NeroCBUI

v1.2.1.11

User's Manual

NeroCBUI 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.

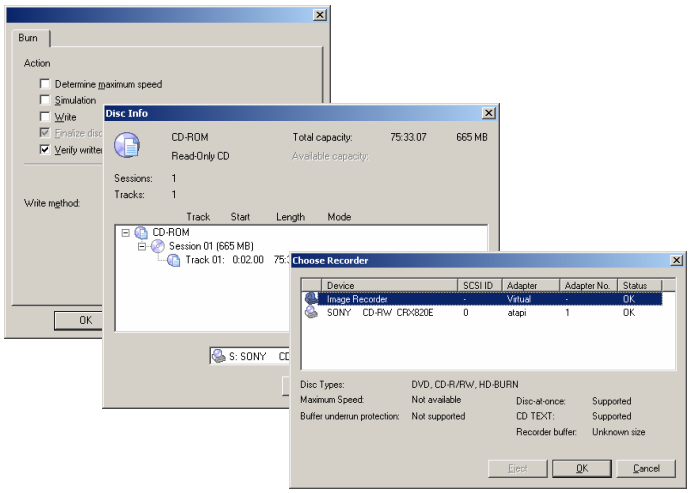
3. Introduction

3.1. Motivation

Many applications developed by software engineers world-wide are based on the *NeroAPI* engine. All these applications bring their own GUI (Graphical User Interface) for driving the burn process and providing related utility functions (erase disc, disc info).

Ahead Software decided to publish *NeroCBUI*, a collection of GUI components built to support those who develop CD/DVD burning applications with the *NeroAPI*.

NeroCBUI contains a number of commonly required dialogs and controls for creating CD and DVD burning applications. *NeroCBUI* was engineered by extracting well-tested implementations for each dialog from Ahead's sources and reassembling them in one single module.



NeroCBUI can be included in applications that use the *NeroAPI*.

Advantages of using *NeroCBUI* in your applications:

- Ready-to-use professional appearance
- Maturity through extensive testing in different environments
- Binary reuse reduces source code size and number of coding errors
- Peer support through quickly broadening user base
- Savings in time for design, implementation and testing

3.2. Overview

This paper will provide a comprehensive list of available options for *NeroCBUI* and examples of use. *NeroCBUI* can perform the following tasks:

NeroCBUI provides the following “ready-to-use” dialogs:

- Burn progress
- Burn settings
- Recorder selection
- Session selection
- Disc info
- Erase rewritable disc
- Disc image file selection and information

NeroCBUI is kept compatible to older *NeroAPI* versions. If it uses features that are only available in a newer *NeroAPI* version, it checks the version of the installed *NeroAPI* first. It does not try to use new features that are not available.

NeroCBUI is implemented in such a way that it can be used by a non-MFC (Microsoft Foundation Classes) application, although *NeroCBUI* itself is based on the MFC.

NeroCBUI supports multiple languages. The language can be selected through its API.

3.3. Requirements

NeroCBUI will work on any platform which is fit for hosting ***Nero* 6.0.0.0**. *Nero* 6.0.0.0 or a later version **needs to be installed and licensed** prior to using *NeroCBUI*.

NeroCBUI will not work with Linux.

You can obtain the latest version of *Nero* from http://www.nero.com.

Note: Not all features of *NeroCBUI* have already been available with *Nero* 6.0.0.0. Some interfaces have been added in more recent versions. If you want to assure that your *NeroCBUI*-based applications work with all versions of *Nero 6*, please do not rely on more recent features.

The following chart indicates the required *Nero* version for a given *NeroCBUI* version. Also, in the description of objects and interfaces in this document the individual items are tagged with the *NeroCBUI* version with which they were introduced.

***NeroCBUI* version**	***Nero* version**
1.0.0.7	6.0.0.0
1.0.0.10	6.0.0.6
1.0.0.11	6.0.0.8
1.0.0.12	6.0.0.9
1.0.0.13	6.0.0.10
1.0.0.14	6.0.0.13
1.0.0.15	6.0.0.14
1.2.0.1	6.0.0.17
1.2.0.3	6.0.0.18
1.2.1.1	6.0.0.20
1.2.1.3	6.0.0.22
1.2.1.4	6.0.0.24
1.2.1.5	6.0.0.25
1.2.1.6	6.0.0.27
1.2.1.8	6.3.0.0
1.2.1.9	6.3.0.4
1.2.1.11	6.3.0.6

4. Using NeroCBUI

In the following we will point out which steps are required to use *NeroCBUI* from your applications. Also, we will give a brief introduction to the TestNeroCBUI application which demonstrates the use of *NeroCBUI*.

4.1. General steps

4.1.1. General Includes

In an MFC project add the following to stdafx.h:

```
#include "NeroAPIGlue.h"
#include <atlbase.h>
#include "NeroCBUI.h"
```

Also, include the library NeroAPIGlue.lib or NeroAPIGlueRT.lib in your project settings.

The inclusion of NeroAPIGlue.h is a general requirement because we cannot use *NeroCBUI* without initializing the *NeroAPI* first. *NeroCBUI* assumes that NeroInit has already been called by the *NeroCBUI* client at some point in time. *NeroCBUI* itself will not initialize the *NeroAPI*.

The inclusion of atlbase.h is required to access the COM functions.

NeroCBUI.h contains the interface of *NeroCBUI*.

4.1.2. NeroCBUIGlue

In your implementation include NeroCBUIGlue.h:

```
#include "NeroCBUIGlue.h"
```

It is required for creating an instance of the NeroAPICompatibilityManager during initialization.

4.1.3. Initialization

First, an object for init settings is required.

```
CComPtr<IInitSettings2> m_pInitSettings;
```

A call to CoInitializeEx is required to initialize the COM library for use.

```
CoInitializeEx (NULL, COINIT_APARTMENTTHREADED);
```

In the *NeroAPI* setup part the *NeroAPI* glue code must be initialized with a call to NeroAPIGlueConnect. Then a call to NeroInit is required. More information on initializing the *NeroAPI* can be found in the *NeroAPI* documentation.

After successful initialization, the InitSettings object of *NeroCBUI* can be created

```
if (SUCCEEDED (m_pInitSettings.CoCreateInstance (__uuidof
                                                    (InitSettings))))
```

Note: The InitSettings object must be kept alive as long as *NeroCBUI* is used. *NeroCBUI* should be uninitialized before the *NeroAPI* is uninitialized.

The *NeroAPI* module handle must be passed to *NeroCBUI*.

Each application that uses *NeroCBUI* has to create an instance of the **NeroAPICompatibilityManager** and pass it to *NeroCBUI* through the IInitSettings3::CompatibilityMgr function.

This is essential if a *NeroAPI* is being used that is older than the version that has been available during compilation.

```
m_pInitSettings->put_HMODULE (NeroAPIGlueGetModuleHandle ());
m_pInitSettings->put_AppTitle ((const signed char*)"Test NeroCBUI");

CComPtr<IUnknown> pCompatibilityMgr;
pCompatibilityMgr.Attach(NeroCBUIGlueGetCompatibilityMgr(
                                    NeroAPIGlueGetModuleHandle(),NULL));
m_pInitSettings->put_CompatibilityMgr(pCompatibilityMgr);
```

4.1.4. Usage Example

Now *NeroCBUI* is ready for use.

The following little piece of code is sufficient for selecting a Nero image file and obtaining information on the contents.

```
static char szFilter[] = "Nero Images (*.nrg)|*.nrg|ISO Images
(*.iso)|*.iso|All Files (*.*)|*.*||";
CFileDialog fd (TRUE,"nrg",NULL, OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,
szFilter, this);

if (IDOK == fd.DoModal ())
{
  CComPtr<IImageInfoDlg> pParam;
  if (SUCCEEDED (pParam.CoCreateInstance (__uuidof
                                          (ImageInfoDlgParam))))
  {
    CComBSTR bstrFilename (fd.GetPathName ());
    pParam->put_ImageFilename (bstrFilename);
    IUnknown * pUnk = pParam;
    OleCreatePropertyFrame (m_hWnd, 0, 0, L"Title", 1, &pUnk, 1,
                  (LPCLSID) &__uuidof (ImageInfoDlg), 0, NULL, NULL);
  }
}
```

4.1.5. Cleanup

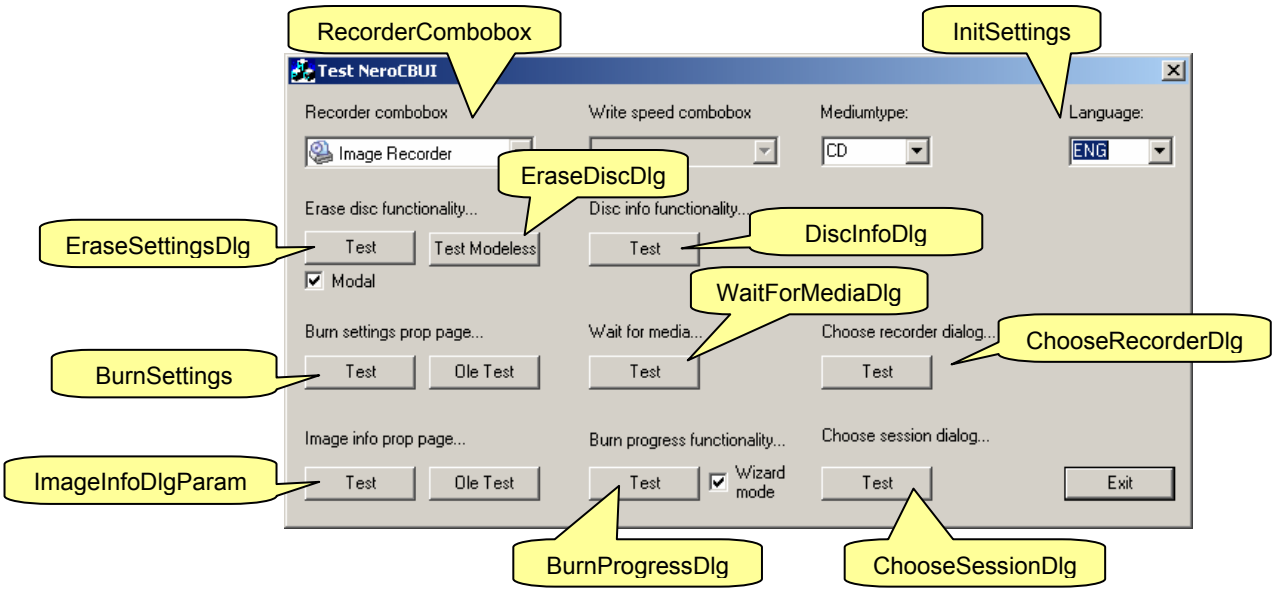
Cleanup is rather straight forward. Calls to NeroDone and NeroAPIGlueDone will release the *NeroAPI* while a call to UnInitialize will close the COM library.

No special cleanup is required for *NeroCBUI*. *NeroCBUI* is unloaded when the InitSettings object is destroyed. However, the InitSettings object should be destroyed before the NeroAPI is uninitialized.

```
NeroDone ();
NeroAPIGlueDone ();
CoUninitialize ();
```

4.2. TestNeroCBUI Application

The TestNeroCBUI application uses all dialogs that are made available through *NeroCBUI*.



The yellow labels on the screenshot indicate the use of *NeroCBUI* objects.

TestNeroCBUI consists of five classes:

CApp, CBurnDlg, CDlg, CPropertyPageBurnSettings and CPropertyPageImageInfo.

CApp contains the initialization code as described in the previous chapter in the InitInstance and ExitInstance methods. Also the implementations of IdleCallback and UserDialog callback are part of CApp.

CBurnDlg is a burn dialog class that is used for burn progress testing.

CDlg is the main dialog for the application. It contains all the button handler functions. Those are named corresponding to their function. For example, the Burn Settings property page function is named OnTestBurnSettings.

The “Ole Test” functions use OleCreatePropertyFrame to create a new property frame, a property sheet dialog box.

CPropertyPageBurnSettings is derived from CPropertyPage and used by the OnTestBurnSettings function of CDlg.

CPropertyPageImageInfo is derived from CPropertyPage and used by the OnTestImageInfo function of CDlg.

Objects of class CPropertyPage represent individual pages of a property sheet, otherwise known as a tab dialog box

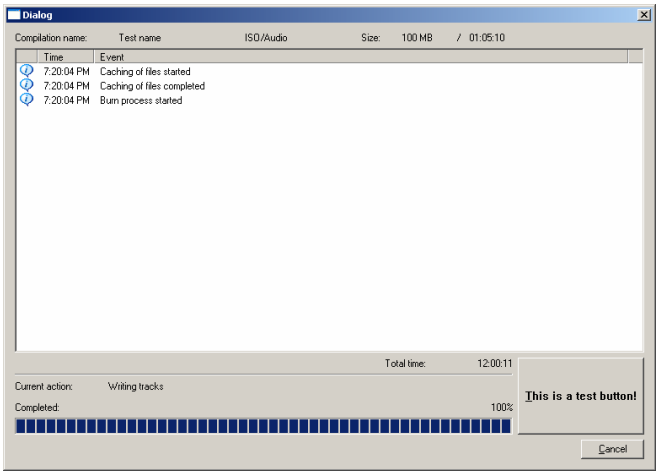
5. Objects and Interfaces

5.1. NEROCBUILib Library Objects

5.1.1. BurnProgressDlg Object

The progress window object should first be assigned its working parameters through the IBurnProgressDlg interface and then the actual Win32 window should be created through IChildWindow.

After the window is in place, IBurnProgressDlg should be used again to initiate the actual burn process.




```
coclass BurnProgressDlg
{
      [default] interface IBurnProgressDlg3;
      interface IBurnProgressDlg2;
      interface IBurnProgressDlg;
      interface IModelessDialog;
      interface IBurnProgressDlgEx;
};
```

Identifier	Introduced in *NeroCBUI* version
IBurnProgressDlg2	1.0.0.15
IBurnProgressDlg3	1.2.0.2

Making the BurnProgressDlg work in an MFC project

1. Create a new CDialog derived class. Usually this will be done with the help of the wizard. The dialog template should be empty, i.e. contain no controls. Its size will mandate the initial size of the BurnProgressDlg.
2. OnInitDialog should be used to create a *NeroCBUI* BurnProgressDlg object and set its properties. This is the place to create the actual window with IModelessDialog::Create method.
3. It is essential to trigger the WM_GETMINMAXINFO message if NULL was passed as pRect to IModelessDialog::Create. If it is non-NULL, WM_GETMINMAXINFO triggering is not necessary.
4. PreTranslateMessage should be overriden to allow the BurnProgressDlg to handle its own dialog messages.
5. OnCancel should be handled (even though there are no controls) so that dialog dismissal is handled properly.
6. OnSize should be handled to allow the sizing of the BurnProgressDlg.
7. Finally, the burn process should be started after the dialog is initialized. It is best to post a user message from the OnInitDialog so that OnInitDialog can return. When the message is processed, it will initiate the burn process. Due to the MFC architecture, even though we posted a message from OnInitDialog, the dialog will not yet be shown at the time of user message processing so we **must** show and update the window manually.

5.1.2. BurnSettings Object

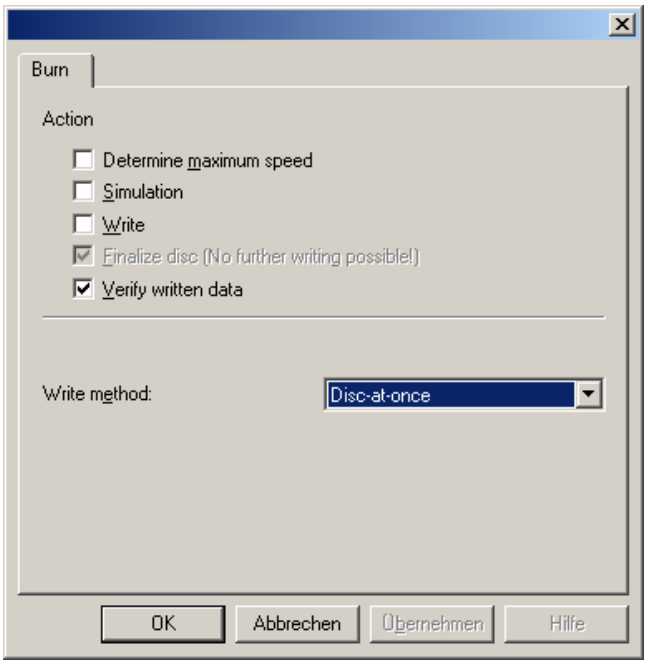
This is the burn settings object. It is directly creatable by the client. The client can set the initial burn settings for this object.

```
coclass BurnSettings
{
      [default] interface IBurnSettings;
};
```

5.1.3. BurnSettingsDlg Object

The burn settings window object should be assigned its working parameters first through IBurnSettingsDlg and INERO_SCSI_DEVICE_INFO interfaces.

Then the client should create the actual Win32 window through IChildWindow interface.




```
coclass BurnProgressDlg
{
      [default] interface IBurnProgressDlg3;
      interface IBurnProgressDlg2;
      interface IBurnProgressDlg;
      interface IModelessDialog;
      interface IBurnProgressDlgEx;
};
```

5.1.4. BurnSettingsDlgParam Object

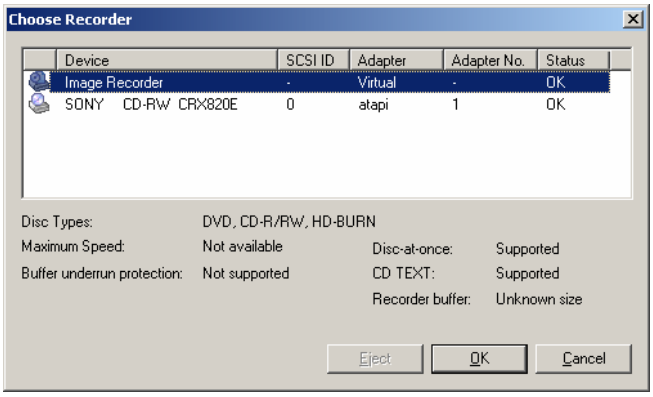
Parameter object for the Burn Settings dialog.

```
coclass BurnSettingsDlgParam
{
      [default] interface IBurnSettingsDlg;
      interface INERO_SCSI_DEVICE_INFO;
};
```

Identifier	Introduced in *NeroCBUI* version
BurnSettingsDlgParam	1.0.0.15

5.1.5. ChooseRecorderDlg Object

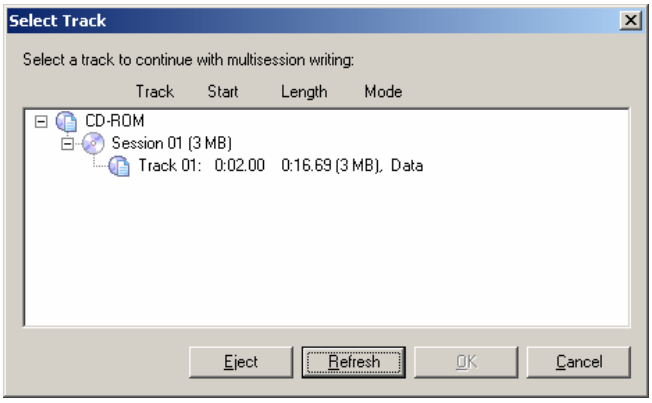
The choose recorder dialog object is first assigned the initial recorder selection (if needed) through the INERO_SCSI_DEVICE_INFO interface and then the IModalDialog interface should be used to display the modal dialog.




```
coclass ChooseRecorderDlg
{
      [default] interface INERO_SCSI_DEVICE_INFO;
      interface IModalDialog;
};
```

5.1.6. ChooseSessionDlg Object

The choose session dialog offers the user currently available sessions for selection to continue multisession writing.



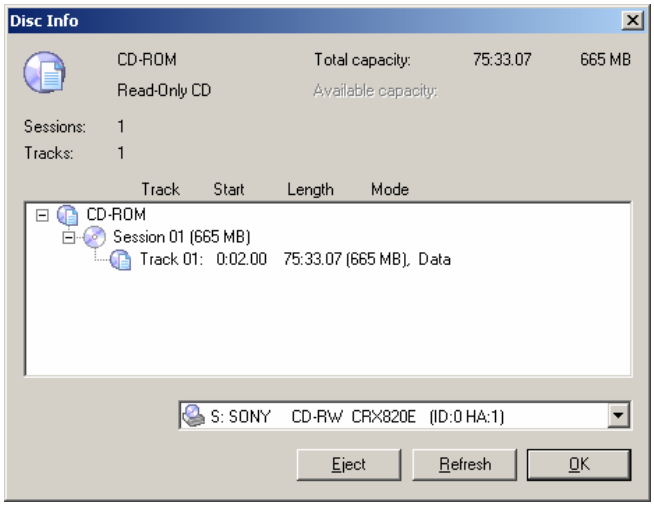

```
coclass ChooseSessionDlg
{
      [default] interface IChooseSessionDlg;
      interface IModalDialog;
      interface INERO_SCSI_DEVICE_INFO;
};
```

Identifier	Introduced in *NeroCBUI* version
ChooseSessionDlg	1.0.0.8

5.1.7. DiscInfoDlg Object

The disc info dialog object is first assigned its working parameters and then it is displayed as a modal dialog box.

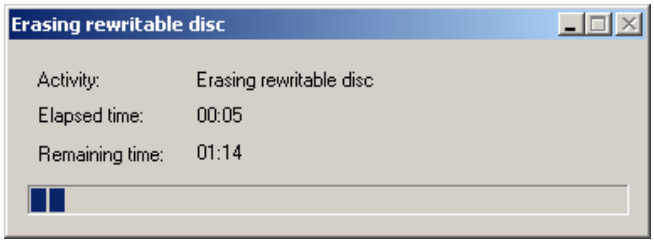
NERO_SCSI_DEVICE_INFO can be both set and obtained from the object.




```
coclass DiscInfoDlg
{
      [default] interface INERO_SCSI_DEVICE_INFO;
      interface IModalDialog;
};
```

5.1.8. EraseDiscDlg Object

The erase disc dialog object is first assigned its working parameters and then it is displayed as a modal or non-modal dialog box.



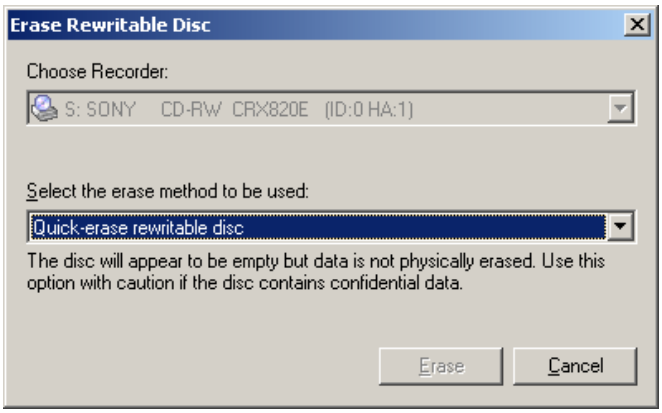

```
coclass EraseDiscDlg
{
      [default] interface IEraseDiscDlg3;
      interface IEraseDiscDlg2;
      interface IEraseDiscDlg;
      interface INEROAPI_CDRW_ERASE_MODE;
      interface INERO_SCSI_DEVICE_INFO;
      interface IModalDialog;
      interface IModelessDialog;
};
```

Identifier	Introduced in *NeroCBUI* version
INERO_SCSI_DEVICE_INFO	1.0.0.11

5.1.9. EraseSettingsDlg Object

The erase settings dialog object is first assigned its working parameters. Those are contained in the INERO_SCSI_DEVICE_INFO and INEROAPI_CDRW_ERASE_MODE interfaces.

Eventually the dialog is displayed through IModalDialog.

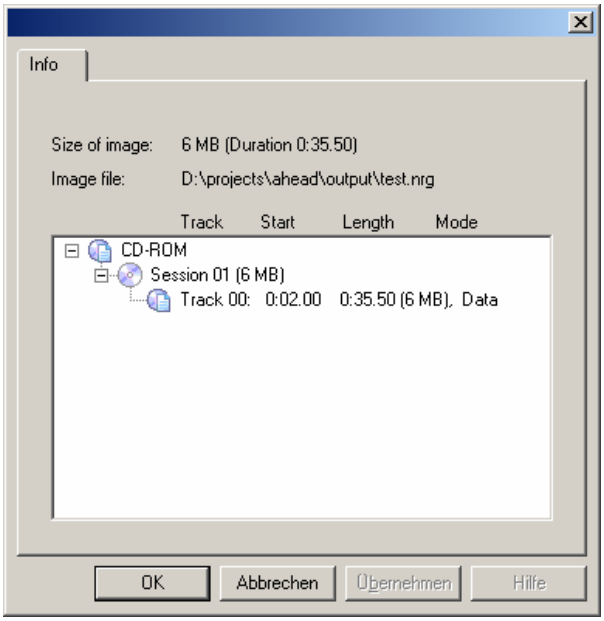



```
coclass EraseSettingsDlg
{
      [default] interface INEROAPI_CDRW_ERASE_MODE;
      interface INERO_SCSI_DEVICE_INFO;
      interface IModalDialog;
      interface IEraseSettingsDlg2;
      interface IEraseSettingsDlg;
};
```

Identifier	Introduced in *NeroCBUI* version
IEraseSettingsDlg	1.0.0.11
IEraseSettingsDlg2	1.2.1.11

5.1.10. ImageInfoDlg Object

The image info dialog provides information about an image file.




```
coclass ImageInfoDlg
{
      [default] interface IImageInfoDlg;
      interface IMfcPropertyPage;
};
```

5.1.11. ImageInfoDlgParam Object

Parameter object for the Image Info dialog.

```
coclass ImageInfoDlgParam
{
      [default] interface IImageInfoDlg;
};
```

Identifier	Introduced in *NeroCBUI* version
ImageInfoDlgParam	1.0.0.15

5.1.12. InitSettings Object

This object should be created first to receive the module handle for *NeroCBUI* to use. The language can also be set here. This object has to persist as long as *NeroCBUI* is used. On destruction of this object *NeroCBUI* is deinitialized.

```
coclass InitSettings
{
      [default] interface IInitSettings3;
      interface IInitSettings2;
      interface IInitSettings;
      interface IInformation;
};
```

5.1.13. RecorderCombobox Object

This is the recorder combobox object. The client should create the object and acquire the IRecorderCombobox and INERO_SCSI_DEVICE_INFO interfaces first to set the working parameters.

Then the client should create the actual Win32 window through IChildWindow interface. The selected recorder can be retrieved using the INERO_SCSI_DEVICE_INFO interface.

```
coclass RecorderCombobox
{
      [default] interface IRecorderCombobox3;
      interface IRecorderCombobox2;
      interface IRecorderCombobox;
      interface INERO_SCSI_DEVICE_INFO;
      interface IChildWindow;
};
```

5.1.14. SpeedCombobox Object

```
coclass SpeedCombobox
{
      [default] interface ISpeedCombobox2;
      interface ISpeedCombobox;
      interface INERO_SCSI_DEVICE_INFO;
      interface IChildWindow;
};
```

Identifier	Introduced in *NeroCBUI* version
SpeedCombobox	1.0.0.15

5.1.15. UserDlgHandler Object

Allows the external use of *NeroCBUI's* own user dialog handler.

```
coclass UserDlgHandler
{
      [default] interface IUserDlgHandler;
};
```

Identifier	Introduced in *NeroCBUI* version
UserDlgHandler	1.2.0.1

5.1.16. WaitForMediaDlg Object

```
coclass WaitForMediaDlg
{
      [default] interface IWaitForMediaDlg;
};
```

Identifier	Introduced in *NeroCBUI* version
WaitForMediaDlg	1.0.0.15

5.2. NEROCBUILib Library Interfaces

5.2.1. BURN_PROGRESS_GUI_FLAGS Enumeration

```
typedef [v1_enum] enum _tagBURN_PROGRESS_GUI_FLAGS
{
      BPGF_TOTAL_TIME                                 = 1L<< 0,
      BPGF_CURRENT_ACTION                             = 1L<< 1,
      BPGF_USED_BUFFER                                = 1L<< 2,
      BPGF_CANCEL_BUTTON                              = 1L<< 3,
      BPGF_CLOSE_DISCARD_BUTTON                       = 1L<< 4,

      BPGF_PRINT_BUTTON                               = 1L<< 6,
      BPGF_SAVE_BUTTON                                = 1L<< 7,
}
BURN_PROGRESS_GUI_FLAGS;
```

5.2.2. BURN_SETTINGS_GUI_FLAGS Enumeration

```
typedef [v1_enum] enum _tagBURN_SETTINGS_GUI_FLAGS
{
      BSGF_DETERMINE_MAXIMUM_SPEED              = 1L<< 0,
      BSGF_SIMULATION                           = 1L<< 1,
      BSGF_WRITE                                = 1L<< 2,
      BSGF_FINALIZE_CD                          = 1L<< 3,
      BSGF_WRITE_SPEED                          = 1L<< 4,
      BSGF_WRITE_METHOD                         = 1L<< 5,
      BSGF_BURN_PROOF                           = 1L<< 6,
      BSGF_PROGRESS                             = 1L<< 7,
      BSGF_VERIFY_WRITTEN_DATA                  = 1L<< 8,
}
BURN_SETTINGS_GUI_FLAGS;
```

5.2.3. IBurnCallbacks Interface

This interface is used solely for the purpose of giving the client a chance to handle the callback notifications from the *NeroAPI* engine. If the user wants them he should implement this interface and pass its pointer where appropriate (see the IBurnProgressDlg interface below).

```
interface IBurnCallbacks : IUnknown
{
      HRESULT OnDoneBurn([in] NEROAPI_BURN_ERROR burnError);

      HRESULT Progress([in] DWORD dwProgressInPercent, [out, retval] BOOL
      * pbRetVal);
      HRESULT SubtaskProgress([in] DWORD dwProgressInPercent, [out,
      retval] BOOL * pbRetVal);
      HRESULT AddLogLine([in] NERO_TEXT_TYPE type, [in] BSTR
      bstrLogLine);
      HRESULT SetPhase([in] BSTR bstrPhase);
      HRESULT SetMajorPhase([in] NERO_MAJOR_PHASE majorPhase);
      HRESULT Aborted ([out, retval] BOOL * pbRetVal);
      HRESULT DisableAbort ([in] BOOL bAbortEnabled);
}
```

5.2.4. IBurnProgressDlg Interface

After creating the BurnProgressDlg object and creating the actual child window through IChildWindow interface, the following interface is used to

1. set working parameters to the object with SetParameters method
2. to initiate the burn process itself through one of the two variants of burn methods.

The IBurnCallbacks interface pointer can be NULL. If non-NULL, it is assumed that the client wants to be notified during the burn process about the major developments.

The Cancel method is used to cancel the burn process.

```
interface IBurnProgressDlg : IUnknown
{
	[propput] HRESULT GUIFlags ([in] DWORD dwFlags);

	[propput] HRESULT CompilationName ([in] BSTR bstrCompilationName);
	[propput] HRESULT CompilationType ([in] BSTR bstrCompilationType);
	[propput] HRESULT CompilationSize ([in] BSTR bstrCompilationSize);
	[propput] HRESULT CompilationTime ([in] BSTR bstrCompilationTime);

	HRESULT Burn([in] const LPNERO_SCSI_DEVICE_INFO pDeviceInfo, [in]
	NERO_CD_FORMAT cdFormat, [in] const void * pWriteCD, [in]
	IBurnSettings * pBurnSettings, [in] IBurnCallbacks *
	pBurnCallbacks);
	HRESULT Burn2([in] const LPNERO_SCSI_DEVICE_INFO pDeviceInfo, [in]
	NERO_CD_FORMAT cdFormat, [in] const void * pWriteCD, [in] DWORD
	dwBurnFlags, [in] DWORD dwSpeed, [in] IBurnCallbacks *
	pBurnCallbacks);

	HRESULT Cancel ();
	HRESULT CanCloseParent ([out, retval] BOOL * pbCanClose);

	HRESULT AllocateProgressBar ([in] DWORD dwPreProgress, [in] DWORD
	dwPostProgress);
	HRESULT SetProgress([in] DWORD dwProgressInPercent);
	HRESULT AddLogLine([in] NERO_TEXT_TYPE type, [in] BSTR
	bstrLogLine);
	HRESULT SetPhase([in] BSTR bstrPhase);
};
```

5.2.5. IBurnProgressDlg2 Interface

```
interface IBurnProgressDlg2 : IBurnProgressDlg
{
	[propput] HRESULT WizardMode ([in] BOOL bWizardMode);
}
```

Identifier	Introduced in *NeroCBUI* version
IBurnProgressDlg2	1.0.0.15

5.2.6. IBurnProgressDlg3 Interface

```
interface IBurnProgressDlg3 : IBurnProgressDlg2
{
      HRESULT AddCustomControl ([in]
      BURN_PROGRESS_CUSTOM_CONTROL_POSITION bpccpWhere, [in]
      IExternalChildWindow * pControl);
};
```

Identifier	Introduced in *NeroCBUI* version
IBurnProgressDlg3	1.2.0.2

5.2.7. IBurnProgressDlgCallbacks Interface

Callback functions for burn progress.

```
interface IBurnProgressDlgCallbacks : IUnknown
{
      HRESULT OnCancel ([out, retval] BOOL * pbAllowCancel);
      HRESULT OnDiscard ();
      HRESULT OnClose ();
}
```

5.2.8. IBurnProgressDlgEx Interface

IBurnProgressDlgEx allows the setting of a client-implemented callback interface.

```
interface IBurnProgressDlgEx : IUnknown
{
      [propput] HRESULT IBurnProgressDlgCallbacks ([in]
      IBurnProgressDlgCallbacks * pCallbacks);
}
```

Identifier	Introduced in *NeroCBUI* version
IBurnProgressDlgEx	1.0.0.8

5.2.9. IBurnSettings Interface

The BurnSettings object is accessed through this interface. It is used solely to store burn settings information. The object is normally created by the client.

```
interface IBurnSettings : IUnknown
{
      [propget] HRESULT BurnFlags([out, retval] DWORD * pdwBurnFlags);
      [propput] HRESULT BurnFlags([in] DWORD dwBurnFlags);

      [propget] HRESULT Speed([out, retval] DWORD * pdwSpeed);
      [propput] HRESULT Speed([in] DWORD dwSpeed);
};
```

5.2.10. IBurnSettingsDlg Interface

Set the working parameters for the burn settings window. The IBurnSettings interface points to the object created by the client.

```
interface IBurnSettingsDlg : IUnknown
{
      [propput] HRESULT BurnSettings ([in] IBurnSettings *
      pBurnSettings);
      [propput] HRESULT GUIFlags ([in] DWORD dwFlags);
      [propput] HRESULT MediaType ([in] NERO_MEDIA_TYPE mediaType);
};
```

5.2.11. IChildWindow Interface

```
interface IChildWindow : IUnknown
{
      HRESULT Subclass ([in] HWND hWnd, [out, retval] BOOL * pbRetVal);
}
```

5.2.12. IChooseSessionDlg Interface

```
interface IChooseSessionDlg: IUnknown
{
      [propget] HRESULT TrackNumber ([out, retval] DWORD * pdwVal);
};
```

Identifier	Introduced in *NeroCBUI* version
IChooseSessionDlg	1.0.0.8

5.2.13. IEraseDiscCallbacks Interface

```
interface IEraseDiscCallbacks : IUnknown
{
      HRESULT OnEraseDone ([in] IEraseDiscDlg * pEraseDiscDlg);
};
```

Identifier	Introduced in *NeroCBUI* version
IEraseDiscCallbacks	1.0.0.11

5.2.14. IEraseDiscDlg Interace

This method allows the client to specify the device handle of the recorder.

The get operation return the result of the erase operation either as integer value or string.

```
interface IEraseDiscDlg : IUnknown
{
      [propput] HRESULT NERO_DEVICEHANDLE ([in] NERO_DEVICEHANDLE
      aDeviceHandle);
      [propget] HRESULT ErasingResult ([out, retval] int*
      piErasingResult);
      [propget] HRESULT ErasingResultString ([out, retval] BSTR *
      pbstrErasingResult);
}
```

5.2.15. IEraseDiscDlg2 Interface

Used to specify additional flags for NeroEraseDisc.

```
interface IEraseDiscDlg2 : IEraseDiscDlg
{
      [propput] HRESULT ErasingFlags ([in] DWORD dwFlags);
}
```

Identifier	Introduced in *NeroCBUI* version
IEraseDiscDlg2	1.0.0.7

5.2.16. IEraseDiscDlg3 Interface

IEraseDiscDlg3 allows EraseDiscDlg to support a modeless dialog together with a modal dialog.

```
interface IEraseDiscDlg3 : IEraseDiscDlg2
{
      [propput] HRESULT Callbacks ([in] IEraseDiscCallbacks *
      pCallbacks);
}
```

Identifier	Introduced in *NeroCBUI* version
IEraseDiscDlg3	1.0.0.11

5.2.17. IEraseDiscDlg4 Interface

```
interface IEraseDiscDlg4 : IEraseDiscDlg3
{
      [propput] HRESULT NeverShowDialogBox ([in] BOOL bDontShow);
}
```

Identifier	Introduced in *NeroCBUI* version
IEraseDiscDlg4	1.0.0.15

5.2.18. IEraseSettingsDlg Interface

```
interface IEraseSettingsDlg : IUnknown
{
      [propput] HRESULT ModelessErase ([in] BOOL bModelessErase);
      [propput] HRESULT Callbacks ([in] IEraseDiscCallbacks *
      pCallbacks);
}
```

Identifier	Introduced in *NeroCBUI* version
IEraseSettingsDlg	1.0.0.11

5.2.19. IEraseSettingsDlg2 Interface

This interface is used to retrieve the HWND of the modeless dialog.

```
interface IEraseSettingsDlg2 : IEraseSettingsDlg
{
      [propget] HRESULT ModelessDlgHWND ([out, retval] HWND*
       phModelessDlgHWND);
}
```

Identifier	Introduced in *NeroCBUI* version
IEraseSettingsDlg2	1.2.1.11

5.2.20. IExternalChildWindow Interface

```
interface IExternalChildWindow : IUnknown
{
      HRESULT Create ([in] HWND hWndParent, [out, retval] HWND * phWnd);
      [propget] HRESULT MinSize ([out, retval] SIZE * pSize);
      [propget] HRESULT MaxSize ([out, retval] SIZE * pSize);
}
```

Identifier	Introduced in *NeroCBUI* version
IExternalChildWindow	1.2.0.2

5.2.21. IImageInfoDlg Interface

```
interface IImageInfoDlg: IUnknown
{
      [propput] HRESULT ImageFilename ([in] BSTR bstrFilename);
}
```

5.2.22. IInformation Interface

```
interface IInformation : IUnknown
{
      [propget] HRESULT Version ([out] WORD * pwMajHi, [out] WORD *
      pwMajLo, [out] WORD * pwMinHi, [out] WORD * pwMinLo);
      [propget] HRESULT InstallFolder ([out, retval] BSTR * pbstrFolder);
}
```

5.2.23. IInitSettings Interface

Interface for initializing *NeroCBUI*. Before *NeroCBUI* can be used the application has to give a handle to *NeroAPI* to *NeroCBUI*. Additionally the language can be set with this interface.

The client should pass the *NeroAPI* module handle through this property. If no handle or the NULL handle is passed, *NeroCBUI* will try to get the handle of the already loaded NeroAPI dll.

The LanguageCode property provides a way to set the language code.

```
interface IInitSettings : IUnknown
{
      [propput] HRESULT HMODULE ([in] HMODULE hNeroAPIModule);
      [propput] HRESULT LanguageCode ([in] const signed char *
      psLanguageCode);
}
```

Identifier	Introduced in *NeroCBUI* version
IInitSettings	1.0.0.8

5.2.24. IInitSettings2 Interface

```
interface IInitSettings2 : IInitSettings
{
      [propput] HRESULT AppTitle ([in] const signed char * psAppTitle);
}
```

Identifier	Introduced in *NeroCBUI* version
IInitSettings2	1.0.0.15

5.2.25. IMfcPropertyPage Interface

```
interface IMfcPropertyPage : IUnknown
{
      [propget] HRESULT AFX_OLDPROPSHEETPAGE ([out, retval]
      LPAFX_OLDPROPSHEETPAGE * ppPSP);
}
```

5.2.26. IModelessDialog

This interface implements the functionality of a modeless dialog.

```
interface IModelessDialog : IUnknown
{
      HRESULT Create ([in] HWND hWndParent, [in] LPCRECT pRect, [out,
      retval] HWND * phWndRet);
      HRESULT Destroy ();
      HRESULT TranslateAccelerator ([in] MSG * pMsg);
}
```

5.2.27. IModalDialog Interface

All objects that display modal dialogs inherit from this interface. It provides the simple mechanism of invoking a modal dialog, specifying the parent window and getting back the result in terms of IDOK, IDCANCEL, etc.

```
interface IModalDialog : IUnknown
{
      HRESULT DoModal ([in] HWND hWndParent, [out, retval] int *
      piRetVal);
}
```

5.2.28. INEROAPI_CDRW_ERASE_MODE Interface

This method allows the client to change the default initial selection for the erase mode.

After the dialog has been dismissed, the client can get the selected erase mode with this method.

```
interface INEROAPI_CDRW_ERASE_MODE : IUnknown
{
      [propput] HRESULT NEROAPI_CDRW_ERASE_MODE ([in]
      NEROAPI_CDRW_ERASE_MODE eraseMode);
      [propget] HRESULT NEROAPI_CDRW_ERASE_MODE ([out, retval]
      NEROAPI_CDRW_ERASE_MODE * pEraseMode);
}
```

5.2.29. INERO_SCSI_DEVICE_INFO Interface

This interface contains the common functionality of getting and setting the NERO_SCSI_DEVICE_INFO information. Some objects inheriting from this interface might not need the get part of this interface. In those cases, the get method will return E_NOTIMPL.

The NERO_SCSI_DEVICE_INFO put method allows the client to specify a particular recorder to be the default initial selection. The default is no selection and the same effect is accomplished if pDeviceInfo is NULL or if it is non-NULL but match could not be found (probably bogus NERO_SCSI_DEVICE_INFO).

The NERO_SCSI_DEVICE_INFO get method should be invoked after the dialog has been displayed and dismissed to retrieve the user's selection. If pDeviceInfo is NULL, the method is a no-op. If it is non-NULL, it is assumed that it points to at least sizeof(NERO_SCSI_DEVICE_INFO) big buffer which will be filled with the user's current selection.

```
interface INERO_SCSI_DEVICE_INFO : IUnknown
{
      [propput] HRESULT NERO_SCSI_DEVICE_INFO ([in] const
      LPNERO_SCSI_DEVICE_INFO pDeviceInfo);

      [propget] HRESULT NERO_SCSI_DEVICE_INFO ([out, retval]
      LPNERO_SCSI_DEVICE_INFO * ppDeviceInfo);
}
```

5.2.30. IRecorderCombobox Interface

After creating the RecorderCombobox object and creating the actual child window through the IChildWindow interface, this interface is used to set the working parameters.

If the client is interested in receiving events from this combobox it should implement the IRecorderComboboxCallbacks interface and pass the callbacks pointer.

```
interface IRecorderCombobox : IUnknown
{
      [propput] HRESULT Callbacks ([in] IRecorderComboboxCallbacks *
      pCallbacks);
}
```

5.2.31. IRecorderCombobox2 Interface

An extension to the IRecorderCombobox interface which mainly supports the addition of custom entries.

```
interface IRecorderCombobox2 : IRecorderCombobox
{
      HRESULT Refresh ([in] BOOL bRescan);
      HRESULT PrependCustomEntry ([in] HICON hIcon, [in] BSTR bstrText,
      [in] const void * pUserData);
      HRESULT AppendCustomEntry  ([in] HICON hIcon, [in] BSTR bstrText,
      [in] const void * pUserData);
      HRESULT RemoveCustomEntry  ([in] const void * pUserData,
      [out, retval] BOOL * pbSuccess);

      [propput] HRESULT CustomEntry([in] const void * pUserData);
      [propget] HRESULT CustomEntry([out, retval] const void **
      pUserData);
}
```

Description	
bRescan	Force a refresh on the combobox.
PrependCustomEntry	Prepend a custom entry to the combo box.
AppendCustomEntry	Append a custom entry to the combo box.
RemoveCustomEntry	Remove a custom entry from the combo box.
propput CustomEntry	Set the current selection to the entry specified by pUserData.
propget CustomEntry	Get the current selection to the entry specified by pUserData.

Identifier	Introduced in *NeroCBUI* version
IRecorderCombobox2	1.2.1.2

5.2.32. IRecorderCombobox3 Interface

Set a drive by name or drive letter.

```
interface IRecorderCombobox3 : IRecorderCombobox2
{
	[propput] HRESULT DriveByName ([in] BSTR bstrName);
	[propput] HRESULT DriveByLetter ([in] BSTR bstrLetter);
}
```

Identifier	Introduced in *NeroCBUI* version
IRecorderCombobox3	1.2.1.35

5.2.33. IRecorderComboboxCallbacks Interface

This interface is normally only implemented by the client when the client wants to be notified of the combobox selection change.

```
interface IRecorderComboboxCallbacks : IUnknown
{
	HRESULT OnFilterRecorder ([in] LPCNERO_SCSI_DEVICE_INFO
	pDeviceInfo, [out, retval] BOOL * pbAccept);
	HRESULT OnFilterEnd ();

	HRESULT OnContentRefreshed ();
	HRESULT OnCurSelInUseStatusChange ([in] BOOL bInUse);
};
```

5.2.34. ISpeedCombobox Interface

```
interface ISpeedCombobox : IUnknown
{
	[propput] HRESULT Callbacks ([in] ISpeedComboboxCallbacks *
	pCallbacks);
	[propput] HRESULT MediaType ([in] NERO_MEDIA_TYPE mediaType);

	[propget] HRESULT Speed([out, retval] DWORD * pdwSpeed);
	[propget] HRESULT SpeedInKBs ([out, retval] DWORD * pdwSpeedInKBs);

	[propput] HRESULT HideWhenEmpty ([in] BOOL bHide);
}
```

Identifier	Introduced in *NeroCBUI* version
ISpeedCombobox2	1.0.0.15

5.2.35. ISpeedCombobox2 Interface

```
interface ISpeedCombobox2 : ISpeedCombobox
{
	[propput] HRESULT SpeedInKbps ([in] DWORD dwSpeedInKbps);
	[propput] HRESULT SpeedInX ([in] DWORD dwSpeedInX);
}
```

Identifier	Introduced in *NeroCBUI* version
ISpeedCombobox2	1.2.1.3

5.2.36. ISpeedComboboxCallbacks Interface

```
interface ISpeedComboboxCallbacks : IUnknown
{
      HRESULT OnContentRefreshed ();
}
```

Identifier	Introduced in *NeroCBUI* version
ISpeedComboboxCallbacks	1.2.0.1

5.2.37. IUserDlgHandler Interface

```
interface IUserDlgHandler : IUnknown
{
      [propput] HRESULT ParentHWND ([in] HWND hWndParent);
      HRESULT HandleUserDlg ([in] NeroUserDlgInOut type, [in] void *data,
      [out, retval] NeroUserDlgInOut * pRetVal);
      [propget] HRESULT Aborted ([out, retval] BOOL * pbAborted);
      [propput] HRESULT Aborted ([in] BOOL bAborted);
}
```

Identifier	Introduced in *NeroCBUI* version
IUserDlgHandler	1.2.0.1

6. Software Version History

Version	Date	Comments
1.0.0.7	May 28, 2003	Added IEraseDiscDlg2 interface to support specifying additional flags to NeroEraseDisc.
1.0.0.8	May 29, 2003	Using polling in ChooseRecorderDlg to periodicaly update the maximum supported speed.
1.0.0.8	June 2, 2003	Added inteface IBurnProgressDlgEx that allows setting a client implemented callbacks interface.
1.0.0.8	June 5, 2003	Added ChooseSessionDlg object and its related IChooseSessionDlg interface.
1.0.0.8	June 24, 2003	Implemented changing the language with IInitSettings. Added translations of several languages.
1.0.0.11	July 18, 2003	Added IEraseDiscCallbacks and IEraseDiscDlg3 interface allowing EraseDiscDlg to support modeless dialog together with a modal one. Changed code to allow coexistence of modal and modeless dialog interfaces. Introduced IDialogImpl for this purpose. EraseSettingsDlg now supporting new IEraseSettingsDlg interface that allows modeless erase. EraseDiscDlg now supporting INERO_SCSI_DEVICE_INFO interface. Added capability for attaching foreign device handles. Disallowing all window close attempts (including Alt+F4) for EraseDiscDlg. RecorderCombobox: Modifying attached NERO_SCSI_DEVICE_INFO to reflect in use status. IEraseDiscCallbacks interface was forwarded only for modeless mode. Now including modal too.
1.0.0.15	September 1, 2003	Overburn message dialog now loads the description string from the resources. This fixes a bug with to long translation strings. Added IInitSettings2 and IBurnProgressDlg2 interfaces. Added "wizard" mode support for burn progress dlg and introduced "smart" resizability through IPanel related classes. Added SpeedCombobox object. Changed BurnSettingsDlg to use SpeedCombobox instead of own implementation. Added WaitForMediaDlg object. Added handler for recorder combobox in order to update write speeds combobox. Change of media type in SpeedCombobox now triggers content update. Updating speeds only if the combobox is visible. Always selecting the highest speed in SpeedCombobox if device changes. Added BurnSettingsDlgParam and ImageInfoDlgParam object support. Added IEraseDiscDlg4 interface. Disallowing message box after disc erase during burning.
1.2.0.0	September 4, 2003	Added support for DLG_MESSAGEBOX user callback
1.2.0.1	September 8, 2003	Added UserDlgHandler object for external use of NeroCBUI's own user dialog handler.
1.2.0.2	September 16, 2003	Added IBurnProgressDlg3 and IExternalChildWindow interfaces.
1.2.1.2	October 8, 2003	Added IRecorderComboBox2 interface to support custom entries and refreshing the combobox on application's request
1.2.1.3	October 23, 2003	Added ISpeedCombobox2 interface.
1.2.1.3	October 27, 2003	Added IRecorderCombobox3 interface.
1.2.1.6	December 3, 2003	Some new language translations have been completed.
1.2.1.7	December 3, 2003	Translations for Greek, Norwegian, Swedish and Thai have been completed.
1.2.1.9	January 8, 2004	Recompiled to include new and now complete translations.
1.2.1.11	February 6, 2004	Added IEraseSettingsDlg2 interface to get the HWND handle of the modeless dialog.