NeroCmd

v1.6

Developer's Manual

NeroCmd will only work with
a fully installed Nero version!

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

1. Contents

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

NeroCmd is a console application that facilitates the processing of commands understood by the *NeroAPI*.

This part of the documentation has been created for developers who want to change the *NeroCmd* parser to fit their own, more refined requirements. Users who want to benefit from the functionality without having to understand the machinery inside should refer to the "NeroCmd User's Manual", which gives a comprehensive description of how to use the application.

3.2. Overview

NeroCmd can perform the following tasks:

- List all available drives
- Display capabilities of a particular drive
- List available read and write speeds for a particular drive
- Get CD info for the currently loaded CD from a particular drive
- List supported Audio formats
- Burn:
    - ISO DVD
    - ISO/Audio CD
    - Video CD
    - Super Video CD
    - CD from image
    - Freestyle CD
- Grab Audio tracks and store them in disk files (Digital Audio Extraction)
- Eject/Load CD from drive
- Erase CD Rewritable/ DVD Rewritable
- Display NeroAPI version information

This paper will guide you through the architecture of the application.

3.3. Requirements

NeroCmd will work on any platform, which is fit for hosting ***Nero 5.5.9.14***. Nero needs to be installed prior to using *NeroCmd*.

NeroCmd will not work with Linux.

You can obtain the latest version of *Nero* from http://www.nero.com.
To compile *NeroCmd* you will need Microsoft Visual C++ 6.0 or Microsoft Visual Studio .NET.

We have used Visual C++ 6.0 with the Visual Studio Service Pack 5. If you experience problems under Visual C++ 6.0, and do not have that service pack installed, you might want to obtain the Service Pack, before taking other options into consideration.

3.4. Required Skills

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

You should also have aquired some familiarity with the *NeroAPI*. The *NeroAPI* documentation can be found in the "NeroAPI/Doc" folder of the *NeroSDK*.

4. Files in the Package

NeroCmd comes as a zipped file; the contents of this ZIP archive are listed and explained below.

4.1. File Description

On the following pages you will find a complete list of files that are part of the *NeroCmd* source code distribution. If the application that you have created does not work it might well be that some of these files are missing. A short description of the file's purpose has been added where suitable or required.

The noteworthy characteristic of the structure is the absence of a direct mapping between classes and files. Many member functions of CBurnContext, the central class for most operations, are quite complex and therefore have their own module.

4.1.1. Visual C++ Project Files

Path \ File Name	Description
NeroCmd.dsw	Visual C++ Workspace for NeroCmd.
NeroCmd.dsp	Visual C++ project file for NeroCmd.

4.1.2. Executable Files

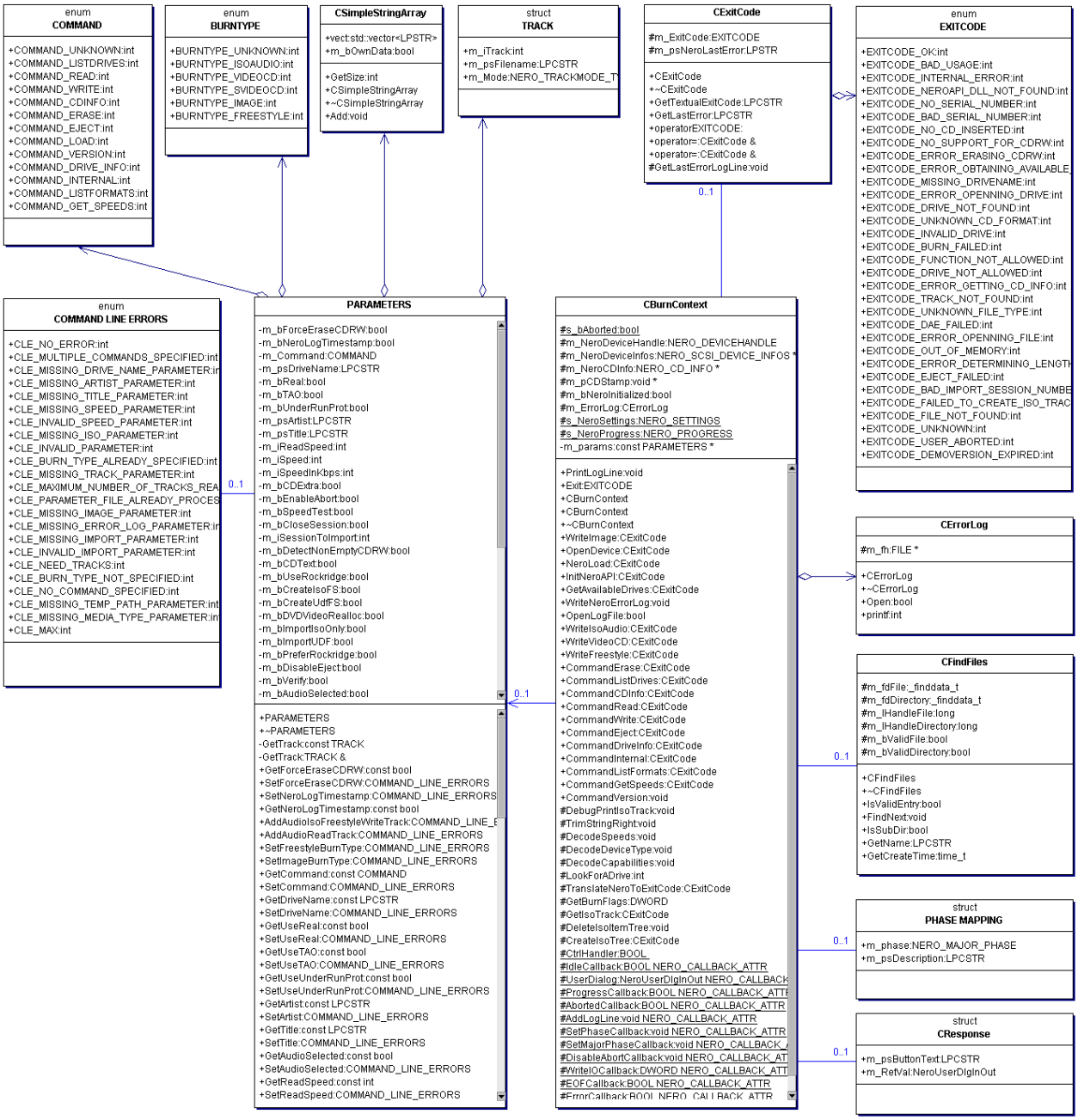
Path \ File Name	Description
NeroCmd.exe	Release version of the NeroCmd executable.

4.1.3. NeroCmd Source Code

Path \ File Name	Description
AbortedCallback.cpp	AbortedCallback implementation.
AddLogLineCallback.cpp	AddLogLine callback implementation.
BurnContext.cpp	Implementation of the CBurnContext class.
BurnContext.h	Central class for CD operations.
CommandCDInfo.cpp	Implements the –cdinfo command.
CommandDriveInfo.cpp	Implements the –driveinfo command.
CommandEject.cpp	Implements the –eject and –load commands.
CommandErase.cpp	Implements the –erase command for CDRW/DVDRW media.
CommandGetSpeeds.cpp	Implements the –get_speeds command.
CommandInternal.cpp	Implements the –internal command. This command is only used for internal testing purposes.
CommandListDrives.cpp	Implements the –listdrives command, which will list all available drives with their main characteristics.
CommandListFormats.cpp	Implements the –listformats command to display the available audio formats.
CommandRead.cpp	Implements DAE (digital audio extraction) through the –read

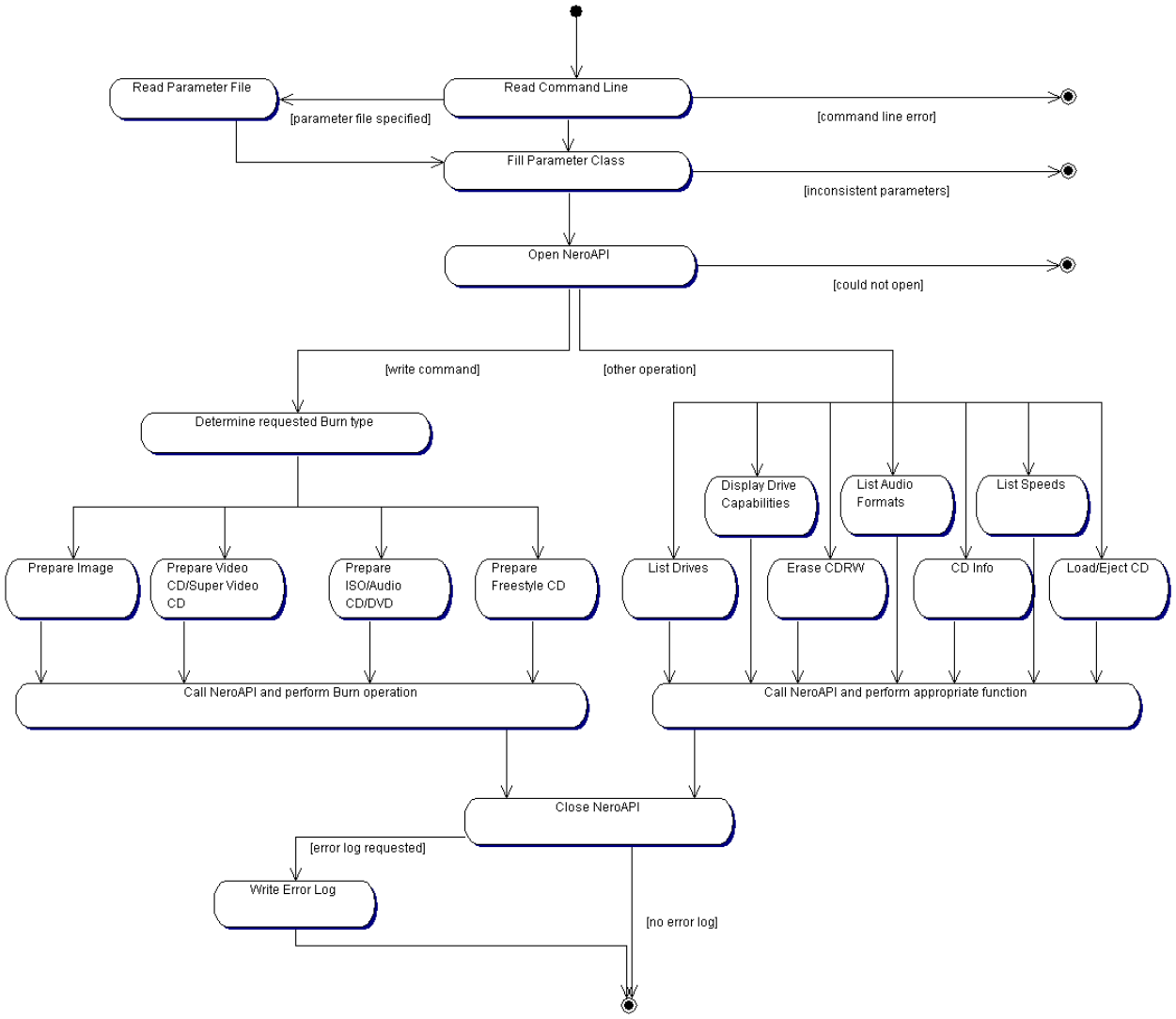
Path \ File Name	Description
	command.
CommandVersion.cpp	Implements the –version command for retrieval and printing of version information.
CommandWrite.cpp	Implements the general –write command and distinguishes between different burn types and acts accordingly.
CtrlHandler.cpp	Handles Ctrl events.
DisableAbortCallback.cpp	This is one of the NeroAPI callbacks. It prints out the info to remind user whether the current operation is abortable.
ErrorLog.cpp	CErrorLog class implementation.
ErrorLog.h	Logging of error messages created by the application
ExitCode.cpp	Translate the numeric error code into a textual representation.
ExitCode.h	Supported exit codes and translation to textual representation.
ExitCodeTranslator.cpp	Translation of NeroAPI errors to EXITCODEs.
FindFile.cpp	Implementation of helper class for ISO tree handling moved from.
FindFile.h	Helper class for ISO tree handling.
getopt.cpp	Decoding of argument list, help function, parsing of parameter file.
getopt.h	COMMAND and BURNTYPE enumerations, PARAMETERS declaration.
IdleCallback.cpp	Callback for idle processing.
IOCallbacks.cpp	Callbacks that do not deal with files directly.
IsoTrack.cpp	ISO tree handling.
NeroCmd.cpp	Main file of the application.
parameters.cpp	PARAMETERS class implementation file
parameters.h	PARAMETERS class declaration, COMMAND_LINE_ERRORS enumeration, enumerations for available burn types and commands; structure for storing a track list.
ProgressCallback.cpp	Callback for displaying progress on current operation.
resource.h	Resource header file for version.rc.
SetMajorPhaseCallback.cpp	Callback for reporting major phase changes.
SetPhaseCallback.cpp	Callback for reporting phase changes.
SimpleStringArray.cpp	Simple string vector class implementation.
SimpleStringArray.h	Declaration of a simple string vector class.
StdAfx.cpp	Source file that contains the standard includes
StdAfx.h	Include file for standard system include files, or project specific include files that are used frequently, but are changed infrequently.
UserDialog.cpp	Interaction with the user.
version.rc	Version resource script.
WriteFreestyle.cpp	Freestyle format burning.
WriteImage.cpp	ISO image burning.
WriteIsoAudio.cpp	ISO/Audio format burning.
WriteNeroErrorLog.cpp	Write the error log to a file.
WriteVideoCD.cpp	Video format burning.

5. NeroCmd Class Overview



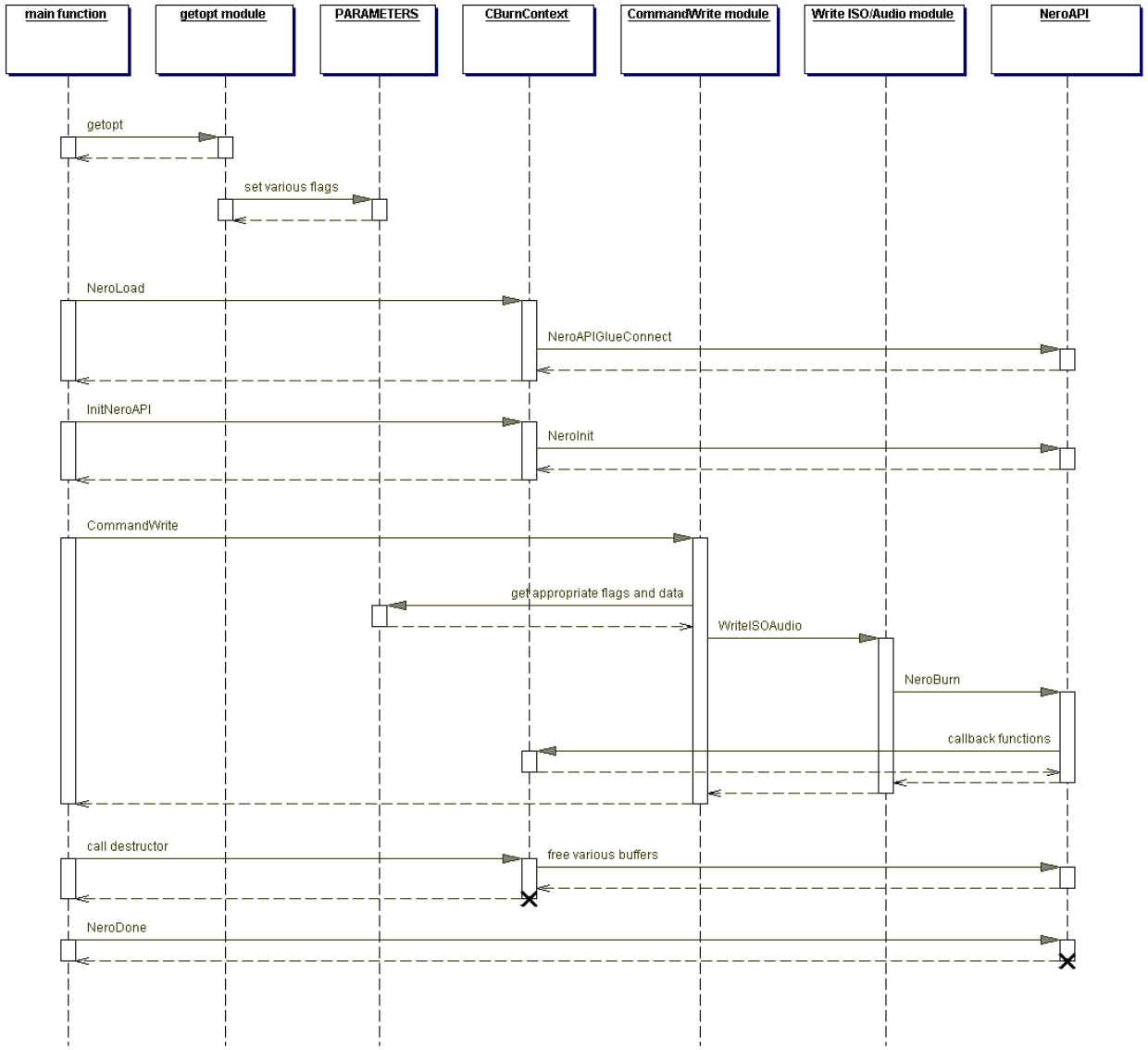
This overview shows the basic components of NeroCmd. CBurnContext is the central class. CBurnContext uses the PARAMETERS class, where all the information obtained by the command line input has been stored, to assemble the appropriate data for every burn format, and execute calls to NeroAPI to perform the desired operation.

6. NeroCmd Activity Overview



The diagram above shows a very rough overview of the required operations.

7. Example Sequence Diagram: Write ISO/Audio



This sequence diagram shows the important operations when NeroCmd has been instructed to write an ISO/Audio CD or DVD. The different instances do not exactly map to classes, to better illustrate the distribution of functionality to different modules.

Including all classes that are used would have made the sequence diagram too complex, so e.g. calls to CFindFiles and CSimpleStringArray classes have been disregarded here. Also, repeated single function calls that – for instance – are required to PARAMETERS have been reduced to "set various flags".

8. Global functions

8.1. The main function

```
int main(int argc, char* argv[])
```

Inside the main() function the PARAMETERS and CBurnContext classes are instantiated; a variable is provided for the EXITCODE enumeration. The global getopt() function is called with the command line parameters the user entered on the command line. The getopt function will fill the PARAMETERS structure according to these command line parameters.

If getopt returns EXITCODE_BAD_USAGE, because the command line parameters have not been provided properly, the application exits. If getopt was successful in parsing the parameters, the NeroAPI is loaded, and the desired command is executed by calling the appropriate member function of CBurnContext.

8.2. The getopt function

```
bool getopt (PARAMETERS & params, int argc, char ** argv)
```

getopt() checks if there are any parameters; if not, a help on usage is displayed on the screen and the application exits. Otherwise the following commands are allowed: listdrives, driveinfo, listformats, get_speeds, version, cdinfo, read, write, erase, eject and load. Only one command at a time is allowed. The actual checking for command line parameter consistency is done in the PARAMETERS class.
If a commercial at "@" is encountered, the ReadParameterFile function is called to overwrite the argument strings with the content of a parameter file.

8.3. The ReadParameterFile function

```
static bool ReadParameterFile (PARAMETERS & params, LPCSTR psFilename)
```

ReadParameterFile is responsible for reading and parsing the parameter list from a disk file. Once the parameters are parsed, they are passed on to the getopt function for decoding.

8.4. The Usage function

```
static void Usage (void)
```

Usage will display a number of help screens on allowed commands and flags.

9. NeroCmd classes

9.1. The CBurnContext

This is the central class for CD and DVD operations. It handles all available operations and callbacks.

9.1.1. The CBurnContext constructor

```
CBurnContext::CBurnContext(PARAMETERS* params)
```

CBurnContext(), as we use it, gets a PARAMETERS pointer, that is then stored in a member variable. It will later be used during NeroAPI callbacks to retrieve settings from the PARAMETERS class. The default constructor is not used.

In the constructor various handles are set to an uninitialized value. The Console Control handler is set to the CtrlHandler member function to provide handling of keyboard inputs like Ctrl+C.

9.1.2. The ~CBurnContext destructor

```
CBurnContext::~CBurnContext ()
```

If the NeroAPI had been successfully initialized, cleanup functions like device closing and memory deallocation are performed. Even if the NeroAPI had not been initialized before, the NeroDone() function is executed to make sure that all threads are stopped before the NeroAPI DLL is closed.

9.1.3. The AbortedCallback member function

```
BOOL NERO_CALLBACK_ATTR CBurnContext::AbortedCallback (void *pUserData)
```

AbortedCallback is one of the "NeroCallback" callback functions. It simply returns the flag maintained in CBurnContext.

9.1.4. The AddLogLine member function

```
void NERO_CALLBACK_ATTR CBurnContext::AddLogLine (void *pUserData,
                                    NERO_TEXT_TYPE type, const char *text)
```

This function analyzes the type parameter and assigns a log header accordingly. Then the header and text are printed, breaking lines after 76 characters.

Header	Description
[i]	Informative text.
[#]	Some operation stopped prematurely.

Header	Description
[!]	Important information.
[?]	A question which requires an answer.
[-]	A message concerning a CD-ROM drive or recorder

9.1.5. The CommandCDInfo member function

```
CExitCode CBurnContext::CommandCDInfo (const PARAMETERS & params)
```

If the user supplied "—cdinfo" on the command line, this function executes a CD info command. It simply calls NeroAPI's NeroGetCDInfo function and displays information about media type, number of free blocks, access type, artist and title.

Then information about every track is displayed, including track number, track type, start block, end block, track length in blocks and session number. If artist and title information are available, they will be displayed as well.

9.1.6. The CommandDriveInfo member function

```
CExitCode CBurnContext::CommandDriveInfo (const PARAMETERS & params)
```

If the user supplied "—driveinfo" on the command line, this function will retrieve and display drive letter, device name, device id, host adapter name, host adapter number, buffer underrun protection technology, drive buffer size, supported media and speeds for a particular drive.

9.1.7. The CommandEject member function

```
CExitCode CBurnContext::CommandEject (const PARAMETERS & params)
```

If the user supplied "—eject" or "—load" on the command line, this function executes the eject and load commands by calling NeroAPI's NeroEjectLoadCD function.

9.1.8. The CommandErase member function

```
CExitCode CBurnContext::CommandErase (const PARAMETERS & params)
```

If the user supplied "—erase" on the command line, this function erases a CDRW either entirely or in quick mode, depending on whether the "—entire" flag was set. The NeroGetCDRWErasingTime function is called. A negative return value indicates that either no CD was inserted or the drive does not support CDRWs. Upon a negative return value the function will terminate.

Otherwise, the estimated time for deletion is displayed and NeroEraseCDRW is called to actually perform the deletion.

9.1.9. The CommandGetSpeeds member function

```
CExitCode CBurnContext::CommandGetSpeeds (const PARAMETERS & params)
```

If the user supplied "—get_speeds" on the command line, this function displays a list of all available read and write speeds for a particular drive.

CommandGetsSpeeds calls NeroGetAvailableSpeeds twice to retrieve read and write speeds for a particular drive, then displays the results.

9.1.10. The CommandInternal member function

```
CExitCode CBurnContext::CommandInternal (const PARAMETERS & params)
```

This command is used strictly for internal testing.

9.1.11. The CommandListDrives member function

```
CExitCode CBurnContext::CommandListDrives (const PARAMETERS & params)
```

If the user supplied "—listdrives" on the command line, this function displays a list of all available drives with their main characteristics.

Information about the available drives has been retrieved during initialization and stored in a member variable of the class.

The number of available drives is used in a for-loop: For every drive the drive letter, device name, buffer underrun protection technology name, host adapter number, host adapter name, and device ID are displayed.

9.1.12. The CommandListFormats member function

```
CExitCode CBurnContext::CommandListFormats (const PARAMETERS & params)
```

If the user supplied "—listformats" on the command line, this function displays a list of all available audio formats, including some basic information about each format.

The formats are retrieved by looping through the NeroAudioGetFormatInfo function until it returns FALSE.

9.1.13. The CommandRead member function

```
CExitCode CBurnContext::CommandRead (const PARAMETERS & params)
```

If the user supplied "—read" on the command line, this function performs DAE (digital audio extraction). First, the CD info is retrieved. Then the function enumerates through the user supplied list of tracks. It tries to find every single track

among the existing tracks. If the track was not found, an error will be reported. Otherwise the audio data will be extracted.
The file extension is determined - supported extensions are WAV and PCM. Any file handles that have been opened during this operation will be closed before the function returns.

9.1.14. The CommandVersion member function

```
void CBurnContext::CommandVersion (void)
```

This function is called when the user supplied "—version" on the command line. A call to NeroGetAPIVersionEx retrieves the information, which is then formatted and displayed.

9.1.15. The CommandWrite member function

```
CExitCode CBurnContext::CommandWrite (const PARAMETERS & params)
```

If the user supplied "—write" on the command line, this function distinguishes between different burn types and acts accordingly. The GetBurnType() function of the PARAMETERS class is used to determine whether an image, Audio/ISO, Video CD, or Super Video CD has been requested by the user. Then the appropriate member function for each burn type will be called.

9.1.16. The CreateIsoTree member function

```
CExitCode CBurnContext::CreateIsoTree (LPCSTR psFilename,
                                       NERO_ISO_ITEM ** ppItem, int iLevel)
```

This function searches for a specified path and recursively adds all files and directories that are found. It starts by defining an instance of the CFindFiles helper class. If the first filename that has been supplied cannot be found, an error is returned.

9.1.17. The CtrlHandler member function

```
BOOL WINAPI CBurnContext::CtrlHandler (DWORD dwCtrlType)
```

Whatever event occurred is handled by aborting any current operation.

9.1.18. The DebugPrintIsoTrack member function

```
void CBurnContext::DebugPrintIsoTrack (const NERO_ISO_ITEM * pItem,
                                                          int iLevel)
```

This function is used solely for debug purposes in order to print the whole ISO tree.

9.1.19. The DecodeCapabilities member function

```
void CBurnContext::DecodeCapabilities (const NERO_SCSI_DEVICE_INFO *
                                                              pNSDI) const
```

This function displays the available capabilities for a drive (DAO, CD text, bus type...).

9.1.20. The DeleteIsoItemTree member function

```
void CBurnContext::DeleteIsoItemTree (NERO_ISO_ITEM * pItem)
```

This function deletes the ISO tree recursively. If a directory is encountered, DeleteIsoItemTree is called again with the first item in that directory. If the item is a reference to another item, NeroFreeMem is called, otherwise NeroFreeIsoItem.

9.1.21. The DisableAbortCallback member function

```
void NERO_CALLBACK_ATTR CBurnContext::DisableAbortCallback (void
                                          *pUserData, BOOL enableAbort)
```

This is one of the NeroAPI callbacks. It prints out information to let the user know whether the current operation is abortable or not.

9.1.22. The EOFCallback member function

```
BOOL NERO_CALLBACK_ATTR CBurnContext::EOFCallback (void *pUserData)
```

This IO callback is one of the callbacks, which perform operations with NeroAPI that do not deal with files directly. It returns the result of a feof() function call, to determine whether the end of a file has been reached.

9.1.23. The ErrorCallback member function

```
BOOL NERO_CALLBACK_ATTR CBurnContext::ErrorCallback (void *pUserData)
```

This IO callback is one of the callbacks, which perform operations with NeroAPI that do not deal with files directly. It uses ferror() to check for error and returns the result.

9.1.24. The Exit member function

```
EXITCODE CBurnContext::Exit(CExitCode code)
```

This function takes an CExitCode class object, translates it to its textual representation by calling its GetTextualExitCode member function, and returns the provided exit code.

9.1.25. The GetAvailableDrives member function

```
CExitCode CBurnContext::GetAvailableDrives (void)
```

This function is used to set the m_NeroDeviceInfos member of CBurnContext to the return value of NeroGetAvailableDevicesEx. It returns EXITCODE_OK or EXITCODE_ERROR_OBTAINING_AVAILABLE_DRIVES depending on the value of m_NeroDeviceInfos.

9.1.26. The GetBurnFlags member function

```
DWORD CBurnContext::GetBurnFlags (const PARAMETERS & params)
```

This function sets the appropriate burn flags according to the user supplied parameters. This includes real mode or simulation, TAO or DAO, abort disabling, speed test, session closing, buffer underrun protection, non-empty CDRW detection, CD text option, eject disabling, and verification.

9.1.27. The GetIsoTrack member function

```
CExitCode CBurnContext::GetIsoTrack (const PARAMETERS & params,
                  CNeroIsoTrack ** ppIsoTrack, NERO_ISO_ITEM ** ppItem)
```

This function creates a CNeroIsoTrack from the user supplied parameters.
It imports a previous session if requested by the user and builds the file and directory tree.

If "—import" was specified NeroGetCDInfo is called and a pointer to a NERO_CD_INFO structure for the specified device is retrieved. The function checks whether the requested import track exists on the CD. (If no track number was specified, the last session will be imported.)

Then the import flags are set according to the "—import_udf", "—import_iso_only" or "—prefer_rockrige" flags. The NeroImportIsoTrackEx function is called.

If the NeroImportIsoTrackEx function fails, it is probably due to an empty drive.

Now the function iterates through the file list and adds each item to the tree. If a directory is found it is recursed and every contained item is added to the tree.

Depending on the user supplied command line parameters "—iso_mode2", "—use_rockridge", "—create_iso_fs", " –dvdvideo_realloc" and "—create_udf_fs" internal flags are set.

Then NeroCreateIsoTrackEx is called. If track creation fails the ISO item tree is deleted.

9.1.28. The IdleCallback member function

```
BOOL NERO_CALLBACK_ATTR CBurnContext::IdleCallback (void *pUserData)
```

This is a NeroAPI callback responsible for idle processing. Since we have no idle processing, we simply return our aborted flag.

9.1.29. The InitNeroAPI member function

```
CExitCode CBurnContext::InitNeroAPI (void)
```

This function fills the Nero settings structure with CBurnContext's this-pointer and the address of the UserDialog function then it initializes the NeroAPI. The return value is mapped to an EXITCODE.

9.1.30. The LookForADrive member function

```
int CBurnContext::LookForADrive (const PARAMETERS & params)
```

This is a support function that enumerates drives and finds the one that matches the specified command line parameters. It will accept both device names and drive letters.

9.1.31. The NeroLoad member function

```
CExitCode CBurnContext::NeroLoad (void)
```

This function connects to the NeroAPI, and sets the m_bNeroInitialized flag if the operation was successful.

9.1.32. The OpenDevice member function

```
CExitCode CBurnContext::OpenDevice (const PARAMETERS & params)
```

This function opens a device. It checks for the presence of a device by enumerating drives and trying to find the requested drive among them.

9.1.33. The OpenLogFile member function

```
bool CBurnContext::OpenLogFile (LPCSTR psLogFilename)
```

This function opens the log file by calling the Open function of the m_ErrorLog member. It returns true if the log could be opened.

9.1.34. The PrintLogLine member function

```
void CBurnContext::PrintLogLine(LPCSTR s)
```

This function prints error log lines that are passed to the CBurnContext class from outside.

9.1.35. The ProgressCallback member function

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

This is a Nero callback, responsible for showing progress of the current operation. Here we display the progress in percent, and update a simple progress meter.

9.1.36. The ReadIOCallback member function

```
DWORD NERO_CALLBACK_ATTR CBurnContext::ReadIOCallback (void *pUserData,
                                                       BYTE *pBuffer, DWORD dwLen)
```

ReadIOCallback will be used when PCM is written to CD. It calls fread() to fill the supplied buffer from a file.

9.1.37. The SetMajorPhaseCallback member function

```
void NERO_CALLBACK_ATTR CBurnContext::SetMajorPhaseCallback (void
                  *pUserData, NERO_MAJOR_PHASE phase, void * reserved)
```

This is a Nero callback that prints the change of major phase (e.g. "Start Caching", "Start Writing", "Done Writing"...)

9.1.38. The SetPhaseCallback member function

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

This is a Nero callback that prints the change of phase.

9.1.39. The TranslateNeroToExitCode member function

```
CExitCode CBurnContext::TranslateNeroToExitCode (NEROAPI_BURN_ERROR err)
```

This function performs a simple translation from NeroAPI's burn error into NeroCmd's own EXITCODE.

9.1.40. The TrimStringRight member function

```
void CBurnContext::TrimStringRight (LPSTR psString)
```

This function rids the string of spaces from the right. It is called from the LookForADrive function to trim drive letters and drive names.

9.1.41. The SelectResponse function

```
static NeroUserDlgInOut SelectResponse (const CResponse response[],
                                                    int iSelection = 0)
```

This static helper function is not a class member, but resides in one module with the UserDialog member function. It displays a set of choices and allows the user to move from one choice to another by pressing the arrow keys.

9.1.42. The UserDialog member function

```
NeroUserDlgInOut NERO_CALLBACK_ATTR CBurnContext::UserDialog (void*
                                pUserData, NeroUserDlgInOut type, void *data)
```

Depending on "type" this function prompts the user for a decision that the NeroAPI requires to proceed with the current process. E.g. the user would have to decide whether or not to erase a non-empty CDRW. The actual user input is provided by the SelectResponse function.

pUserData contains a pointer to the CBurnContext instance that initialized the NeroAPI. Thus, functions from CBurnContext can be called during a callback, which otherwise would not be possible.

9.1.43. The WriteFreestyle member function

```
CExitCode CBurnContext::WriteFreestyle (const PARAMETERS & params)
```

This function is responsible for burning a freestyle CD and will be called when the user has supplied one of the "—freestyle" command line parameters. After a few preparations, GetIsoTrack is called to obtain an ISO track based on the given parameters.

The CD stamp information is provided, and a for-loop is used to build the track structure. NeroBurn is called and cleanup functions are performed.

9.1.44. The WriteImage member function

```
CExitCode CBurnContext::WriteImage (const PARAMETERS & params)
```

This function is responsible for burning an ISO image and will be used if the "—image" command line parameter has been supplied.
Writing an image is a straight forward process. A NERO_WRITE_IMAGE structure is created, and the image file name member is filled. Then NeroBurn is called.

9.1.45. The WriteIOCallback member function

```
DWORD NERO_CALLBACK_ATTR CBurnContext::WriteIOCallback (void *pUserData,
                                                        BYTE *pBuffer, DWORD dwLen)
```

WriteIOCallback will used when PCM data is being read from CD.

9.1.46. The WriteIsoAudio member function

```
CExitCode CBurnContext::WriteIsoAudio (const PARAMETERS & params)
```

This function is responsible for burning an ISO/Audio CD or ISO DVD. The size of the CD 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.

The function fills the NERO_WRITE_CD structure with the information the user provided. The burn process is started by calling NeroBurn, passing a pointer to the NERO_WRITE_CD structure.

9.1.47. The WriteNeroErrorLog member function

```
void CBurnContext::WriteNeroErrorLog (const PARAMETERS & params)
```

This function simply writes the standard Nero error log out to a file – "neroerr.txt".

If "—nero_log_timestamp" was specified, the timestamp will be added to the file name prefix.

9.1.48. The WriteVideoCD member function

```
CExitCode CBurnContext::WriteVideoCD (const PARAMETERS & params)
```

This function performs burning Video or Super Video CDs.

A reference to a parameter object is supplied in the function parameter list. Objects for NERO_WRITE_VIDEO_CD, ExitCode, and NERO_ISO_ITEM are instantiated.

Size is calculated from the size of NERO_WRITE_CD, adding the number of tracks in the parameter structure, multiplied by the size of each NERO_VIDEO_ITEM. Memory is allocated for the given size and returned as a pointer to a NERO_WRITE_VIDEO_CD structure.

If not enough memory is available, the application exits with EXITCODE_OUT_OF_MEMORY. Otherwise, the allocated memory is filled with 0-bytes.

The nwvcdSVCD member is set according the m_BurnType member of the parameters class. It becomes true if GetBurnType() returns BURNTYPE_SVCD_CD. The nwcdNumItems member is set to the m_iNumTracks member of parameters. pItem is set to point to NULL;

The temporary path (if supplied by user) gets copied to the appropriate field of the NERO_WRITE_VIDEO_CD structure. At most 256 chars are copied. This is the current size of the NERO_WRITE_VIDEO_CD field. It is ensuredthat the string does not exceed the field size.

A try-catch-combination follows.

The call to GetIsoTrack returns a result code which is compared to EXITCODE_OK. If it differs an exception is thrown, providing the exit code that was just obtained.

In the following loop a temporary pointer to a NERO_VIDEO_ITEM is assigned to each of the nwcdItems of the parameters structure, one by one in every execution of the loop until all tracks have been processed.

Then the name of the source file is copied from the m_psFilename member of the parameters' item structure to the temporary item. The last byte of the string is set to 0 to terminate it. strrchr() searches for the last occurrence of ".", then the extension is compared to the allowed file types "mpg", "mpeg", "jpg", "jpeg" and "avi", thus setting nviItemType to NERO_MPEG_ITEM, NERO_JPEG_ITEM or NERO_NONENCODED_VIDEO_ITEM. If none applies the error log is written and an "unknown file type" exception is thrown.

NeroBurn is called with the required data. Afterwards TranslateNeroToExitCode is called. Memory is freed up and the code returned.

9.2. PARAMETERS

The PARAMETERS class combines all possible flags and additional data.

Additionally, it checks the consistence of the data provided by its setters. There is a Set function and a Get function for every available property. Those properties are derived from command line parameters. There are quite many, and the operations are so similar that we will not list them here.

9.2.1. The PARAMETERS constructor

```
PARAMETERS::PARAMETERS ()
```

All values are set to reasonable defaults.

9.2.2. The ~PARAMETERS destructor

```
PARAMETERS::~PARAMETERS ()
```

The destructor performs no other than the default operation.

9.3. TRACK

```
struct TRACK {
   int m_iTrack;
   LPCSTR m_psFilename;
   NERO_TRACKMODE_TYPE m_Mode;
};
```

This is the Track structure as used by PARAMETERS.

9.4. CErrorLog

9.4.1. The CErrorLog constructor

```
CErrorLog::CErrorLog ()
```

The constructor merely sets the file handle member to NULL, thus marking it as undefined.

9.4.2. The ~CErrorLog destructor

```
CErrorLog::~CErrorLog ()
```

If the file handle member differs from NULL it is passed to the fclose function.

9.4.3. The Open member function

```
bool CErrorLog::Open (LPCSTR psFilename)
```

This function first verifies that the filename, which was passed to this function, is not a NULL value. Then it tries to open that file in write mode. The function returns true if the file could be opened.

9.4.4. The printf member function

```
int CErrorLog::printf (const char * format, ...)
```

This function adds one line to the log file. It provides formatted output of error log entries to the log file.

9.5. EXITCODE enumeration

The EXITCODE enumeration type is used when the application terminates. It indicates what kind of error has occurred, if any.

9.6. CExitCode

CExitCode is a wrapper class for EXITCODE. All functions which are capable of returning errors will return an instance of this class. CExitCode also preserves the textual NeroAPI error.

9.6.1. The CExitCode constructor

```
CExitCode::CExitCode (EXITCODE code)
```

The constructor saves the error code in a member variable and obtains the last error log line from the NeroAPI.

9.6.2. The CExitCode destructor

```
CExitCode::~CExitCode ()
```

In the destructor the error string is freed.

9.6.3. The GetLastError member function

```
LPCSTR GetLastError (void) const
```

This function returns the last error or an empty string.

9.6.4. The GetLastErrorLogLine member function

```
void CExitCode::GetLastErrorLogLine (void)
```

The last error is requested from the NeroAPI and returned.

9.6.5. The GetTextualExitCode member function

```
LPCSTR CExitCode::GetTextualExitCode (void) const
```

This function translates the numeric error code into a textual representation.

9.6.6. The assignment operator for CExitCode classes

```
CExitCode & CExitCode::operator= (const CExitCode & code)
```

This is the assignment operator if the source is another CExitCode.

9.6.7. The assignment operator for EXITCODE enumerations

```
CExitCode & CExitCode::operator= (const EXITCODE code)
```

This will is the assignment operator if the source is a plain EXITCODE constant.

9.6.8. The cast operator

```
operator EXITCODE () const {return m_ExitCode;}
```

The cast operator simply returns the member variable, which is of the EXITCODE enumeration type.

9.7. CResponse

```
struct CResponse {
   LPCSTR m_psButtonText;
   NeroUserDlgInOut m_RetVal;
};
```

This simple structure holds a pair consisting of response text and the corresponding return value.

9.8. CSimpleStringArray

This class implements a simple array of strings as STL vector.

9.8.1. The CSimpleStringArray constructor

```
CSimpleStringArray::CSimpleStringArray ()
```

The constructor merely sets the Boolean m_bOwnData member to true.

9.8.2. The ~CSimpleStringArray destructor

```
CSimpleStringArray::~CSimpleStringArray ()
```

The CSimpleStringArray destructor iterates through the vector and deletes the strings if the m_bOwnData flag is set, which by default is not the case. This prevents deleting strings that do not belong to the object that uses CSimpleStringArray.

9.8.3. The Add member function

```
void CSimpleStringArray::Add (LPSTR psString)
```

This member adds a string to the CSimpleStringArray vector by calling the insert function.

9.9. CFindFiles

CFindFiles is a helper class for enumerating a directory tree for ISO tree handling.

9.9.1. The CFindFiles constructor

```
CFindFiles::CFindFiles (LPCSTR psPath)
```

The constructor takes an LPCSTR path as parameter. The _findfirst function is used to locate the first entry and store its handle. Depending on whether a valid handle was returned the entry is marked as valid or invalid.

9.9.2. The ~CFindFiles destructor

```
CFindFiles::~CFindFiles ()
```

If a handle exists it will be passed to the _findclose function to perform cleanup for files and directories.

9.9.3. The FindNext member function

```
void CFindFiles::FindNext (void)
```

This functions will find the next file or directory entry and set the valid flag.

9.9.4. The GetCreateTime member function

```
inline time_t CFindFiles::GetCreateTime (void) const
```

This function returns the time when a file or directory was created.

9.9.5. The GetName member function

```
inline LPCSTR CFindFiles::GetName (void) const
```

This function returns the name of a file or directory.

9.9.6. The IsSubDir member function

```
inline bool CFindFiles::IsSubDir (void)
```

This function indicates whether or not the entry is a subdirectory by checking if the _A_SUBDIR attribute is set.

9.9.7. The IsValidEntry member function

```
inline bool CFindFiles::IsValidEntry (void)
```

This function returns the m_bValid flag which has been set during construction or changed during FindNext.

10. Version History

Version	Date	Comments
1.0	November 24, 2000	Initial version.
1.1	December 6, 2000	Updates according to the changes in NeroAPI (version 5.0.3.4). - UserDialog callback function has been modified to support new DLG_NON_EMPTY_CDRW callback. - Two new command line parameters have been added. One is –detect_non_empty_cdrw and the other is –cd_text. - The code around NeroImportIsoTrack has been updated according to the NeroAPI changes. Error handling has been improved.
1.2	April 5, 2001	Updates according to the changes in NeroAPI (version 5.5.0.6). - Changed project name from NeroBATCH to NeroCmd
1.3	April 23, 2001	Updates according to the changes in NeroAPI (version 5.5.1.4). - Added two cmd line options according to the two new NBF_ constants - Added a new EXITCODE_ and a corresponding error message for a new NEROAPI_INIT_DEMOVERSION_EXPIRED constant - Changed NeroAPI initialization to utilize the new shared API feature
1.4	October 28, 2001	Added handling of the –dvd parameter. The DVD burning feature was introduced with NeroAPI 5.5.4.4.
1.5	December 3, 2001	Several Bugfixes: - UserDialog-Input wasn't evaluated properly. - Buffer was overwritten occasionally, causing the application to terminate with an error. Improved Log File Handling Added –force_erase_cdrw command line parameter to delete CDRW without user interaction when combined with –detect_non_empty_cdrw Added –nero_log_timestamp to keep multiple versions of the NeroAPI error log for batch runs.
1.6	November 22, 2002	Improved file and directory handling. Added --driveinfo for information about a particular drive --listformats for listing available audio formats --get_speeds for read and write speeds --recursive for handling of subdirectories