Packet does not fit in available space
6400	Illegal mode for this track
6401	Invalid packet size
64C0	Illegal speed for this medium
6500	Voltage fault
7200	Session fixation error
7201	Session fixation error writing lead-in
7202	Session fixation error writing lead-out
7203	Session fixation error - incomplete track in session
7204	Empty or partially written reserved track
7300	CD control error
7301	Power calibration area almost full

Error Code	Error Info
7302	Power calibration area is full
7303	Power calibration area error
7304	Program memory area update failure
7305	Program memory area is full
7306	RMA/PMA is almost full
8900	Inappropriate command
9082	Read Write Control Task Error
9141	Mechanism Control Task Error
91A1	Mechanism Control Task Error
C100	Illegal track status
C200	Reserved track present
C400	Illegal reserve length for reserve track command
C500	Buffer underrun
C601	Unwritten area encountered
C602	Link blocks encountered
CC00	Not enough space
CD00	No track present to close session
CE00	Unrecordable track descriptor encountered
CF00	Damaged track present
D000	PMA area full
D100	PCA area full
D400	Exit from pseudo track at once recording

12.17. MP 6200 Errors

Error Code	Error Info
0000	No additional sense information
0006	I/O process terminated
0011	Audio play in progress
0012	Audio play paused
0100	Recovery needed
0101	Recovery failed
0200	No seek complete
0400	Logical unit not ready, cause not reportable
0401	Logical unit in process of becoming ready
0402	Logical unit not ready, initialization command required
0403	Logical unit not ready, manual intervention required
0601	Power calibration area almost full
0800	Logical unit communication failure
0900	Track following error
0901	Track servo error
0902	Focus servo failure
0903	Spindle servo failure
0980	Radial skating error

Error Code	Error Info
0981	Sledge servo failure
0982	PLL no lock
0983	Servo off track
0984	ATIP sync error
0985	ATIP/subcode jumped error
0C00	Write error
1000	Invalid block size for command
1001	Packet does not fit into available space
1002	Buffer underrun
1003	Disc is not empty
1004	Current program area is empty
1005	Current program area is not empty
1100	Unrecovered read error
1200	Invalid UPC/EAN number
1201	Error reading UPC/EAN number
1202	Invalid ISRC number
1500	Random positioning error
1700	Recovered data with no error correction applied
1701	Recovered data with retries
1800	Recovered data with error correction applied
1801	Recovered data with error correction & retries applied
1A00	Parameter list length error
2000	Invalid command parameter list
2100	Logical block out of range
2400	Invalid field in CDB
2500	Logical unit not supported
2600	Invalid field in parameter list
2601	Parameter not supported
2602	Parameter value invalid
2700	Write protected
2800	Not ready to ready change, medium changed
2900	Power on, reset or bus device reset occurred
2A00	Parameters changed
2A01	Mode parameters changed
2A02	Padding blocks added
2C00	Command sequence error
2F00	Commands cleared by another initator
3000	Incompatible medium installed
3001	Cannot read medium - unknown format
3002	Cannot read medium - incompatible format
3004	Cannot write medium - unknown format
3005	Cannot write medium - incompatible format
3100	Medium format corrupted
3101	Format command failed
3900	Saving parameters not supported

Error Code	Error Info
3A00	Medium not present
3F00	Target operating conditions have changed
3F01	Microcode has been changed
3F03	Inquiry data has changed
4000	Diagnostic failure on component
4400	Internal target failure
4500	Select or reselect failure
4700	SCSI parity error
4800	Initiator detected error message received
4900	Invalid message error
4E00	Overlapped commands attempted
5000	Write append error
5300	Media load or eject failed
5302	Medium removal prevented
5380	Eject button is pressed in prevent condition
5700	Unable to recover table of contents
57C0	Subcode not found
57C1	ATIP not found
57C2	No ATIP or subcode
57C3	PMA error
57C4	TOC read error
57C5	Disc information error
5A00	Operator request or state change input
5A01	Operator medium removal request
6300	End of user area encountered on this track
6400	Illegal mode for this track
6480	Read in lead-in
6481	Read in lead-out
64C0	Illegal speed for this medium
7100	Session fixation error
7101	Session fixation error while writing lead-in
7102	Session fixation error while writing lead-out
7103	Session fixation error - incomplete track in session
7104	Session fixation error - empty or partially written track
7105	Cannot write session at once, no cue sheet
8200	Command now not valid
9000	Reserve track check error
9082	Read Write Control Task Error
9141	Mechanism Control Task Error
91A1	Mechanism Control Task Error
9200	Header search error
9801	Header monitor error
9802	EDC error
9803	Read link, run-in or run-out
9804	Last one block error

Error Code	Error Info
9805	Illegal block size
9806	Not all data transferred
9807	CDBD overrun error
A100	Invalild start address
A500	Application code mismatch
AA00	End of medium reached
C101	Invalid track number
C280	Syscon ROM error
C281	Syscon RAM error
C290	EFM encoder error
C291	EFM decoder error
C292	DAC error
C295	Syscon EEPROM error
C2A0	Block decoder communication error
C2A1	Block encoder communication error
C2A2	Block encoder/decoder path error
C300	CD-R engine selftest error
C400	Buffer parity error
C500	Data transfer error
C601	Unwritten area encounterred
C900	Illegal block length
CA00	Power calibration error
CA80	Power range error
CA81	Modulation error
CA82	Beta upper range error
CA83	Beta lower ranger error
CA84	Alpha upper range error
CA85	Alpha lower range error
CD00	No track present to complete
D000	PMA area is full
D001	PMA update failed
D102	Power calibration area is full
3000	12V failure
3100	Undefined syscon error
E101	Syscon communication error
E102	Unknonw syscon error
E200	Syscon not ready
E300	Command rejected
E400	Command not accepted
E500	Verify error at the beginning of track
E501	Verify error at the ending of track
E502	Verify error at the beginning of lead-in
E503	Verify error at the ending of lead-in
E504	Verify error at the beginning of lead-out
E505	Verify error at the ending of lead-out

Error Code	Error Info
FF00	Command phase timeout error
FF01	Data in phase timeout error
FF02	Data out phase timeout error
FF03	Status phase timeout error
FF04	Message in phase timeout error
FF05	Message out phase timeout error

13. Known Limitations

Currently there are no Linux versions of the *NeroAPI* and the *Nero* Software Development Kit (*NeroSDK*).

14. Bibliography

14.1. C Programming Books

For those who never have programmed before:

Greg M. Perry: Absolute Beginner's Guide to C

From the guys who invented C. Only for beginners who like challenge:

Brian W. Kernighan and Dennis M. Ritchie: The C Programming Language

14.2. C Programming Online Resources

This site is both for C and C++ programming

http://www.cprogramming.com

Steve Summit's Introductory C course

http://www.eskimo.com/~scs/cclass/cclass.html

14.3. C++ Programming Books

If you know something better than those let us know.

Ivor Horten: Beginning Visual C++ 6

Davis Chapman: Sams Teach Yourself Visual C++ in 21 days

From the inventor of C++. Not for the faint of heart.

Bjarne Stroustrup: The C++ Programming Language

The author's favourite author. ;-) Read it if you think you know C++ inside and out.

James Coplien: Advanced C++ Programming Styles and Idioms

14.4. C++ Online Resources

Valencia Community College C++ programming course

http://m2tech.net/cppclass/

Intended for C users who want to make the transition to C++

http://www.icce.rug.nl/docs/cplusplus/cplusplus.html

A very good site

http://www.codeproject.com

14.5. General CD/CD-ROM Online Resources

Glossary of CD-ROM and DVD technologies

http://www.sigcat.org/resource/gloss697.htm

14.6. Audio CD Online Resources

Digital Audio on CD

http://www.disctronics.co.uk/cdref/cdaudio/cdaudio.htm

14.7. Super Video CD Online Resources

MPEG-2 encoder test

http://www.tecoltd.com/enctest/enctest.htm

A well-researched page on SVCD
http://www.iki.fi/znark/video/svcd/overview/

German page with a similar mission

http://www.ratos.de/