NXPowerLite SDK

User Manual

Version 7.0.X, June 2016

neuxpower.com

Simple Storage Reduction Software

Table of Contents

Introduction & overview

Introduction

What is NXPowerLite technology?

NXPowerLite is a file optimization technology which is used for reducing the size of image-heavy files – by as much as 99% in some cases. It works by eliminating any excess baggage and converting graphics to the most appropriate file format and resolution. Optimized files remain in their original format and retain all their functionality, so they can be opened and edited by anybody. There's no need for special decompression or viewing software.

What is the NXPowerLite SDK?

NXPowerLite Software Development Kit is a collection of libraries, wrappers and interfaces that enables developers to integrate the NXPowerLite optimization engine into other applications. It is based around a C API but also includes interfaces in the following code languages and formats:

- .NET Wrapper
- C++ header files
- Java Wrapper (supporting Java on Windows only)

For examples of each of the above in code, we have included code samples with the SDK for each coding language, except the C API. For further information see the **Getting Started** guide.

Documentation for the C API is not available. If you need to use these interfaces please contact support . To use the C API please use the enclosed header file nxopt.h or see the C samples in the SDK. We recommend using the C++ wrapper detailed in this documentation.

What file formats can the NXPowerLite SDK process?

Microsoft® PowerPoint® (97-2016)
Microsoft® Word (97-2016)
Microsoft® Excel® (97-2016)
JPEG
Adobe® PDF (up to PDF 1.7 extension level 8)
TIFF (uncompressed)
Zip

Is a 64-bit version available?

The SDK ships including both 32- and 64-bit versions.

What does the SDK contain?

The SDK contains the following folders:

bin	Contains all of the runtime libraries and executables required to run the NXPowerLite optimizer. See 'How do I use the Optimizer SDK?' section below for more details of the contents of this folder.
doc	Contains this document. Additional information can be found in the header files in the include folder.
include	Contains the header files required by the C and C++ interfaces. See 'How do I use the Optimizer SDK?' section below for more details
lib	Contains nxcapi.lib, which is required to link the C and C++ interfaces.
samples	Contains example code in C, C++, C# and Java. All languages demonstrate two examples - an optimization of a single file in its simplest form with no options set, and a more advanced sample that recursively optimizes a folder structure of files, setting options that match the screen settings of the desktop optimizer. In addition, there is an example command line application in the C++ samples. This replaces the previously-supplied command line executable. All of the samples have pre-built binaries, and the C, C++ and C# examples have Visual Studio 2013 Solutions to help build the samples. Note that all the samples build binaries that require parameters to be passed in to run correctly, which are as follows:

- Simple samples - an input file to be optimized, and the filename of the output (post-optimization) file. Both must be absolute paths.
- Advanced samples - an absolute path to the top level of the directory tree to be processed.

How do I use the NXPowerLite SDK?

To install, unzip the SDK distribution and copy to a place on your local drive. For the samples to build and run the folder structure must be maintained. Please ensure you are using the correct architecture (32- or 64-bit), as the SDK will not run if the architecture of the DLLs does not match the architecture of the client application.

Using the files in the bin folder

The bin folder contains a large number of DLLs and resource files, most of which are required by all the wrappers for the SDK to run correctly. These must be locatable on the DLL search path, and the simplest way to achieve this is to add the bin folder to your system path. The structure of the folder must be maintained, otherwise the SDK may become unstable.

While most of the files are required by all the wrappers, Neuxpower.NXPowerLite.dll has a more specific purpose being only required by C#. Similary both nxwrapj.jar and nxwrapj.dll are only required by Java. They may be relocated to a more convenient location as long as the remaining DLLs can be found by the system.

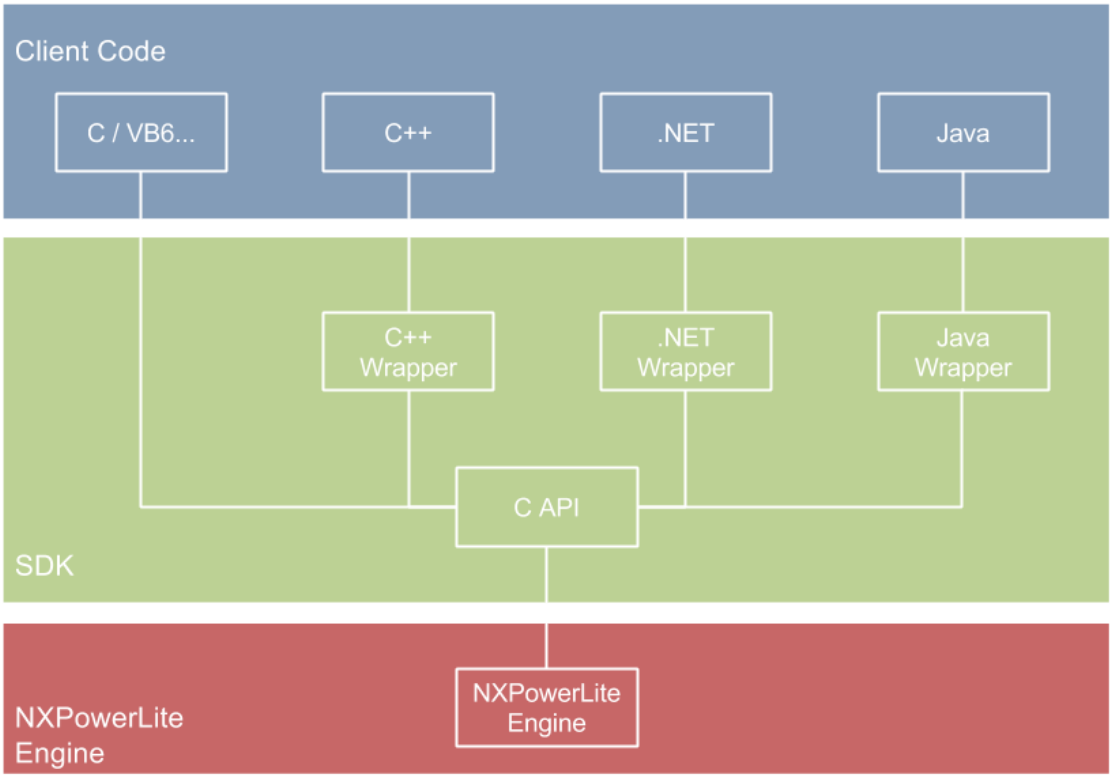
Using the files in the include folder

The include folder contains the header files required to build the C and C++ SDK interfaces. Please see the sample code for fuller details of usage. The files included are:

nxopt.h	Main entry point for the C SDK interface. Must also be present for the C++ API.
NXOptimizerLibrary.h NXOptimizer.h	Main entry points for the C++ SDK, these are not required for the C API.
NXException.h	Exception class for the C++ API, this is not required for the C API.
nxtypes.h	Basic types definition - required for both C and C++ but you should not need to directly reference it for the C++ wrapper.
nxerrors.h	Defines error types, required for both C and C++.
nxstatus.h	Defines status messages that may be returned by the call to Optimize; this is required for both C and C++.
nxerrorranges.h	Defines error and status message number ranges. Required for both C and C++.

Legal Notices

Architecture Overview



Supported File Extensions

File Types and File Extensions

NXPowerLite SDK can process the following types of files:

- PowerPoint
- Word
- Excel
- PDF (including PDF/A, PDF/E, PDF/VT, PDF/X)
- JPEG
- TIFF (uncompressed)
- Zip

However, there is not a one-to-one mapping of types of files to file extensions in every case. For example, PowerPoint files can have any one of the 9 file extensions listed below. Each file type and the supported file extensions within that file type are detailed below, with a description of what type of file each extension denotes.

By default NXPowerLite SDK will try to optimize any file in any of the file formats above, regardless of the file extension.

PowerPoint

File type	File Extension	Description
PowerPoint Presentation	.pptx	A PowerPoint 2007 or later presentation.
PowerPoint Macro-Enabled Presentation	.pptm	A presentation that contains Visual Basic for Applications (VBA) code.
PowerPoint 97-2003 Presentation	.ppt	A presentation that you can open in PowerPoint 97 to Office PowerPoint 2003.
PowerPoint Design Templates	.potx	A PowerPoint 2007 or later presentation template that you can use to format future presentations.
PowerPoint Macro-Enabled Design Template	.potm	A template that includes pre-approved macros that you can add to a template to be used in a presentation.
PowerPoint 97-2003 Design Template	.pot	A template that you can open in PowerPoint 97 to Office PowerPoint 2003.
PowerPoint Show	.ppsx	A presentation that always opens in Slide Show view rather than in Normal view.
PowerPoint Macro-Enabled Show	.ppsm	A slide show that includes pre-approved macros that you can run from within a slide show.
PowerPoint 97-2003 Show	.pps	A slide show that you can open in PowerPoint 97 to Office PowerPoint 2003.

Word

File type	File Extension	Description
Document	.docx	A Word 2007 or later document.
Macro-enabled document	.docm	A document that contains Visual Basic for Applications (VBA) code.
Word 97-2003 Document	.doc	A document that you can open in Word 97 to Office Word 2003.
Word Design Templates	.dotx	A Word 2007 or later document template that you can use to format future documents.
Word Macro-Enabled Design Template	.dotm	A template that includes pre-approved macros that you can add to a template to be used in a document.
Word 97-2003 Design Template	.dot	A template that you can open in Word 97 to Office Word 2003.

Excel

File type	File Extension	Description
Workbook	.xlsx	An Excel 2007 or later workbook.
Macro-enabled Workbook	.xlsm	A workbook that contains Visual Basic for Applications (VBA) code.
Excel 97-2003 Workbook	.xls	A workbook that you can open in Excel 97 to Office Excel 2003.
Word Design Templates	.xltx	An Excel 2007 or later workbook template that you can use to format future workbooks.
Excel Macro-Enabled Design Template	.xltm	A template that includes pre-approved macros that you can add to a template to be used in a workbooks.
Excel 97-2003 Design Template	.xlt	A template that you can open in Excel 97 to Office Excel 2003.

JPEG

JPEG files can have a number of possible extensions despite having the same internal structure. The following are expected JPEG file extensions:

- .jpeg

- .jpg
- .jpe
- .jif
- .jfi
- .jfif

TIFF

There are two file extensions associated with TIFF files:

- .tiff
- .tif

PDF & Zip

PDF and Zip files can only have one file extension ('.pdf' & '.zip' respectively). It is possible to have different types of PDF file, for example, a PDF/A file for archiving, however these files still have the extension '.pdf'.

Optimization Settings

Overview

Optimization settings allow you to adjust the balance between file reduction and file content quality. The default settings have been carefully chosen to deliver significant space savings without noticeably affecting quality, but for maximum safety we recommend that a backup or snapshot is taken before using the SDK in a production environment, particularly when experimenting with more aggressive optimization settings.

NXPowerLite Desktop (v7) profile mirroring map

The Desktop Edition of NXPowerLite has in-built profiles which are used to set the target screen resolution on which files will be displayed. The table below details the values that the SDK would need to be set at to mirror these profiles.

Property	Screen Profile	Print Profile	Mobile Profile
AllowCroppingOffice	true	true	true
AllowResizingOffice	true	false	true
DotsPerInchOffice* (See explanation below)	128	200	100
AllowRemoveHiddenContentOffice	true	true	true
AllowJPEGConversionOffice	true	true	true
JPEGQualityOffice	7	7	7
TargetScreenHeightPowerPoint	768	1200	600
TargetScreenWidthPowerPoint	1024	1600	800
AllowResizingJPEG	false	false	true
MinPixelSizeJPEG	0	0	0
JPEGQualityJPEG	7	8	7
AllowRemoveMetadataJPEG	false	false	false
LongestDimensionJPEG	1600	1600	800
MSOfficeOnly	false	false	false
AllowResizingPDF	true	false	true

Property	Screen Profile	Print Profile	Mobile Profile
AllowJPEGConversionPDF	true	true	true
AllowRemoveHiddenContentPDF	true	true	true
DotsPerInchPDF* (See explanation below)	128	200	100
JPEGQualityPDF	7	7	7
AllowASCIIToBinaryPDF	true	true	true
ArchiveEntryExtensionFilter	N/A	N/A	N/A
SupportedExtensions	N/A	N/A	N/A

DotsPerInch*

For simplicity in NXPowerLite Desktop Edition we have grouped Dots per Inch settings and target height and width into the same setting. In order to calculate the DPI from the given target height/width combo-box setting, we take the largest dimension and divide it by 8. For example, if the combo-box desktop setting is 1600 x 1200 then the DPI is 200.

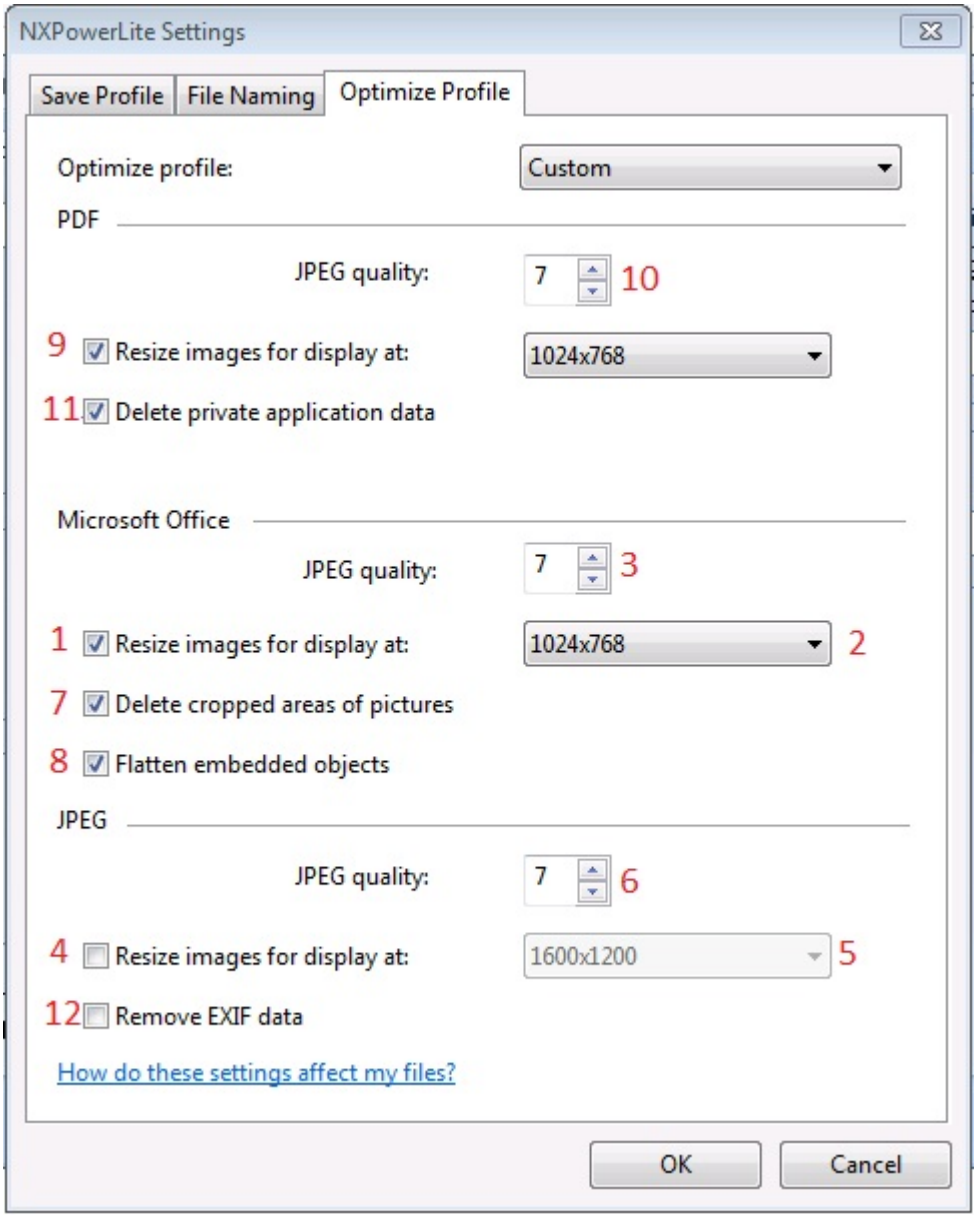
Of further note, certain file formats use either DPI or target height & width settings. For example, PowerPoint files use a screen target height & width not DPI value. For more information see the table at the bottom of the page.

NXPowerLite Desktop comparison

For users who have previously used NXPowerLite Desktop Edition the table and images below will help in configuring familiar settings.

Property	SDK Default values	NXPowerLite Desktop (See image below for reference)
AllowCroppingOffice	false	7
AllowResizingOffice	true	1
DotsPerInchOffice	200	N/A (See above for explanation)
AllowRemoveHiddenContentOffice	false	8
AllowJPEGConversionOffice	true	N/A (Always enabled)
JPEGQualityOffice	7	3
TargetScreenHeightPowerPoint	1200	2
TargetScreenWidthPowerPoint	1600	2
AllowResizingJPEG	false	4
MinPixelSizeJPEG	1470000 (e.g. 1400 x 1050)	N/A (Set to 0)
JPEGQualityJPEG	8	6
AllowRemoveMetadataJPEG	false	12
LongestDimensionJPEG	1600	5
MSOfficeOnly	false	N/A
AllowResizingPDF	true	9
AllowJPEGConversionPDF	true	N/A (Always enabled)
AllowRemoveHiddenContentPDF	false	11
DotsPerInchPDF	200	N/A (See above for explanation)
JPEGQualityPDF	7	10
AllowASCIIToBinaryPDF	false	N/A (Always enabled)
ArchiveEntryExtensionFilter	Any file type	N/A (All supported types optimized in archive)
SupportedExtensions	Full supported extensions	N/A

Property	SDK Default values	NXPowerLite Desktop (See image below for reference)



To which file formats does each setting apply?

Property	PowerPoint	Word	Excel	JPEG	PDF	Zip
AllowCroppingOffice	✔	✔	✔	✗	✗	✔*
AllowResizingOffice	✔	✔	✔	✗	✗	✔*
DotsPerInchOffice	✗	✔	✔	✗	✗	✔*
AllowRemoveHiddenContentOffice	✔	✔	✔	✗	✗	✔*
AllowJPEGConversionOffice	✔	✔	✔	✗	✗	✔*
JPEGQualityOffice	✔	✔	✔	✗	✗	✔*
TargetScreenHeightPowerPoint	✔	✗	✗	✗	✗	✔*
TargetScreenWidthPowerPoint	✔	✗	✗	✗	✗	✔*
AllowResizingJPEG	✗	✗	✗	✔	✗	✔*
MinPixelSizeJPEG	✗	✗	✗	✔	✗	✔*

Property	PowerPoint	Word	Excel	JPEG	PDF	Zip
JPEGQualityJPEG	✗	✗	✗	✔	✗	✔*
AllowRemoveMetadataJPEG	✗	✗	✗	✔	✗	✔*
LongestDimensionJPEG	✗	✗	✗	✔	✗	✔*
MSOfficeOnly	✔	✔	✔	✗	✗	✔*
AllowResizingPDF	✗	✗	✗	✗	✔	✔*
AllowJPEGConversionPDF	✗	✗	✗	✗	✔	✔*
AllowRemoveHiddenContentPDF	✗	✗	✗	✗	✔	✔*
DotsPerInchPDF	✗	✗	✗	✗	✔	✔*
JPEGQualityPDF	✗	✗	✗	✗	✔	✔*
AllowASCIIToBinaryPDF	✗	✗	✗	✗	✔	✔*
ArchiveEntryExtensionFilter	✗	✗	✗	✗	✗	✔

* These settings do not apply directly to the Zip file, but apply to files of the appropriate type that appear within a Zip file. For example, AllowCroppingOffice will apply to Office documents found within a Zip file.

Getting Started

The following page outlines how to compile and build an application that uses the NXPowerLite SDK in C, C++ or C#.

System Requirements

- Windows XP SP3 or above
- A suitable compiler, e.g. a recent version of Visual Studio
- C# developers will need .net 2.0 or later.
- Java developers will need a Java Development Kit (available from Oracle®) to undertake development using the SDK, version 1.5 or higher.

Installing the SDK

The SDK is packaged into a zip folder (nxsdk.zip for 32-bit version and nxsdk64.zip for the 64-bit version). Unzip the package(s) and copy the contents to a convenient place on your machine. The library is now ready for use.

The architecture of the SDK for Java needs to match the architecture of your Java Virtual Machine, so if you are running a 64-bit JVM then you need to use the 64-bit SDK, and vice versa.

Using the SDK

To get started, here is an example showing you how to use the objects of this simple SDK library. In the example we show you how to initialize the library using the **OptimizerLibrary** (C++, **C#**, **Java**), how to construct an **Optimizer** (C++, **C#**, **Java**) object to perform the optimization, how to catch any NXException (C++, **Java**) or **NXPowerLiteException** (C#) objects that may be thrown and how to optimize a file and how to terminate the library. The example is a console application where the input and output filenames are passed in as commandline arguments.

Note that you can also code the SDK using 'pure' C - the setup steps are similar to the C++ code; see the samples folder in the distribution for code usage.

CS Simple code example

```
using System;
using System.Collections.Generic;
using System.Text;
using Neuxpower.NXPowerLite;

namespace OptimizeFile
{
```

```
    class Program
    {
      static void Main(string[] args)
      {
        if (args.Length != 2)
        {
          // Description of utility
          Console.WriteLine("Optimize one file.");
          Console.WriteLine("Usage: ");
          Console.WriteLine("NXPowerLiteSDK.exe inputFile outputFile");
          Console.WriteLine("  inputFile - path of file to be
  optimized");
          Console.WriteLine("  outputFile - path to optimized file");
          return;
        }

        try
        {
          using (OptimizerLibrary lib = new
  OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
  OptimizerLibrary.NXSDK_VER_MINOR)) //Initialize the library and assign
  unmanaged resources

        using (Optimizer opt = new Optimizer()) //Assign unmanaged
resources
        {
         // With 'args[0]' as the input file and 'args[1]' as the output
file, do the optimization
          NXStatus nxStatus = opt.Optimize(args[0], args[1]);
          if (NXStatus.nxstatusOk == nxStatus)
          {
            Console.WriteLine("OK");
          }
          else
          {
            Console.WriteLine("Failed to optimize; status " +
nxStatus.ToString());
            // Although there was no error, the optimization did not take
place.
            // You can query 'nxStatus' to find out the reason why the
file could not be optimized.
          }

        }//Unmanaged resources are disposed here and the library is
terminated.
      }
      catch (NXPowerLiteException ex)
      {
        // Catch and display any NXPowerLite specific errors.

        Console.WriteLine("");
        Console.WriteLine("A NXPowerLiteException occurred.");
        Console.WriteLine("Error code: " + ex.NXError.ToString());
      }
    }
  }
}
```

C/C++ Simple code example

```
// Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>
#include <string>
#include "NXException.h"
#include "NXOptimizerLibrary.h"
#include "NXOptimizer.h"

int wmain(int argc, wchar_t* argv[])
{
  if (argc != 3)
  {
    std::wcerr << L"Optimize a file using C++ wrapper." << std::endl;
    std::wcerr << L"Usage:  OptimizeFile.exe input_filename
output_filename" << std::endl;
    std::wcerr << L"  input_filename   Full path to file to be
optimized." << std::endl;
    std::wcerr << L"  output_filename  Full path to optimized file." <<
std::endl;
    return 1;
  }

  try
  {
    nxpowerlite::OptimizerLibrary lib(NXSDK_VER_MAJOR,
NXSDK_VER_MINOR);
    nxpowerlite::Optimizer opt;

    std::wstring wstrInFilename(argv[1]);
    std::wstring wstrOutFilename(argv[2]);

    std::wcout << L"Optimizing." << std::endl;
    std::wcout << L"input: " << wstrInFilename << std::endl;
    std::wcout << L"To output: " << wstrOutFilename << std::endl;

    NXStatus nxStatus = opt.optimize(wstrInFilename.c_str(),
wstrOutFilename.c_str());

    if (nxstatusOk == nxStatus)
    {
      std::wcout << L"OK" << std::endl;
    }
    else
    {
      std::wcerr << L"Failed to optimize; status code: " <<
static_cast<int>(nxStatus) << std::endl;
    }

    return 0;
  }
  catch (nxpowerlite::NXException& ex)
  {
    std::wcerr << std::endl;
    std::wcerr << L"NXPowerLite exception occurred!" << std::endl;
    std::wcerr << L"Error code: " << static_cast<int>(ex.errorCode())
<< std::endl;
```

```
    return 2;
  }
  catch (...)
  {
    std::wcerr << std::endl;
    std::wcerr << L"A general exception occurred!" << std::endl;

    return 3;
  }
}
```

Java Simple Code Sample

```
import com.neuxpower.nxpowerlite.sdk.NXException;
import com.neuxpower.nxpowerlite.sdk.Optimizer;
import com.neuxpower.nxpowerlite.sdk.OptimizerLibrary;

public class SampleJavaSimple
{

  public static void main(String[] args)
  {

    if (args.length != 2)
    {
      System.err.println("Optimize a file using the Java wrapper.");
      System.err.println("Usage:  java -cp .:nxwrapj.jar
SampleJavaSimple input_filename output_filename");
      System.err.println("input_filename   Full path to file to be
optimized.");
      System.err.println("output_filename  Full path to optimized
file.");
      System.exit(1);
    }

    Optimizer opt = null;
    OptimizerLibrary library = null;
    int exitValue = 0;
    try
    {
      library = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR);
      opt = new Optimizer();
      if (opt != null)
      {
        System.out.println("Optimizing " + args[0]);
        System.out.println("Output file is " + args[1]);
        Optimizer.NXStatus status = opt.optimize(args[0], args[1]);
        if (Optimizer.NXStatus.nxstatusOk == status)
        {
          System.out.println("OK");
        }
        else
        {
          System.out.println("Failed to optimize file: status " +
status);
        }
```

```
      }
    }
    catch (NXException nxe)
    {
      // NXPowerLite exception - this will contain a NXErr value to
indicate what has gone wrong
      exitValue = 2;
    }
    catch (UnsatisfiedLinkError ule)
    {
       //  This exception gets thrown if nxwrapj.dll is in an incorrect
location or has the wrong architecture.
       exitValue = 3;
    }
    catch (Exception e)
    {
      // General exception
      exitValue = 3;
    }
    finally
    {
      try
      {
      if (opt != null)
      {
        opt.dispose();
      }
      if (library != null)
      {
        library.dispose();
      }
    }
    catch (NXException e)
    {
      System.err.println("");
      System.err.println("NXPowerLiteException occurred: " +
e.errorCode());
      exitValue = 2;
    }
  }
    System.exit(exitValue);
  }
}
```

Now it is time to **compile** and **link** the project files, then finally run the executable.

Compiling, linking and running the project

In order to build and run a project using the NXPowerLite SDK, you need to reference the following:

- If you are using C or C++:
  - The **include** folder in the NXPowerLite SDK needs to be added to your compiler include paths
  - The **lib** folder in the NXPowerLite SDK needs to be added to your linker additional library directories, and **nxcapi.lib** needs to be added to your linker inputs.
- If you are using C#:
  - You need to add **Neuxpower.NXPowerLite.dll** which can be found in the bin folder as a reference.

- If you are using Java:
  - nxwrapj.jar needs to be on your classpath, and nxwrapj.dll and nxcapi.dll need to be on the dll loading path. We show how this can be configured from command line .bat fines in the SDK Java samples.

- To run in any language, the **bin** folder in the NXPowerLite SDK needs to be added to your system path, or the contents of this folder need to be copied either to a location on the system path, or to the same folder as your built application.

> Further code examples can be found in the Samples folder of the sdk distribution.

Logging

The SDK and the Optimizer.exe process that it launches both provide logging functionality. This can help us to debug problems that may be occurring. By default logging is turned off but can be turned on or off for each component independently. For each component the log output is written to a file saved to a location specified by the user.

How to enable logging for the SDK

To enable logging the registry needs to be edited in the following way:

1. Navigate to the following key: [HKEY_LOCAL_MACHINE\SOFTWARE\Neuxpower\NXPowerLite SDK]
2. Add 'logging' as a REG_BINARY type and set the value to 1 to enable logging.
3. Add 'log file path' as a REG_SZ type. The value should be the absolute path to where you want the log file to be saved (including the filename) e.g. "C:\programdata\neuxpower\sdk_logs\sdk.log"

> If you are using a 32-bit NXPowerLite SDK on a 64-bit system, the registry path will be:
>
> [HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Neuxpower\NXPowerLite SDK]
>
> If you are using a 64-bit NXPowerLite SDK on a 64-bit system, you should omit Wow6432Node from the registry path.

How to enable logging for the Optimizer process

To enable logging the registry needs to be edited in the following way:

1. Navigate to the following key: [HKEY_LOCAL_MACHINE\SOFTWARE\Neuxpower\NXPowerLite SDK]
2. Add 'Optimizer logging' as a REG_BINARY type and set the value to 1 to enable logging.
3. Add 'Optimizer log file path' as a REG_SZ type. The value should be the absolute path to where you want the log file to be saved (including the filename) e.g. "C:\programdata\neuxpower\sdk_logs\optimizer.log"

> The Optimizer process is always 32-bit even in a 64-bit NXPowerLite SDK, so on 64-bit systems the registry path will be:
>
> [HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Neuxpower\NXPowerLite SDK]

> We do not advise setting the SDK and Optimizer logging to output to the same file. Should you wish to see both outputs at the same time, you can omit the log file paths to output to OutputDebugString. This can then be viewed via DebugView, available from Microsoft.

Reference

C#

Classes

OptimizerLibrary

Description

The OptimizerLibrary is the class responsible for the intialization and termination of the SDK library and inherits from System.IDisposable interface. By constructing an OptimizerLibrary object, the library automatically becomes initialized. When the object is destroyed by the garbage collector, unmanaged resources are automatically cleaned up and the library is terminated. It is recommended, however, that you do not rely on the garbage collector to destroy the object but, rather, ensure that Dispose is called either explicitly, in your code, or implicitly by employing the 'using' statement (see example code below). In this way, you have control over when the library is terminated and unmanaged resources are cleaned up.

This class also specifies the maximum number of Optimizer processes that should be used at any one time in a multi-threaded environment where several requests to **Optimize** can be made simultaneously. The default number of Optimizers is 1. However, at construction, a maximum number can be specified (see Constructors below). Specifying more than one Optimizer process where only one thread is executing will behave as if only 1 Optimizer process was specified and there will be no improvement in performance. You can only take advantage of multiple Optimizer processes by using several threads. Ideally, you should use the same number of threads (calling **Optimize** on each thread) as Optimizer processes specified.

It should be noted that, currently, the Optimizer process generates heavy input/output disk traffic. Specifying multiple processes could dramatically slow performance. It is recommended, for now, that you use no more than 1 Optimizer process (e.g. use the two-argument constructor only). Anything more than this should be considered experimental.

The OptimizerLibrary object must be instantiated before any other library objects can be created.

Constructors

OptimizerLibrary(int iMajorVersion, int iMinorVersion)	The constructor initializes the library and sets the number of Optimizer processes to be used to 1. The iMajorVersion argument should be set using OptimizerLibrary.NXSDK_VER_MAJOR, and iMinorVersion should be set using

OptimizerLibrary(int iMajorVersion, int iMinorVersion, int iMaxOptimizers)	OptimizerLibrary.NXSDK_VER_MINOR. The constructor initializes the library and the maximum number of Optimizer processes to use is set to be equal to *iMaxOptimizers*. Currently we recommend that only 1 Optimizer process is used. Anything greater should be considered experimental (see yellow note box above). The iMajorVersion argument should be set using OptimizerLibrary.NXSDK_VER_MAJOR, and iMinorVersion should be set using OptimizerLibrary.NXSDK_VER_MINOR.

It is recommended that you initialize and terminate the library as illustrated in the following example:

```
try
{
  using (OptimizerLibrary lib = new
OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR, OptimizerLibrary.NXSDK_VER_MINOR))
// Automatically initializes the library.
  using (Optimizer opt = new Optimizer())
  {
    // Optionally change any properties and call optimize().
    // ...
    // ...
  }// Automatically terminates the library and cleans up unmanaged resources
  catch (NXPowerLiteException& ex)
  {
    // Deal with any errors.  Catch NXPowerLiteException for specific NXPowerLite
errors.
    // ...
    // ...
  }
  catch(Exception& ex)
  {

    // Catch any general exceptions

  }
}
```

Destructor

~OptimizerLibrary()	This destructor cannot be called directly. This is used by the garbage collector when it destroys the object. Automatically cleans up unmanaged resources and terminates the library.

Method

Dispose

Optimizer

Description

This is the main class through which all file optimization operations are made. The class has properties that determine how an optimization should be performed. These can be altered using the property setters or, alternatively, the default values can be used (see Properties below for more detail). If changes are to be made to the properties, these should be done before calling the **Optimize** method for them to have any effect.

This class inherits the System.IDisposable interface. When the object is destroyed by the garbage collector, unmanaged resources are automatically cleaned up. It is recommended, however, that you do not rely on the garbage collector to destroy the object but, rather, ensure that Dispose is called either explicitly in your code, or implicitly by employing the 'using' statement (see example code below). In this way, you have control over when the unmanaged resources are cleaned up.

Optimizer also has a **BlockOnBusy** property which affects the behaviour of the return from a call to **Optimize** when the Optimizer process is busy in a multi-threaded setup. If 'true' then the calling thread will wait until the Optimizer process is free. If 'false' then the thread will return immediately allowing the caller time to perform other actions before attempting optimization again.

The **OptimizerLibrary** must be instantiated before constructing an Optimizer object otherwise construction will fail.

Constructors

Optimizer()

The **OptimizerLibrary** must be instantiated before constructing an Optimizer object otherwise construction will fail.

Properties

Each call to **Optimize** performs an optimization operation on the input file passed to the method. The optimization uses settings that determine how the optimization should be performed. Most of the properties of this object reflect those settings. These values can be viewed and altered; in the latter case changes should be made before an optimization takes place.

It is possible to call **Optimize** without the need to alter any of the properties. In this case, the default property values will be applied during optimization.

Office file specific properties:

- **AllowCroppingOffice**
- **AllowJPEGConversionOffice**
- **AllowRemoveHiddenContentOffice**
- **AllowResizingOffice**
- **DotsPerInchOffice**
- **JPEGQualityOffice**

- **MSOfficeOnly**
- **TargetScreenHeightPowerPoint**
- **TargetScreenWidthPowerPoint**

JPEG file specific properties:

- **AllowRemoveMetadataJPEG**
- **AllowResizingJPEG**
- **JPEGQualityJPEG**
- **LongestDimensionJPEG**
- **MinPixelSizeJPEG**

PDF file specific properties:

- **AllowASCIIToBinaryPDF**
- **AllowJPEGConversionPDF**
- **AllowRemoveHiddenContentPDF**
- **AllowResizingPDF**
- **DotsPerInchPDF**
- **JPEGQualityPDF**

Zip file specific properties:

ArchiveEntryExtensionFilter

Miscellaneous properties:

- **BlockOnBusy**
- **SupportedExtensions**

As well as the properties that affect optimization settings, there are a few read-only properties that can be accessed only after a successful optimization.

Read-only properties accessible after successful optimization:

- **LastFileHadHiddenContent**
- **LastFileHiddenContentRemoved**
- **LastFileType**

Method

Optimize

OptimizeEx

Dispose

Methods

OptimizerLibrary

Dispose

Signature

void Dispose()

Description

This method is inherited from the IDisposable interface. You can use this method explicitly to clean up any unmanaged resources and to terminate the library. However, as this method is automatically called when a 'using' statement is employed, you may prefer to rely on this style of coding to execute this functionality (see example code below).

Applies to:

OptimizerLibrary

Parameters

None

Return value

void

Example code

```
// Preferred method for ensuring that Dispose is called behind the scenes
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    // Set properties and do optimization, catching exceptions
    // ...
    // ...
  }// At this point OptimizerLibrary is out of scope and its Dispose() method has been
called.
   // The library has automatically terminated and unmanaged resources have been cleaned
up.
}
```

Note that the 'using' statement for the OptimizerLibrary object used above is equivalent to the following:

```
{
    OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR);
    try
    {
        // Create Optimizer object, set properties and do optimization, catching
exceptions
        // ...
        // ...
    }
    finally
    {
        if(lib != null)
        {
            ((IDisposable)lib).Dispose();
        }
    }
 }
```

Optimizer

Dispose

Signature

void Dispose()

Description

This method is inherited from the IDisposable interface. You can use this method explicitly to clean up any unmanaged resources used by the object. However, as this method is automatically called when a 'using' statement is employed, you may prefer to rely on this style of coding to execute this functionality (see example code below).

Applies to:

Optimizer

Parameters

None

Return value

void

Remarks/Notes

When this method is called, a request is made to the garbage collector not to call the finalizer for the specified object.

Example code

```
// Preferred method for ensuring that Dispose is called behind the scenes
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    // Set properties and do optimization, catching exceptions
    // ...
    // ...
  }// At this point Optimizer is out of scope and its Dispose() method has been called.
   // The library has automatically terminated and unmanaged resources have been cleaned
up.
}
```

Note that the 'using' statement for the Optimizer object used above is equivalent to the following:

```
{
    Optimizer opt = new Optimizer();
    try
    {
         // Set properties and do optimization, catching exceptions
        // ...
        // ...
    }
    finally
```

```
    {
        if(opt != null)
        {
            ((IDisposable)opt).Dispose();
        }
    }
}
```

Optimize

Signature

NXStatus Optimize(string inputFilename, string outputFilename)

Description

Optimizes a single file according to the settings stored as properties of the Optimizer object. The file is read from the path given in inputFilename and an optimized version is written to the path given in outputFilename. The input file must exist and the output file must **not** already exist. If this is not the case then an exception will be thrown. For a list of possible error codes that can exist in raised exceptions, see **Error Code Descriptions**.

Applies to:

All file types

Optimizer

Parameters

inputFilename	The path to the file that is to be optimized.
outputFilename	The path of the resulting optmized file.

Return value

NXStatus

Remarks/Notes

This method can throw a **NXPowerLiteException**. For a list of possible error codes that can be associated with the raised exception, see **Error Code Descriptions**.

Example code

```
try
{
    using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
```

```
    using (Optimizer opt = new Optimizer())
    {
        // Alter any properties...
        opt.AllowCroppingOffice = true;
        // ...

        NXStatus nxStatus = opt.Optimize(inFileName, outFileName);
        if (NXStatus.nxstatusOk == nxStatus)
        {
            Console.WriteLine("SUCCESSFULLY OPTIMIZED");
        }
        else
        {
            Console.WriteLine("FAILED TO OPTIMIZE");
            // Although the SDK did not optimize the file, no error occurred. Query
nxStatus to determine why optimization did not take place.
        }
    }
}
catch (NXPowerLiteException ex)
{
    Console.WriteLine("");
    Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
    Console.WriteLine("Error code: " + ex.NXError.ToString());
    Console.WriteLine("Error date: " + ex.ErrorDate);
    Console.WriteLine("Error time: " + ex.ErrorTime);
    Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
    Console.WriteLine("");
    Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

OptimizeEx

Signature

```
NXStatus OptimizeEx(string inputfillename, string outputFilename,
Optimizer.ProgressCallback progressCallback,

                    Optimizer.CancelCallback cancelCallback)
```

Description

Optimize a single file according to the settings stored as properties on the Optimizer object. The file is read for the path given in inputFilename and an optimized version is written to the path given in outputFilename. The input file must exist and the output file must not already exist. If either of these is not the case then nxstatusFileNotFound or nxstatusOutputFileExists respectively will be returned . Progress notification from the optimizer can be used in the progressCallback delegate and the cancelCallback delegate can be used to cancel the optimization. If the user cancels successfully then the new NXStatus code nxstatusUserCancelled will be returned.

Applies to:

All file types

Optimizer

Parameters

inputFilename	The path to the file that is to be optimized.
outputFilename	The path to the resulting optimized file.
progressCallback	Delegate that provides progress notification.
cancelCallback	Delegate that can cancel the optimization.

Return value

NXStatus

Remarks/Notes

This method can throw a **NXPowerLiteException**. For a list of possible error codes that can be associated with the raised exception, see **Error Code Descriptions**.

Example code

```
using System;
using System.Collections.Generic;
using System.Text;
using Neuxpower.NXPowerLite;
using System.Threading;

class Program
{
    private const int MAX_OPT_TIME = 10000; // 10 seconds is maximum time we want to
allow to optimize a file

    // Class Callback encapsulates optimization progress notification and cancellation
    public class Callback
    {
        // Flag that when set will cancel optimization
        private bool _cancel = false;

        // A thread that determines that maximum time that the optimizer can run
        // before being cancelled.
        private Thread _stopWatchThread;

        // A stop watch for measuring time intervals
        private System.Diagnostics.Stopwatch _stopWatch = new
System.Diagnostics.Stopwatch();

        // c'tor
        public Callback()
        {
        }

        // The progress notification callback function
        // <param name="total">The total abstract effort involved in optimizing a
file</param
        // <param name="done">The current effort</param>
        public void progress(uint total, uint done)
        {
```

```
            double percentage = 0.0;
            if (total == 0 && done == 0)
            {
                percentage = 100.0;
            }
            else
            {
                percentage = ((double)done / (double)total) * 100.0;
            }
            double progress = percentage / 100.0;
            int length = 20; // modify this to change the length of the progress bar
            int block = (int)(length * progress + 0.5);
            StringBuilder sb = new StringBuilder(length);
            for (int i = 0; i < block; i++)
            {
                sb.Append("*");
            }
            for (int j = 0; j < (length - block); j++)
            {
                sb.Append("-");
            }
            string Empty = "";
            string Done = "Done.";
            Console.Write(string.Format("\rProgress: [{0}] {1:0.00}% {2}",
sb.ToString(), percentage, (progress >= 1.0) ? Done : Empty));
        }

        // The cancellation callback function
        // <returns>false to continue optimization, true to stop optimization</returns>
        public bool cancel()
       {
            return _cancel;
        }

        // Start the stopwatch and the background thread
        public void startStopWatch()
        {
            _stopWatchThread = new Thread(watchDogFunction);
            _stopWatchThread.Start();
            _stopWatch.Start();
        }

        // Stop the stopwatch, if the background thread is still
        // running it is interrupted
        public void stopStopWatch()
        {
            if (null != _stopWatchThread)
            {
                _stopWatchThread.Interrupt();
                _stopWatch.Stop();
            }
        }


        // The elapsed time of optimization in milliseconds
        public long elapseTime()
        {
            return _stopWatch.ElapsedMilliseconds;
        }

        // A function that runs on the background thread, sleeping for a
        // maximum of MAX_OPT_TIME milliseconds. This will force
        // cancellation after a set period of time.
        private void watchDogFunction()
```

```
        {
             try
            {
                Thread.Sleep(MAX_OPT_TIME);
                _cancel = true;
            }
        }
        catch (ThreadInterruptedException)
        {
            return;
        }
        catch (Exception)
        {
            return;
        }
    }

    static void Main(string[] args)
    {
        try
        {
            using (OptimizerLibrary lib = new
OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR, OptimizerLibrary.NXSDK_VER_MINOR))
            using (Optimizer opt = new Optimizer())
            {
                // Alter any properties...
                opt.AllowCroppingOffice = true;
                // ...

                Callback cb = new Callback();
                cb.startStopWatch(); // Start thread to cancel after 10 seconds
                NXStatus nxStatus = opt.OptimizeEx(inFileName, outFileName, cb.progress,
cb.cancel);
                cb.stopStopWatch(); // Stop thread as no longer needed

                if (NXStatus.nxstatusOk == nxStatus)
                {
                    Console.WriteLine("SUCCESSFULLY OPTIMIZED");
                }
                else
                {
                    Console.WriteLine("FAILED TO OPTIMIZE");
                    // Although the SDK did not optimize the file, no error occurred.
Query nxStatus to determine why optimization did not take place.
                }
           }
        }
        catch (NXPowerLiteException ex)
        {
            Console.WriteLine("");
            Console.WriteLine("A NXPowerLiteException occurred with message " +
ex.Message);
            Console.WriteLine("Error code: " + ex.NXError.ToString());
            Console.WriteLine("Error date: " + ex.ErrorDate);
            Console.WriteLine("Error time: " + ex.ErrorTime);
            Console.WriteLine("The stack trace is " + ex.StackTrace);
        }
        catch (Exception ex)
        {
            Console.WriteLine("");
            Console.WriteLine("A General exception occurred with message " +
ex.Message);
        }
    }
```

```
}
```

Callback delegates

Cancel Callback

Signature

public delegate bool CancelCallback()

Description

Called periodically during optimization of a file, giving an opportunity to cancel optimization.

Return value

If the delegate returns true, optimization will stop at the next cancellation point and OptimizeEx will return **nxstatusUserCancelled**. If the delegate returns false then optimization will proceed.

ProgressCallback

Signature

public delegate void ProgressCallback(uint total, uint done)

Description

Delegate that specifies a method that can be used to obtain progress notification from the optimizer.

Parameters

total	The total abstract effort involved in optimizing a file.
done	The current effort.

Exceptions

NXPowerLiteException

Description

This class Inherits from the System.Exception class. It encapsulates any errors that may be raised

during execution. For a list of the NXPowerLite specific error codes that can be raised see **Error Code Descriptions**.

See example code for an example of how to catch NXPowerLiteException objects and how to display the error that occurred.

Constructors

NXPowerLiteException()

NXPowerLiteException(string message)

NXPowerLiteException(string message, SystemException innerException)

NXPowerLiteException(SerializationInfo info, StreamingContext context)

NXPowerLiteException(**NXErr** nxErr)

NXPowerLiteException(**NXErr** nxErr, SystemException innerException)

NXPowerLiteException(**NXErr** nxErr, string message)

NXPowerLiteException(**NXErr** nxErr, string message, SystemException innerException)

Properties

The following properties can be read only:

ApplicationCulture

ApplicationVersion

ErrorDate

ErrorTime

MachineName

Method

OSVersion

AppInnerException

NXError

NXPowerLiteException properties

ApplicationVersion

Description

ApplicationCulture

Description

The Culture (locale etc) of the Neuxpower.NXPowerLite.dll assembly.

Applies to:

NXPowerLiteException

Value type

string

Read/Write

Read only

ErrorDate

Description

The date the exception occurred in the format "yyyy-MM-dd".

Applies to:

NXPowerLiteException

Value type

string

Read/Write

Read only

ErrorTime

Description

The time the exception occurred in the long string format specified by the user's locale.

Applies to:

NXPowerLiteException

Value type

string

Read/Write

Read only

MachineName

Description

The name of the machine on which the error was raised.

Applies to:

NXPowerLiteException

Value type

string

Read/Write

Read only

Method

Description

The name of the method in which the exception was thrown.

Applies to:

NXPowerLiteException

Value type

string

Read/Write

Read only

OSVersion

Description

The version number of the operating system on which the exception was thrown.

Applies to:

NXPowerLiteException

Value type

string

Read/Write

Read only

AppInnerException

Description

This property is the description of the exception that was thrown that caused the current exception to be raised.

Applies to:

NXPowerLiteException

Value type

string

Read/Write

Read only

NXError

Description

The NXPowerLite specific error code returned by the Neuxpower.NXPowerLite.dll assembly. For a list of the error codes see **Error Code Descriptions**. This may be null.

Applies to:

NXPowerLiteException

Value type

NXErr?

Read/Write

Read only

C++

Classes

OptimizerLibrary

Description

The OptimizerLibrary is the class responsible for the intialization and termination of the SDK library. Instantiation of an OptimizerLibrary object will automatically becomes initialize the SDK library. Calling the destructor on the object will clean up resources and terminate the library.

This class also specifies the maximum number of Optimizer processes that should be used at any one time in a multi-threaded environment where several requests to optimize can be made simultaneously. The default number of Optimizers is 1. However, at construction, a maximum number can be specified (see Constructors below). Specifying more than one Optimizer process where only one thread is executing will behave as if only 1 Optimizer process was specified and there will be no improvement in performance. You can only take advantage of multiple Optimizer processes by using several threads. Ideally, you should use the same number of threads (calling optimize on each thread) as Optimizer processes specified.

It should be noted that, currently, the Optimizer process generates heavy input/output disk traffic. Specifying multiple processes could dramatically slow performance. It is recommended, for now, that you use no more than 1 Optimizer process (e.g. use the default constructor only). Anything more than this should be considered experimental.

OptimizerLibrary is in the nxpowerlite namespace.

Required header files are: NXOptimizerLibrary.h and NXException.h.

The OptimizerLibrary object must be instantiated before any other library objects can be created.

Constructors

OptimizerLibrary(int iMajorVersion, int iMinorVersion)	The constructor initializes the library and sets the number of Optimizer processes to be used to 1. The iMajorVersion argument should be set using NXSDK_VER_MAJOR, and iMinorVersion should be set using NXSDK_VER_MINOR.
OptimizerLibrary(int iMajorVersion, int iMinorVersion, int iMaxOptimizers)	The constructor initializes the library and the maximum number of Optimizer processes to use is set to be equal to *iMaxOptimizers*. Currently we recommend that only 1 Optimizer process is used. Anything greater should be considered experimental (see yellow note box above). The iMajorVersion argument should be set using NXSDK_VER_MAJOR, and iMinorVersion should be set using NXSDK_VER_MINOR.

```
#include "NXException.h"
#include "NXOptimizerLibrary.h"
  //...
  try
  {
    nxpowerlite::OptimizerLibrary lib(NXSDK_VER_MAJOR, NXSDK_VER_MINOR);

    // create Optimizer Object; Optimize files
    // ...
  }
  catch (nxpowerlite::NXException& ex)
  {
   // Handle exception
  }

  //...
```

Destructor

~OptimizerLibrary()	This destructor cleans up the SDK library.

Optimizer

Description

This is the main class through which all file optimization operations are made. The class has properties that determine how an optimization should be performed. These can be altered using the property setters or, alternatively, the default values can be used (see Properties below for more detail). If changes are to be made to the properties, these should be done before calling the optimize member function for them to have any effect.

Optimizer also has a **blockOnBusy** property which affects the behaviour of the return from a call to optimize when the Optimizer process is busy in a multi-threaded setup. If 'true' then the calling thread will wait until the Optimizer process is free. If 'false' then the thread will return immediately allowing the caller time to perform other actions before attempting optimization again.

The Optimizer is in the namespace nxpowerlite.

Required header files are: NXException.h and NXOptimizer.h

The OptimizerLibrary must be instantiated before constructing an Optimizer object otherwise construction will fail.

Constructors

Optimizer() Creates an Optimizer object.

```
#include "NXException.h"
#include "NXOptimizerLibrary.h"
#include "NXOptimizer.h"
  //...
  try
  {
    nxpowerlite::OptimizerLibrary lib(NXOPT_VER_MAJOR, NXOPT_VER_MINOR);
    nxpowerlite::Optimizer opt;
    // Optimize files;
    //...
  }
  catch (nxpowerlite::NXException& ex)
  {
    //Handle exception
  }

  //...
```

Destructor

~Optimizer() Destroys an Optimizer object.

Properties

Each call to optimize performs an optimization operation on the input file passed to the member function. The optimization uses settings that determine how the optimization should be performed. Most of the properties of this object reflect those settings. These values can be viewed and altered; in the latter case changes should be made before an optimization takes place.

It is possible to call optimize without the need to alter any of the properties. In this case, the default property values will be applied during optimization.

Office file specific properties:

- **allowCroppingOffice**
- **allowJPEGConversionOffice**
- **allowRemoveHiddenContentOffice**
- **allowResizingOffice**

- **dotsPerInchOffice**
- **JPEGQualityOffice**
- **msOfficeOnly**
- **targetScreenHeightPowerPoint**
- **targetScreenWidthPowerPoint**

JPEG file specific properties:

- **allowResizingJPEG**
- **allowRemoveMetadataJPEG**
- **JPEGQualityJPEG**
- **longestDimensionJPEG**
- **minPixelSizeJPEG**

PDF file specific properties:

- **allowASCIIToBinaryPDF**
- **allowJPEGConversionPDF**
- **allowRemoveHiddenContentPDF**
- **allowResizingPDF**
- **dotsPerInchPDF**
- **JPEGQualityPDF**

Zip file specific properties:

archiveEntryExtensionFilter

Miscellaneous properties:

- **blockOnBusy**
- **SupportedExtensions**

As well as the properties that affect optimization settings, there are a few read-only properties that can be accessed only after a successful optimization.

Read-only properties accessible after successful optimization:

- **lastFileHadHiddenContent**
- **lastFileHiddenContentRemoved**
- **lastFileType**

Method

optimize

optimizeEx

Member functions

Optimizer

optimize

Signature

NXStatus optimize(const wchar_t* pwszInputFile, const wchar_t* pwszOutputFile)

Description

Optimizes a single file according to the settings stored as properties of the Optimizer object. The file is read from the path given in pwszInputFile and an optimized version is written to the path given in pwszOutputFile. The input file must exist and the output file must **not** already exist. If this is not the case then an exception will be thrown. For a list of possible error codes that can exist in raised exceptions, see **Error Code Descriptions**.

Applies to:

All file types

Optimizer

Parameters

pwszInputFile	The path to the file that is to be optimized.
pwszOutputFile	The path of the optmized file.

Return value

NXStatus

Remarks/Notes

This method can throw a **NXException ('NXException Properties' in the on-line documentation)**. For a list of possible error codes that can be associated with the raised exception, see **Error Code Descriptions**.

Example code

```
int wmain(int argc, wchar_t* argv[])
{
    if (argc != 3)
    {
        return 1;
    }
    try
    {
        nxpowerlite::OptimizerLibrary lib(NXSDK_VER_MAJOR, NXSDK_VER_MINOR);
        nxpowerlite::Optimizer opt;
        std::wstring wstrInFilename(argv[1]);
        std::wstring wstrOutFilename(argv[2]);

        std::wcout << L"Optimizing." << std::endl;
        std::wcout << L"input: " << wstrInFilename << std::endl;
        std::wcout << L"To output: " << wstrOutFilename << std::endl;

        boost::posix_time::ptime startTime =
boost::posix_time::second_clock::local_time();

        NXStatus nxStatus = opt.optimize(wstrInFilename.c_str(), wstrOutFilename.c_str);

        boost::posix_time::ptime endTime =
boost::posix_time::second_clock::local_time();
        boost::posix_time::time_duration td = endTime - startTime;

        std::wcout << std::endl << L"Duration: " << td.total_seconds() << L" seconds."
<< std::endl;

        if (nxstatusOk == nxStatus)
        {
            std::wcout << L"SUCCESSFULLY OPTIMIZED." << std::endl;
            std::wcout << L"File has hidden content: " <<
bool_to_wstring(opt.lastFileHadHiddenContent()) << std::endl;
            std::wcout << L"File had hidden content removed: " <<
bool_to_wstring(opt.lastFileHiddenContentRemoved()) << std::endl;
            std::wcout << L"The file type is: " << NXFileTypeToString(opt.fileType()) <<
```

```
std::endl;
        }
        else
        {
            std::wcout << L"FAILED TO OPTIMIZE." << std::endl;
            std::wcout << L"Status code: " << NXStatusToString(nxStatus) << std::endl;
        }
    }
    catch (nxpowerlite::NXException& ex)
    {
        std::wcout << std::endl;
        std::wcout << L"NXPowerLite exception occurred!" << std::endl;
        std::wcout << L"Error code: " << NXErrToString(ex.errorCode()) << std::endl;
        return 2;
    }
    catch (...)
    {
        std::wcout << std::endl;
        std::wcout << L"A general exception occurred!" << std::endl;
        return 3;
    }
}
```

optimizeEx

Signature

NXStatus optimizeEx(const wchar_t* pwszInputFile, const wchar_t* pwszOutputFile, **nxpowerlite::IProgress** `progressCallback`,

`nxpowerlite::ICancel` `cancelCallback`)

Description

Optimize a single file according to the settings stored as properties on the Optimizer object. The file is read for the path given in inputFilename and an optimized version is written to the path given in outputFilename. The input file must exist and the output file must not already exist. If either of these is not the case then nxstatusFileNotFound or nxstatusOutputFileExists respectively will be returned . Progress notification from the optimizer can be accessed via the nxp::IProgress interface and, similarly, cancellation of the optimization process can be performed using the nxp::ICancel interface. If the user cancels successfully then the new NXStatus code nxstatusUserCancelled will be returned.

Applies to:

All file types

Optimizer

Parameters

pwszInputFile	The path to the file that is to be optimized.
pwszOutputFile	The path of the optmized file.
progressCallback	Interface that provides progress notification.
cancelCallback	Interface through which optimization can be cancelled.

Return value

NXStatus

Remarks/Notes

This method can throw a NXException. For a list of possible error codes that can be associated with the raised exception, see **Error Code Descriptions**.

Example code

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include <string>

#include <boost/thread.hpp>

#include "NXException.h"
#include "NXOptimizerLibrary.h"
#include "NXOptimizer.h"

const int MAX_OPT_TIME = 60000; // 60 seconds is maximum time allowed to optimize a
file
void updateInlineProgress(const std::wstring& wstrTitle, double progress);

class Callbacks : public nxpowerlite::IProgress, public nxpowerlite::ICancel
{
public:
    Callbacks() : nxpowerlite::IProgress(), nxpowerlite::ICancel(), m_fCancel(false)
    {
        startWatchDog();
    }

    virtual ~Callbacks(){}

    virtual void progress(unsigned long ulTotal, unsigned long ulDone)
    {
        double percentage = 0.0;
        if (ulTotal == 0 && ulDone ==0)
        {
            percentage = 100.0;
        }
        else
        {
            percentage = ((double)ulDone / (double)ulTotal) * 100.0;
        }

        updateInlineProgress(L"C++ Wrapper Progress Report", percentage / 100.0);
    }

    virtual bool cancel()
    {
        return m_fCancel;
```

```
    }

    void startWatchDog()
    {
        try
        {
            m_spWatchDogThread =
            boost::shared_ptr<boost::thread>(new
boost::thread(&Callbacks::watchDogFunc, this, boost::ref(m_fCancel)));
        }
        catch (...)
        {
            return;
        }
    }

    void stopWatchDog()
    {
        if (m_spWatchDogThread)
        {
            m_spWatchDogThread->interrupt();
        }
    }

private:
    void watchDogFunc(bool& fCancel)
    {
        try
        {
            boost::posix_time::millisec sleepTime(MAX_OPT_TIME);
            boost::this_thread::sleep(sleepTime);
            fCancel = true;
        }
        catch (boost::thread_interrupted)
        {
            return;
        }
        catch (...)
        {
            return;
       }
    }

private:
    bool m_fCancel;
    boost::shared_ptr<boost::thread> m_spWatchDogThread;
};

void printUsage()
{
    std::wcout << L"Optimize a file using C++ wrapper." << std::endl;
    std::wcout << L"Usage:  sampleCPP.exe input_filename output_filename" <<
std::endl;
    std::wcout << L"  input_filename   Full path to file to be optimized." <<
std::endl;
    std::wcout << L"  output_filename  Full path to optimized file." << std::endl;
}

const wchar_t* bool_to_wstring(bool fValue)
{
    switch (fValue)
    {
        case true: return L"True";
        case false: return L"False";
```

```
    }
    return L"";
}

/**
 * @brief Displays a progress bar
 *
 * This is an inline progress bar
 * Access level: public
 * Qualifier:
 *
 * @param[in] const std::wstring & wstrTitle - The title of the progress bar
 * @param[in] double progress - A measure of the progress has values in the
 * range [0.0,1.0].
 *
 * @return void/
 */
void updateInlineProgress(const std::wstring& wstrTitle, double progress)
{
    assert(progress >= 0);

    int length = 20; // modify this to change the length of progress bar
    int block = (int)(length * progress + 0.5);
    std::wstring wstrHash(block, L'*');
    std::wstring wstrDash((length - block), L'-');
    std::wstring wstrFill = wstrHash + wstrDash;
    double iComplete = progress * 100.0;
    std::wstring wstrEmpty;
    std::wstring wstrDone(L" Done.");
    fwprintf(stdout, L"\r%s: [%s] %.2f%% %s", wstrTitle.c_str(), wstrFill.c_str(),
iComplete, (progress >= 1.0) ? wstrDone.c_str() : wstrEmpty.c_str());
    fflush(stdout);
}

int wmain(int argc, wchar_t* argv[])
{
    if (argc != 3)
    {
        printUsage();
        return 1;
    }
    try
    {
        nxpowerlite::OptimizerLibrary lib(NXSDK_VER_MAJOR, NXSDK_VER_MINOR);
        nxpowerlite::Optimizer opt;
        std::wstring wstrInFilename(argv[1]);      //Comment[1]
        std::wstring wstrOutFilename(argv[2]);     //Comment[2]

        std::wcout << L"Optimizing." << std::endl;
        std::wcout << L"input: " << wstrInFilename << std::endl;
        std::wcout << L"To output: " << wstrOutFilename << std::endl;

        boost::posix_time::ptime startTime =
boost::posix_time::second_clock::local_time();

        Callbacks cb;
        NXStatus nxStatus = opt.optimizeEx(wstrInFilename.c_str(),
wstrOutFilename.c_str(), &cb, &cb);
        cb.stopWatchDog();

        boost::posix_time::ptime endTime =
boost::posix_time::second_clock::local_time();
        boost::posix_time::time_duration td = endTime - startTime;
```

```
        std::wcout << std::endl << L"Duration: " << td.total_seconds() << L"
seconds." << std::endl;

        if (nxstatusOk == nxStatus)
        {
            std::wcout << L"SUCCESSFULLY OPTIMIZED." << std::endl;
            std::wcout << L"File has hidden content: " <<
bool_to_wstring(opt.lastFileHadHiddenContent()) << std::endl;
            std::wcout << L"File had hidden content removed: " <<
bool_to_wstring(opt.lastFileHiddenContentRemoved()) << std::endl;
            std::wcout << L"The file type is: " << NXFileTypeToString(opt.fileType())
<< std::endl;
        }
        else
        {
            std::wcout << L"FAILED TO OPTIMIZE." << std::endl;
            std::wcout << L"Status code: " << NXStatusToString(nxStatus) <<
std::endl;
        }
    }
    catch (nxp::NXException& ex)
    {
        std::wcout << std::endl;
        std::wcout << L"NXPowerLite exception occurred!" << std::endl;
        std::wcout << L"Error code: " << NXErrToString(ex.errorCode()) << std::endl;
        return 2;
    }
    catch (...)
    {
        std::wcout << std::endl;
        std::wcout << L"A general exception occurred!" << std::endl;
        return 3;
    }
}
```

Callback interfaces

ICancel

Description

This interface provides the mechanism to cancel the optimization of a file in mid flow. Clients can provide a concrete implementation of ICancel which can then be passed as a parameter to Optimizer::optimizeEx. The concrete object will then be called at various points during optimization to provide the client with an opportunity to cancel the process.

If the concrete implementation of the method 'cancel()' returns 'true' then optimization will be stopped and cancelled; if it returns 'false' then optimization will continue.

Constructors

ICancel()

Destructor

virtual ~ICancel()

Method

virtual bool cancel() = 0;

IProgress

Description

This interface provides progress notification functionality. Clients can provide a concrete implementation of IProgress which can then be passed as a parameter to Optimizer::optimizeEx. The concrete object will then be notified of progress while optimization takes place.

Constructors

IProgress()

Destructor

virtual ~IProgress()

Method

virtual void progress(unsigned long ulTotal, unsigned long ulDone)=0;

Exceptions

NXException

Description

This class encapsulates any errors that may be raised by the SDK. For a list of the NXPowerLite specific error codes that can be raised see **Error Code Descriptions**.

The NXException is in the namespace nxpowerlite.

Required header file is: NXException.h.

See example code for an example of how to catch NXException objects and how to display the error that

occurred.

Constructors

NXException(**NXErr** nxErr) Constructs an NXException object corresponding to the Error Code nxErr.

Properties

The following property is readonly

errorCode

NXException properties

---

errorCode

Description

The NXPowerLite specific error code returned by the C++ wrapper for the SDK. For a list of the error codes see **Error Code Descriptions**.

Applies to:

NXException

Value type

NXErr

Read/Write

Read only

Example code

```
#include "NXException.h"

  //...

  try
  {
    //...
  }
  catch (nxpowerlite::NXException& ex)
  {
    std::wcerr << L"NXPowerLite exception occurred!" << std::endl;
    std::wcerr << L"Error code: " << NXErrToString(ex.errorCode()) << std::endl;
```

```
}
//...
```

Java

Classes

OptimizerLibrary

Description

The OptimizerLibrary is the class responsible for the intialization and termination of the SDK library. By constructing an OptimizerLibrary object, the library automatically becomes initialized. When the object is destroyed by the Java garbage collector, unmanaged resources are automatically cleaned up and the library is terminated. It is recommended, however, that you do not rely on the garbage collector to destroy the object but, rather, ensure that dispose is called explicitly in your code. In this way, you have control over when the library is terminated and unmanaged resources are cleaned up.

This class specifies the maximum number of Optimizer processes that should be used at any one time in a multi-threaded environment where several requests to **optimize** can be made simultaneously. The default number of Optimizers is 1. However, at construction, a maximum number can be specified (see Constructors below). Specifying more than one Optimizer process where only one thread is executing will behave as if only 1 Optimizer process was specified and there will be no improvement in performance. You can only take advantage of multiple Optimizer processes by using several threads. Ideally, you should use the same number of threads (calling **optimize** on each thread) as Optimizer processes specified.

It should be noted that, currently, the Optimizer process generates heavy input/output disk traffic. Specifying multiple processes could dramatically slow performance. It is recommended, for now, that you use no more than 1 Optimizer process (e.g. use the default constructor only). Anything more than this should be considered experimental.

The OptimizerLibrary object must be instantiated before any other library objects can be created.

Internally, the NXPowerLite Java SDK uses Java Native Interface, and has to load nxwrapj.dll to access the optimizer functionality. The first time this happens will probably be when instantiating the instance of OptimizerLibrary. If it fails to find this, the exception java.lang.UnsatisfiedLinkError will be thrown. Note that the dll needs to be on the path specified by the Java property java.library.path.

If an error within the optimizer code then **NXException** is thrown. Errors may include nxerrAlreadyInitialized.

This class is in the package com.neuxpower.nxpowerlite.sdk.

Constructors

OptimizerLibrary Constructors **COPYCODE**

```
OptimizerLibrary(int sdkMajorVersion, int sdkMinorVersion) throws NXException;
OptimizerLibrary(int sdkMajorVersion, int sdkMinorVersion, int numOptimizers) throws
NXException;

try
{
  OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR);
  // Do stuff
  lib.dispose(); // Shown in simplified try-catch structure.
}
catch (NXException nxe)
{
  System.err.println("NXException occurred: " + nxe.getNXErr());
}
catch (UnsatisfiedLinkError ule)
{
  // Watch out for this one, it means that nxwrapj.dll cannot be located
}
```

Method

dispose

Optimizer

Description

This is the main class through which all file optimization operations are made, with methods setting optimizer properties that determine how an optimization should be performed. The properties can be altered using the methods detailed in the properties reference, alternatively, the default values can be used. If changes are to be made to the properties, these should be done before calling the **optimize** or **optimizeEx** methods for them to have any effect.

When the object is destroyed by the Java garbage collector, the Optimizer is automatically cleaned up. It is recommended, however, that you do not rely on the garbage collector to destroy the object but, rather, ensure that dispose is called explicitly in your code. In this way, you have control over when the unmanaged resources are cleaned up.

Optimizer also has a **BlockOnBusy** property which affects the behaviour of the return from a call to **optimize** or **optimizeEx** when the Optimizer process is busy in a multi-threaded setup. If 'true' then the calling thread will wait until the Optimizer process is free. If 'false' then the thread will return

immediately allowing the caller time to perform other actions before attempting optimization again.

The **OptimizerLibrary** must be instantiated before constructing a Optimizer instance otherwise construction will fail.

Calls to optimize return a Optimizer.NXStatus enum value, mapping to the **status codes** as follows:

```
public enum NXStatus {
nxstatusOk,
nxstatusErrorOccurred,
nxstatusUnsupportedFileFormat,
nxstatusOptBusy,
nxstatusFileInUse,
nxstatusAlreadyOptimized,
nxstatusCannotReduce,
nxstatusOfficeFastSavedDoc,
nxstatusInvalidFormat,
nxstatusJPEGTooSmall,
nxstatusFileVersionTooOld,
nxstatusUnsupportedPDFSubset,
nxstatusNotSavedByMSOffice,
nxstatusFileVersionTooNew,
nxstatusDigitallySigned,
nxstatusEncryptedOrPassword,
nxstatusFileNotFound,
nxstatusOptCrashed,
nxstatusTaggingFailed,
nxstatusPDFFailure,
nxstatusFailedToCopyInputFile,
nxstatusCannotCreateOutputFile,
nxstatusOutputFileExists,
nxstatusOptimizerFailure,
nxstatusUserCancelled,
nxstatusUnsupportedTIFFFile,
nxstatusUnsupportedTIFFTags
}
```

This class is in the package com.neuxpower.nxpowerlite.sdk.

Constructors

Optimizer Constructor **COPYCODE**

```
public Optimizer() throws NXException;

// The following code example should be wrapped in a try/catch clause:
Optimizer opt = new Optimizer();
```

Properties

Each call to **optimize** or **optimizeEx** performs an optimization operation on the input file passed to the method. The optimization uses settings that determine how the optimization should be performed. The properties of this object reflect those settings. These values can be viewed and altered; in the latter case changes should be made before an optimization takes place.

It is possible to call **optimize** or **optimizeEx** without the need to alter any of the properties. In this case, the default property values will be applied during optimization.

Office file specific properties:

- **AllowCroppingOffice**
- **AllowResizingOffice**
- **DotsPerInchOffice**
- **AllowRemoveHiddenContentOffice**
- **AllowJPEGConversionOffice**
- **JPEGQualityOffice**
- **TargetScreenHeightPowerPoint**
- **TargetScreenWidthPowerPoint**
- **MSOfficeOnly**

JPEG file specific properties:

- **AllowResizingJPEG**
- **MinPixelSizeJPEG**
- **JPEGQualityJPEG**
- **AllowRemoveMetadataJPEG**
- **LongestDimensionJPEG**

PDF file specific properties:

- **AllowASCIIToBinaryPDF**
- **AllowResizingPDF**
- **AllowJPEGOptPDF**
- **AllowRemoveHiddenContentPDF**
- **DotsPerInchPDF**
- **JPEGQualityPDF**

Zip file specific properties:

ArchiveEntryExtensionFilter

Miscellaneous properties:

- **BlockOnBusy**
- **SupportedExtensions**

Method

optimize

optimizeEx

dispose

Member functions

OptimizerLibrary

dispose

Signature

```
public void dispose() throws NXException;
```

Description

You can call this function to end use of the optimizer library

Applies to:

OptimizerLibrary

Return value

There is no return value, however if an error occurs **NXException** will be thrown. NXErr values may include nxerrNotInitialized.

Example code

See **OptimizerLibrary**

Optimizer

optimize

Signature

NXStatus optimize(java.lang.String inputFileName, java.lang.String outputFileName) throws NXException;

Description

Optimizes a single file according to the settings stored as properties of the Optimizer object. The file is read from the path given in inputFilename and an optimized version is written to the path given in outputFilename. The input file must exist and the output file must **not** already exist. If this is not the case then an exception will be thrown. For a list of possible error codes that can exist in raised exceptions, see **Error Code Descriptions**.

Applies to:

All file types

Optimizer

Parameters

inputFilename	The path to the file that is to be optimized.
outputFilename	The path of the resulting optmized file.

Return value

Optimizer.NXStatus

Remarks/Notes

This method can throw a **NXException**. For a list of possible error codes that can be associated with the raised exception, see **Error Code Descriptions**.

Example code

```
try // Simplified try-catch setup
{
  OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR);
  Optimizer opt = new Optimizer();
  opt.allowCropping(true);
  // Set other properties as required
  Optimizer.NXStatus status = opt.optimize(inputFilename, outputFilename);
  if (Optimizer.NXStatus.nxstatusOk == status )
  {
    System.out.println("SUCCESSFULLY OPTIMIZED");
  }
  else
  {
    System.out.println("FAILED TO OPTIMIZE");
    // Although the SDK did not optimize the file, no error occurred.
```

```
    // Query nxStatus to determine why optimization did not take place.
    }
  }
  // The try-catch statement is simplified here, but if any of the property setting
methods or optimize itself
  // throw an exception, this clean up will be missed.
  opt.dispose();
  lib.dispose();
}
catch (NXException nxe)
{
    System.out.println("A NXException occurred with NXErr" + nxe.getNXErr());
}
catch (Exception e)
{
    System.out.println("A General exception occurred with message " + ex.Message);
}
```

dispose

Signature

```
public void dispose() throws NXException;
```

Description

You can call this function to end use of the optimizer

Applies to:

Optimizer

Return value

There is no return value, however if an error occurs **NXException** will be thrown. NXErr values may include nxerrNotInitialized.

Example code

See **optimize**

optimizeEx

Signature

Optimizer.NXStatus optimizeEx(java.lang.String inputFileName, java.lang.String outputFileName, **IProgressCallback** progressCallback, **ICancelCallback** cancelCallback) throws NXException;

Description

Optimize a single file according to the settings stored as properties on the Optimizer object. The file is read for the path given in inputFilename and an optimized version is written to the path given in outputFilename. The input file must exist and the output file must not already exist. If either of these is not the case then nxstatusFileNotFound or nxstatusOutputFileExists respectively will be returned . Progress notification from the optimizer can be accessed via the **IProgressCallback** interface and, similarly, cancellation of the optimization process can be performed using the **ICancelCallback** interface. If the user cancels successfully then the new NXStatus code nxstatusUserCancelled will be returned.

Applies to:

All file types

Optimizer

Parameters

inputFileName	The path to the file that is to be optimized.
outputFileName	The path of the optmized file.
progressCallback	Interface that provides progress notification.
cancelCallback	Interface through which optimization can be cancelled.

Return value

NXStatus

Remarks/Notes

This method can throw a **NXException**. For a list of possible error codes that can be associated with the raised exception, see **Error Code Descriptions**.

Example code

```
import com.neuxpower.nxpowerlite.sdk.ICancelCallback;
import com.neuxpower.nxpowerlite.sdk.IProgressCallback;
import com.neuxpower.nxpowerlite.sdk.NXException;
import com.neuxpower.nxpowerlite.sdk.Optimizer;
import com.neuxpower.nxpowerlite.sdk.OptimizerLibrary;
public class SampleJavaAdvanced
{

  class CancelHandler implements ICancelCallback
  {
     public boolean cancel() {
     return _cancelled;
  }

    public boolean _cancelled = false; // Set to true if the user indicates that they
want to cancel
  }

  CancelHandler _cancelHandler = new CancelHandler();

  class ProgressHandler implements IProgressCallback
  {
    public void progress(int total, int done)
    {
```

```
    System.out.println("Completed " + done + " of " + total);
  }
}

ProgressHandler _progressHandler = new ProgressHandler();

// Thrown if the user cancels
class UserCancelledException extends Exception
{

}

private Optimizer _optimizer = null;

/** Basic constructor - creates the Optimizer object and sets it up to reflect the
 * NXPowerLite desktop product screen settings.
 * @throws NXException
 */
public SampleJavaAdvanced() throws NXException
{
  _optimizer = new Optimizer();
  // Different to the desktop default settings - this demonstrates only optimizing
  // *.doc, *.xls and *.ppt files within a zip file
  _optimizer.archiveEntryExtensionFilter("doc:xls:ppt");
  // Set properties for standalone JPEG files
  _optimizer.allowResizingJPEG(false);
  _optimizer.longestDimensionJPEG(1600);
  _optimizer.minPixelSizeJPEG(0);
  _optimizer.allowRemoveMetadataJPEG(false);
  _optimizer.JPEGQualityJPEG(7);
  // Set properties for Office files
  _optimizer.allowCroppingOffice(true);
  _optimizer.allowJPEGConversionOffice(true);
  _optimizer.allowResizingOffice(true);
  //_optimizer.allowRemoveHiddenContentOffice(true); - handled below
  _optimizer.targetScreenHeightPowerPoint(768);
  _optimizer.targetScreenWidthPowerPoint(1024);
  _optimizer.msOfficeOnly(false);
  _optimizer.JPEGQualityOffice(7);
  _optimizer.dotsPerInchOffice(128);
  // Set properties for PDF files
  _optimizer.dotsPerInchPDF(128);
  _optimizer.allowResizingPDF(true);
  _optimizer.allowJPEGConversionPDF(true);
  _optimizer.allowRemoveHiddenContentPDF(true);
  _optimizer.JPEGQualityPDF(7);
}

/**
 * Function to process folder structure, recurses on each folder found.
 * @param folder - folder to process
 * @throws NXException
 * @throws UserCancelledException
 */
public void processFolder(File folder) throws NXException, UserCancelledException
{
  for (String childName : folder.list())
  {
    File fileOrFolder = new File(folder, childName);
    if (fileOrFolder.isDirectory())
    {
      processFolder(fileOrFolder);
    }
    else
```

```
      {
        processFile(fileOrFolder);
      }
    }
  }

  /**
   * Tidy function that calls the code to terminate the optimizer handle
   * @throws NXException
   */
  public void finished() throws NXException
  {
    _optimizer.dispose();
  }

  /**
   * Calls the optimizer with the given file name
   * @param file - the file to be optimized
   * @throws NXException - if the optimization has caused an error. This should cause
   * sthe sample to stop running.
   * Messages reflected by the status do not stop the process, but are treated as
informational.
   * @throws UserCancelledException
   *
   */
  private void processFile(File file) throws NXException, UserCancelledException
  {
    String optimizedSuffix = "(Optimized)";
    String name = file.getName(); // Includes extension
    int extPos = name.lastIndexOf(".");
    String newName = "";
    String ext = "";
    if (-1 != extPos)
    {
      ext = name.substring(extPos, name.length());
      newName = name.substring(0, extPos) + optimizedSuffix + ext;
    }
    else
    {
      // No extension
      newName = name + optimizedSuffix;
    }
    // Flatten embedded objects for PowerPoint files only.
    _optimizer.allowRemoveHiddenContentOffice(true);
    _optimizer.allowRemoveHiddenContentOffice(false);

    File newFile = new File(file.getParentFile(), newName);
    String inputFile = file.getAbsolutePath();
    String outputFile = newFile.getAbsolutePath();
    System.out.println("Optimizing " + inputFile);
    System.out.println("Output file is " + outputFile);
    Optimizer.NXStatus status = _optimizer.optimizeEx(inputFile, outputFile,
_progressHandler, _cancelHandler);
    if (Optimizer.NXStatus.nxstatusOk == status)
    {
      System.out.println("OK - file type is " + _optimizer.lastFileType());
      if (_optimizer.lastFileHadHiddenContent())
      {
        if (_optimizer.lastFileHiddenContentRemoved())
        {
          System.out.println("Last file had hidden content removed");
        }
        else
        {
```

```
          System.out.println("Last file had hidden content which was not removed");
        }
      }
      else
      {
        System.out.println("Last file had no hidden content");
      }
    }
    else
    {
      System.out.println("Failed to optimize file: status " + status);
      if (status == Optimizer.NXStatus.nxstatusUserCancelled)
      {
        throw new UserCancelledException();
      }
    }
    // Not all files will check the cancelled flag during optimisation, so this catches
if the cancel flag has been set.
    if (_cancelHandler._cancelled)
    {
      throw new UserCancelledException();
    }
  }

  /**
   * Entry point for the sample
   *
   * To build, the classpath must include nxwrapj.jar which is in the bin folder of the
sdk.
   * To run, it must be able to access the DLLs and executables in the bin folder of the
SDK.
   * The simplest way to do that is to add the bin folder to the system path variable -
   * we have include batch scripts to run the samples which illustrate this.
   *
   * @param args - single string required, which is the full path of the folder to
optimize.
   *
   * @return - 0 for success, 1 for invalid arguments, 2 for an NXException, 3 for any
other exception.
   *
   */
  public static void main(String[] args)
  {

    if (args.length != 1)
    {
      System.err.println("Optimize a folder structure using the Java wrapper.");
      System.err.println("Usage:  java -cp .:nxwrapj.jar SampleJavaAdvanced
input_directory");
      System.err.println("input_directory:   Full path to directory structure to be
optimized.");
      System.exit(1);
    }
    File initialFolder = new File(args[0]);
    if (false == initialFolder.isDirectory() || false == initialFolder.exists())
    {
      System.err.println("Please supply a valid folder path.");
      System.exit(1);
    }
    int exitValue = 0;
    OptimizerLibrary library = null;
    try
    {
      library = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
```

```
OptimizerLibrary.NXSDK_VER_MINOR);
      final SampleJavaAdvanced sample = new SampleJavaAdvanced();
      if (sample != null)
      {
        sample.processFolder(initialFolder);
        sample.finished();
        // This calls the dispose function, which must be called once we've finished
work, and before
        // the OptimizerLibrary dispose function is called.
      }
    }
    catch (NXException nxe)
    {
      System.err.println("");
      System.err.println("NXPowerLiteException occurred: " + nxe.errorCode());
      exitValue = 2;
    }
    catch (UnsatisfiedLinkError ule)
    {
      //  This exception gets thrown if nxwrapj.dll is in an incorrect location or has
the wrong architecture.
      System.err.println("");
      System.err.println("UnsatisfiedLinkError with message: " + ule.getMessage());
      System.err.println("This probably means that the java code has been unable to
locate or load nxwrapj.dll," +
        " or the DLL is the wrong architecture");
      exitValue = 3;
    }
    catch (UserCancelledException uce)
    {
      System.err.println("");
      System.err.println("The optimization run has been cancelled");
      exitValue = 3;
    }
    catch (Exception e)
    {
      System.err.println("");
      System.err.println("A general exception occurred: " + e.getMessage());
      exitValue = 4;
    }
    finally
    {
      try
      {
        if (library != null)
        {
          library.dispose();
        }
      }
      catch (NXException e)
      {
        System.err.println("");
        System.err.println("NXPowerLiteException occurred: " + e.errorCode());
        exitValue = 2;
      }
    }
    System.exit(exitValue);
  }
}
```

errorCode

Signature

```
public NXException.NXErr errorCode();
```

Description

Returns the NXErr value that is set in the event of an **NXException** being thrown from the optimizer code.

Applies to:

NXException

Return value

The enum value NXexception.NXErr. See **NXException** for details of the enum.

Example code

```
try
{
  // Code using OptimizerLibrary and Optimizer
}
catch (NXException nxe)
{
  System.err.println("An error occurred within NXPowerLite: " + nxe.errorCode());
}
```

Callback interfaces

ICancelCallback

Description

This interface provides the mechanism to cancel the optimization of a file in mid flow. Clients can provide a concrete implementation of ICancelCallback which can then be passed as a parameter to **Optimizer.optimizeEx**. The concrete object will then be called at various points during optimization to provide the client with an opportunity to cancel the process.

If the concrete implementation of the method 'cancel()' returns 'true' then optimization will be stopped and cancelled; if it returns 'false' then optimization will continue.

Method

boolean cancel();

IProgressCallback

Description

This interface provides progress notification functionality. Clients can provide a concrete implementation of IProgress which can then be passed as a parameter to **Optimizer.optimizeEx**. The concrete object will then be notified of progress while optimization takes place.

Method

void progress(int total, int done);

Exceptions

NXException

Description

NXException inherits from java.lang.Exception and is thrown whenever an error occurs within the Java code. The nature of the error can be found by calling errorCode(), which will return one of the following values which map to the values in **Error Code Descriptions**:

```
public enum NXErr
{
  // General errors

nxerrNoError,
nxerrAlreadyInitialized,
nxerrNotInitialized,
nxerrFailedToInitialize,
nxerrBadHandle,
nxerrBadMemoryAllocation,
nxerrInvalidParameter,
nxerrOutOfDiskSpace,
nxerrSDKIncompatible,
nxerrSDKTooOld,
nxerrIncompatibleComponents,
nxerrUnexpectedException,
nxerrMoreData,
nxerrPropertyNotReady,
  // Internal errors
nxerrFailedToGeneratePipeName,
nxerrFailedToCleanupPipe,
nxerrFailedToSendMessageToOpt,
nxerrFailedToReadMessageFromOpt,
nxerrFailedToFindFreeOpt,
nxerrFailedToStartOpt,
nxerrFailedToTerminateOpt,
nxerrFailedToCloseOptHandle,
nxerrOptProcessTableIndexOutOfBounds,
nxerrOptPipeReadingFilenameFailed,
```

```
nxerrOptPipeReadingSettingsFailed,
nxerrOptPipeReadingVersionProtocolFailed,
nxerrFailedToDeleteOptFile,
nxerrFailedToCopyString,
nxerrFailedToSendProgressMsgRspToOpt,
nxerrFailedToReadProgressMsgFromOpt,
nxerrOptPipeReadingProgressMsgRspFailed,
nxerrOptPipeWritingProgressMsgFailed
}
```

This class is in the package com.neuxpower.nxpowerlite.sdk.

Method

getNXErr

Optimizer properties

BlockOnBusy

Description

This value affects the behavior of the return from a call to Optimize in the case where the Optimizer process is busy (e.g. the Optimizer is already optimizing a file as a result of a call from a different Optimizer object running on a different thread). If the property is set to 'true' then the calling thread will block until the Optimizer process becomes free before optimizing and then returning. If set to 'false', the thread will return immediately allowing the caller time to perform other actions before attempting to call Optimize again.

Signature

C# Property Setting **COPYCODE**

```
public bool BlockOnBusy
```

C++ Functions **COPYCODE**

```
void blockOnBusy(bool fBlockOnBusy);
bool blockOnBusy() const;
```

Java Method **COPYCODE**

```
public void blockOnBusy(boolean blockOnBusy) throws NXException;
public boolean blockOnBusy() throws NXException;
```

Applies to:

Optimizer (C#)

Optimizer (C++)

Optimizer (Java)

Value type

bool

Default value

true

Read/Write

Read and write.

Remarks/Notes

This property may throw the following exceptions, depending on which language you are programming in:

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

For a list of possible error codes that can be associated with the raised exception, see **Error Code Descriptions**.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.BlockOnBusy = false;
    string inputFilename;
    string outputFilename;
    // Get input/output filenames
    // ...
    // ...
    while(nxstatusOptBusy == opt.Optimize(inputFilename, outputFilename))
    {
      // Do some other processing...
      // sleep thread for 5 seconds
    }
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
```

```
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example COPYCODE

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.blockOnBusy(false);
while (nxstatusOptBusy == opt.optimize(inFileName.c_str(), outFileName.c_str()))
{
  // Sleep
}
```

Java Example COPYCODE

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.blockOnBusy(false);
while (Optimizer.nxstatusOptBusy == opt.optimize(inFileName, outFileName))
{
  // Sleep
}
```

SupportedExtensions

Description

This read-only property indicates the file extensions NXPowerLite expects to be able to optimize. The value of this property may change in future SDK versions.

Signature

C# Property Setting COPYCODE

```
public string SupportedExtensions
```

C++ Function COPYCODE

```
std::wstring supportedExtensions() const;
```

Java Methods COPYCODE

```
public String supportedExtensions() throws NXException;
```

Applies to:

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

string

Default value

"doc:docm:docx:dot:dotm:dotx:jfi:jfif:jif:jpe:jpeg:jpg:pdf:pot:potm:potx:pps:ppsm:ppsx:ppt:pptm:pptx:xls:xlsm:xlsx:xlt:xltm:xltx:tif:tiff:zip"

For a full explanation of supported file types and extensions please see **Supported File Extensions**.

Read/Write

Read only

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    string filter = opt.SupportedExtensions();
    // Manipulate 'filter' (e.g. remove some extensions) before passing to 'opt.ArchiveEntryExtensionFilter(filter)'.
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
std::wstring filter = opt.supportedExtensions();
// Manipulate 'filter' (e.g. remove some extensions) before passing to 'opt.ArchiveEntryExtensionFilter(filter)'.
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
String filter = opt.supportedExtensions();
// Manipulate 'filter' (e.g. remove some extensions) before passing to 'opt.ArchiveEntryExtensionFilter(filter)'.
// Set other properties
Optimizer.NXStatus nxStatus = opt.optimize(inFileString, outFileString);
```

LastFileHiddenContentRemoved

Description

This is a read-only property that determines if the successfully optimized document had its hidden content removed. This property will throw an exception if called before a successful call is made to Optimize() or OptimizeEx(). If the property returns true, then the hidden content in the optimized document was removed. If the property returns false then hidden content was not removed. Always returns ‘false’ for JPEG and TIFF file types. Returns ‘true’ if at least one file in a zip archive had hidden content removed.

Signature

C# Property Setting COPYCODE

```
public bool LastFileHiddenContentRemoved
```

C++ Function COPYCODE

```
bool lastFileHiddenContentRemoved() const;
```

Java Method COPYCODE

```
boolean lastFileHiddenContentRemoved() throws NXException;
```

Applies to:

Microsoft Office and PDF files.

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

false

Read/Write

Read only

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example COPYCODE

```
Language: CS
try
{
    using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
    using (Optimizer opt = new Optimizer())
    {
        // Alter any properties...
        opt.AllowCroppingOffice = true;
        // ...

        NXStatus nxStatus = opt.Optimize(inFileName, outFileName);
        if (NXStatus.nxstatusOk == nxStatus && opt.LastFileHiddenContentRemoved)
        {
            Console.WriteLine("SUCCESSFULLY OPTIMIZED AND HIDDEN CONTENT REMOVED");
```

```
        }
        else
        {
            Console.WriteLine("FAILED TO OPTIMIZE");
            // Although the SDK did not optimize the file, no error occurred. Query
nxStatus to determine why optimization did not take place.
        }
    }
}
catch (NXPowerLiteException ex)
{
    Console.WriteLine("");
    Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
    Console.WriteLine("Error code: " + ex.NXError.ToString());
    Console.WriteLine("Error date: " + ex.ErrorDate);
    Console.WriteLine("Error time: " + ex.ErrorTime);
    Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
    Console.WriteLine("");
    Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example COPYCODE

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
// Set properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
if(nxstatusOk == nxStatus && opt.lastFileHiddenContentRemoved())
{
  // Log the fact that the file just optimized had hidden content removed.
}
```

Java Example COPYCODE

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
Optimizer.NXStatus nxStatus = opt.optimize(inFileName, outFileName);
if(Optimizer.NXStatus.nxstatusOk == nxStatus && opt.lastFileHiddenContentRemoved())
{
  // Log the fact that the file just optimized had hidden content removed.
}
```

LastFileHadHiddenContent

Description

This is a read-only property that determines if the successfully optimized document contained any hidden content. This property will throw an exception if called before a successful call is made to Optimize() or OptimizeEx(). If the property returns true, then the optimized document had some hidden content. If the property returns false then the document did not have any hidden content. Always returns 'false' for JPEG and TIFF file types. Returns 'true' if at least one file (Microsoft Office or PDF file) in a zip archive contained hidden content.

Signature

C# Property Setting COPYCODE

```
public bool LastFileHadHiddenContent
```

C++ Function COPYCODE

```
bool lastFileHadHiddenContent() const;
```

Java Method COPYCODE

```
boolean lastFileHadHiddenContent() throws NXException;
```

Applies to:

Microsoft Office and PDF files.

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

false

Read/Write

Read only

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example COPYCODE

```
Language: CS
try
{
    using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
    using (Optimizer opt = new Optimizer())
    {
        // Alter any properties...
        opt.AllowCroppingOffice = true;
        // ...

        NXStatus nxStatus = opt.Optimize(inFileName, outFileName);
        if (NXStatus.nxstatusOk == nxStatus && opt.LastFileHadHiddenContent)
        {
            Console.WriteLine("SUCCESSFULLY OPTIMIZED AND FILE CONTAINED HIDDEN
CONTENT");
```

```
        }
        else
        {
            Console.WriteLine("FAILED TO OPTIMIZE");
            // Although the SDK did not optimize the file, no error occurred. Query
nxStatus to determine why optimization did not take place.
        }
    }
}
catch (NXPowerLiteException ex)
{
    Console.WriteLine("");
    Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
    Console.WriteLine("Error code: " + ex.NXError.ToString());
    Console.WriteLine("Error date: " + ex.ErrorDate);
    Console.WriteLine("Error time: " + ex.ErrorTime);
    Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
    Console.WriteLine("");
    Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
// Set properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
if(nxstatusOk == nxStatus && opt.lastFileHadHiddenContent())
{
  // Log the fact that the file just optimized had hidden content.
}
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
Optimizer.NXStatus nxStatus = opt.optimize(inFileName, outFileName);
if(Optimizer.NXStatus.nxstatusOk == nxStatus && opt.lastFileHadHiddenContent())
{
  // Log the fact that the file just optimized had hidden content.
}
```

LastFileType

Description

This is a read-only property that determines the file type of the successfully optimized document. This property will throw an NXPowerLiteException if called before a successful call is made to Optimize() or OptimizeEx(). The property returns a member of the enum NXFileType (see below) that describes the file type of the optimized file.

Signature

C# Property Setting **COPYCODE**

```
public NXFileType LastFileType
```

C++ Function **COPYCODE**

```
NXFileType lastFileType() const;
```

Java Method **COPYCODE**

```
NXFileType lastFileType() throws NXException;
```

Applies to:

All supported file types.

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

NXFileType

This type is as follows:

C# Example **COPYCODE**

```
public enum NXFileType
{
  nxftNotSupported, /* An Unsupported file type. */
  nxftPPF, /* Binary PowerPoint file  type. */
  nxftDOC, /* Binary Word file type. */
  nxftXLS, /* Binary Excel file type.*/
  nxftPPTX, /* XML format PowerPoint file type.*/
  nxftDOCX, /* XML format Word file type. */
  nxftXLSZ, /* XML format Excel file type.*/
  nxftJPEG, /* Jpeg file type. */
  nxftZIP, /* Zip file type. */
  nxftPDF, /* PDF file type. */
  nxftTIFF /* TIFF file type. */
}
```

C++ Example **COPYCODE**

```
public enum NXFileType
{
  nxftNotSupported, /* An Unsupported file type. */
  nxftPPF, /* Binary PowerPoint file  type. */
  nxftDOC, /* Binary Word file type. */
  nxftXLS, /* Binary Excel file type.*/
  nxftPPTX, /* XML format PowerPoint file type.*/
  nxftDOCX, /* XML format Word file type. */
  nxftXLSZ, /* XML format Excel file type.*/
  nxftJPEG, /* Jpeg file type. */
  nxftZIP, /* Zip file type. */
  nxftPDF, /* PDF file type. */
  nxftTIFF /* TIFF file type. */
}
```

Java Example **COPYCODE**

```
public enum NXFileType {
  nxftNotSupported, /* An unsupported file type. */
  nxftPPT, /* Binary PowerPoint file type. */
  nxftDOC, /* Binary Word file type. */
  nxftXLS, /* Binary Excel file type. */
  nxftPPTX, /* XML PowerPoint file type. */
  nxftDOCX, /* XML Word file type. */
  nxftXLSX, /* XML Excel file type. */
  nxftJPEG, /* Jpeg file type. */
```

```
  nxftZIP, /* Zip file type. */
  nxftPDF, /* PDF file type. */
  nxftTIFF; /* TIFF file type. */
}
```

Read/Write

Read only

Example code

C# Example **COPYCODE**

```
Language: CS
try
{
    using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
    using (Optimizer opt = new Optimizer())
    {
        // Alter any properties...
        opt.AllowCroppingOffice = true;
        // ...

        NXStatus nxStatus = opt.Optimize(inFileName, outFileName);
        if (NXStatus.nxstatusOk == nxStatus)
        {
            Console.WriteLine("SUCCESSFULLY OPTIMIZED");
            NXFileType ft = opt.LastFileType;
            // Use 'ft' to display the type of the file just optimized.
        }
        else
        {
            Console.WriteLine("FAILED TO OPTIMIZE");
            // Although the SDK did not optimize the file, no error occurred. Query
nxStatus to determine why optimization did not take place.
        }
    }
}
catch (NXPowerLiteException ex)
{
    Console.WriteLine("");
    Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
    Console.WriteLine("Error code: " + ex.NXError.ToString());
    Console.WriteLine("Error date: " + ex.ErrorDate);
    Console.WriteLine("Error time: " + ex.ErrorTime);
    Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
    Console.WriteLine("");
    Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
// Set properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
if(nxstatusOk == nxStatus)
{
```

```
    NXFileType ft = opt.lastFileType();
    // Log the fact that the file was optimized and use 'ft' to display its file type.
}
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
Optimizer.NXStatus nxStatus = opt.optimize(inFileName, outFileName);
System.out.println("File type is " + opt.lastFileType());
```

PDF file specific properties

AllowASCIIToBinaryPDF

Description

If set to true, ASCII85-encoded PDF files can be optimized. This allows greater reduction to be achieved on ASCII-only PDFs and PDFs containing ASCII-encoded images, but might prevent the optimized files from being processed by tools that cannot handle binary PDF files. We believe that these tools are rare, so it should be safe to enable this feature.

Signature

C# property setting **COPYCODE**

```
public bool AllowASCIIToBinaryPDF
```

C++ Functions **COPYCODE**

```
void allowASCIIToBinaryPDF(bool allowASCIIToBinaryPDF);
bool allowASCIIToBinaryPDF() const;
```

Java Methods **COPYCODE**

```
public void allowASCIIToBinaryPDF(boolean allowASCIIToBinaryPDF) throws NXException;
public boolean allowASCIIToBinaryPDF() throws NXException;
```

Applies to:

PDF files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

false

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.AllowASCIIToBinaryPDF = true;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.allowASCIIToBinaryPDF(true);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.allowASCIIToBinaryPDF(true);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

AllowResizingPDF

Description

This property determines whether or not images will be physically resized. If this property is 'true', any oversized images in PDF will be resized taking into account the **DotsPerInchPDF** value. If it is 'false', images will not be resized.

Signature

C# Property Setting **COPYCODE**

```
public bool AllowResizingPDF
```

C++ Functions **COPYCODE**

```
void allowResizingPDF(bool allowResizingPDF);
bool allowResizingPDF() const;
```

Java Methods **COPYCODE**

```
public void allowResizingPDF(boolean allowResizingPDF) throws NXException;
public boolean allowResizingPDF() throws NXException;
```

Applies to:

PDF files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

true

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example COPYCODE

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.AllowResizingPDF = true;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example COPYCODE

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.allowResizingPDF(true);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example COPYCODE

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.allowResizingPDF(true);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

AllowJPEGConversionPDF

Description

Determines whether or not non-JPEG images within PDF files will be compressed using the JPEG format, where appropriate. If this property is 'true', NXPowerLite will decide whether or not to use JPEG compression on a per-image basis. If it is 'false', NXPowerLite will not consider using JPEG compression. However, if the input file contains JPEG images, these may still be present as JPEGs in the output file.

Signature

C# Property Setting COPYCODE

```
public bool AllowJPEGConversionPDF
```

C++ Functions COPYCODE

```
void allowJPEGConversionPDF(bool allowJPEGConversionPDF);
```

```
bool allowJPEGConversionPDF() const;
```

Java Methods COPYCODE

```
public void allowJPEGConversionPDF(boolean allowJPEGConversionPDF) throws NXException;
public boolean allowJPEGConversionPDF() throws NXException;
```

Applies to:

PDF files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

true

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example COPYCODE

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.AllowJPEGConversionPDF = true;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
```

```
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.allowJPEGConversionPDF(true);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.allowJPEGConversionPDF(true);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

AllowRemoveHiddenContentPDF

Description

If this property is set to 'true' then private data will be removed from a PDF file when it is optimized. If 'false' then the private data will not be removed.

Applications that create PDFs files, such as Adobe Photoshop or Acrobat, are able to store information within a PDF file that they can use when opening or editing the file. This information can only be used by the application which created the file and is not needed to display a PDF file. For most cases we recommend deleting this data as it will have no effect on the use of the PDF file.

Signature

C# Property Setting **COPYCODE**

```
public bool AllowRemoveHiddenContentPDF
```

C++ Functions **COPYCODE**

```
void allowRemoveHiddenContentPDF(bool allowRemoveHiddenContentPDF);
bool allowRemoveHiddenContentPDF() const;
```

Java Methods **COPYCODE**

```
public void allowRemoveHiddenContentPDF(boolean allowRemoveHiddenContentPDF) throws
NXException;
public boolean allowRemoveHiddenContentPDF() throws NXException;
```

Applies to:

PDF files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

false

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.AllowRemoveHiddenContentPDF= true;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.allowRemoveHiddenContentPDF(true);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
```

```
Optimizer opt = new Optimizer();
opt.allowRemoveHiddenContentPDF(true);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

DotsPerInchPDF

Description

Sets the resolution (in dots per inch) of the target device. This is used in deciding the target dimensions for images in PDF files when scaling them down. This value is ignored if **AllowResizingPDF** is 'false'.

Signature

C# Property Setting **COPYCODE**

```
public int DotsPerInchPDF
```

C++ Functions **COPYCODE**

```
void dotsPerInchPDF(int dotsPerInchPDF);
int dotsPerInchPDF() const;
```

Java Methods **COPYCODE**

```
public void dotPerInchPDF(int dotPerInchPDF) throws NXException;
public int dotPerInchPDF() throws NXException;
```

Applies to:

PDF files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

int

Default value

200

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example COPYCODE

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.DotsPerInchPDF = 150;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example COPYCODE

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.dotsPerInchPDF(150);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example COPYCODE

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.dotsPerInchPDF(150);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

JPEGQualityPDF

Description

Determines the quality to be used when compressing images within files using the JPEG format. This is a number from 1-9 inclusive, where 1 is the lowest quality and 9 is the highest. This value will affect the size of your files as well as the quality of JPEG images. Higher quality will generally result in a larger file size, but better looking pictures.

Signature

C# Property Setting **COPYCODE**

```
public int JPEGQualityPDF
```

C++ Functions **COPYCODE**

```
void JPEGQualityPDF(int JPEGQualityPDF);
int JPEGQualityPDF() const;
```

Java Methods **COPYCODE**

```
public void JPEGQualityPDF(int JPEGQualityPDF) throws NXException;
public int JPEGQualityPDF() throws NXException;
```

Applies to:

PDF files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

int

Default value

7

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.JPEGQualityPDF = 8;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
```

```
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.JPEGQualityPDF(8);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.JPEGQualityPDF(8);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

Office file specific properties

AllowCroppingOffice

Description

Specifies whether cropping of images should be allowed or not in Office files. If this property is 'true', any portions of an image that are not visible in the file due to cropping will be removed.

Signature

C# Property Setting **COPYCODE**

```
public bool AllowCroppingOffice
```

C++ Functions **COPYCODE**

```
void allowCroppingOffice(bool allowCroppingOffice);
bool allowCroppingOffice() const;
```

Java Methods **COPYCODE**

```
public void allowCroppingOffice(boolean allowCroppingOffice) throws NXException;
public boolean allowCroppingOffice() throws NXException;
```

Applies to:

Office files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

false

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.AllowCroppingOffice = true;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.allowCroppingOffice(true);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.allowCroppingOffice(true);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

AllowResizingOffice

Description

This property determines whether or not images will be physically resized. If this property is 'true', any oversized images in PowerPoint files will be scaled down to suit the screen target dimensions specified in **TargetScreenHeightPowerPoint** and **TargetScreenWidthPowerPoint**. Images in Word and Excel will be resized taking into account the **DotsPerInchOffice** value. If it is 'false', images will not be resized.

Signature

C# Property Setting **COPYCODE**

```
public bool AllowResizingOffice
```

C++ Functions **COPYCODE**

```
void allowResizingOffice(bool allowResizingOffice);
bool allowResizingOffice() const;
```

Java Methods **COPYCODE**

```
public void allowResizingOffice(boolean allowResizingOffice) throws NXException;
public boolean allowResizingOffice() throws NXException;
```

Applies to:

Office files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

true

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.AllowResizingOffice = true;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.allowResizingOffice(true);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.allowResizingOffice(true);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

DotsPerInchOffice

Description

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.DotsPerInchOffice = 200;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.dotsPerInchOffice(200);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.dotsPerInchOffice(200);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

AllowRemoveHiddenContentOffice

Description

Embedded documents are items created by another application that have been embedded within a document, spreadsheet or presentation. Double-clicking on an embedded document will allow you to edit it with the application in which it was created. If the Optimizer is able to flatten embedded documents, it will convert them into images. Flattening embedded documents will reduce the size of your files considerably, but the embedded documents will no longer be editable.

If this property is 'true' and the Optimizer finds embedded documents within a Word, Excel® or PowerPoint® file, it will attempt to flatten them'.

If this property is set to 'false', flattening embedded objects will not be attempted.

Signature

C# Property Setting **COPYCODE**

```
public bool AllowRemoveHiddenContentOffice
```

C++ Functions **COPYCODE**

```
void allowRemoveHiddenContentOffice(bool allowRemoveHiddenContentOffice);
bool allowRemoveHiddenContentOffice() const;
```

Java Methods **COPYCODE**

```
public void allowRemoveHiddenContentOffice(boolean allowRemoveHiddenContentOffice)
throws NXException;
public boolean allowRemoveHiddenContentOffice() throws NXException;
```

Applies to:

Office files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

false

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example COPYCODE

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.AllowRemoveHiddenContentOffice = false;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example COPYCODE

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.allowRemoveHiddenContentOffice(false);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example COPYCODE

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.allowRemoveHiddenContentOffice(false);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

AllowJPEGConversionOffice

Description

Determines whether or not non-JPEG images within Office files will be compressed using the JPEG format, where appropriate. If this property is 'true', NXPowerLite will decide whether or not to use JPEG compression on a per-image basis. If it is 'false', NXPowerLite will not consider using JPEG compression. However, if the input file contains JPEG images, these may still be present as JPEGs in the output file.

Signature

C# Property Setting COPYCODE

```
public bool AllowJPEGConversionOffice
```

C++ Functions COPYCODE

```
void allowJPEGConversionOffice(bool allowJPEGConversionOffice);
```

```
bool allowJPEGConversionOffice() const;
```

Java Methods **COPYCODE**

```
public void allowJPEGConversionOffice(boolean allowJPEGConversionOffice) throws
NXException;
public boolean allowJPEGConversionOffice() throws NXException;
```

Applies to:

Office files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

true

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.AllowJPEGConversionOffice= false;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
```

```
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.allowJPEGConversionOffice(false);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.allowJPEGConversionOffice(false);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

JPEGQualityOffice

Description

Determines the quality to be used when compressing images within files using the JPEG format. This is a number from 1-9 inclusive, where 1 is the lowest quality and 9 is the highest. This value will affect the size of your files as well as the quality of JPEG images. Higher quality will generally result in a larger file size, but better looking pictures.

Signature

C# Property Setting **COPYCODE**

```
public int JPEGQualityOffice
```

C++ Functions **COPYCODE**

```
void JPEGQualityOffice(bool JPEGQualityOffice);
bool JPEGQualityOffice() const;
```

Java Methods **COPYCODE**

```
public void JPEGQualityOffice(int JPEGQualityOffice) throws NXException;
public int JPEGQualityOffice() throws NXException;
```

Applies to:

Office files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

int

Default value

7

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

Determines the quality to be used when compressing images within files using the JPEG format. This is a number from 1-9 inclusive, where 1 is the lowest quality and 9 is the highest. This value will affect the size of your files as well as the quality of JPEG images. Higher quality will generally result in a larger file size, but better looking pictures.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.JPEGQualityOffice = 5;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
```

```
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.JPEGQualityOffice(5);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.JPEGQualityOffice(5);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

TargetScreenHeightPowerPoint

Description

Sets the height in pixels of the target display size for files. This is used in deciding the target height of images when scaling them down. This is also used by PowerPoint to help calculate the resolution of optimized images. This value is ignored if **AllowResizingOffice** is 'false'.

Signature

C# Property Setting **COPYCODE**

```
public int TargetScreenHeightPowerPoint
```

C++ Functions **COPYCODE**

```
void targetScreenHeightPowerPoint(int targetScreenHeightPowerPoint);
int targetScreenHeightPowerPoint() const;
```

Java Methods **COPYCODE**

```
public int targetScreenHeightPowerPoint() throws NXException;
public void targetScreenHeightPowerPoint(int targetScreenHeightPowerPoint) throws
NXException;
```

Applies to:

Office files (PowerPoint only)

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

int

Default value

1200

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.TargetScreenHeightPowerPoint = 1200;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.targetScreenHeightPowerPoint(1200);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.targetScreenHeightPowerPoint(1200);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

TargetScreenWidthPowerPoint

Description

Sets the width in pixels of the target display size for files. This is used in deciding the target width of images when scaling them down. This is also used by PowerPoint to help calculate the resolution of optimized images. This value is ignored if **AllowResizingOffice** is 'false'.

Signature

C# Property Setting **COPYCODE**

```
public int TargetScreenWidthPowerPoint
```

C++ Functions **COPYCODE**

```
void targetScreenWidthPowerPoint(int targetScreenWidthPowerPoint);
int targetScreenWidthPowerPoint() const;
```

Java Methods **COPYCODE**

```
public int targetScreenWidthPowerPoint() throws NXException;
public void targetScreenWidthPowerPoint(int targetScreenWidthPowerPoint) throws
NXException;
```

Applies to:

Office files (PowerPoint only)

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

int

Default value

1600

Read/Write

Read and write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.TargetScreenWidthPowerPoint = 1600;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.targetScreenWidthPowerPoint(1600);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Method **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.targetScreenWidthPowerPoint(1200);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

MSOfficeOnly

Description

If this property is 'true', then files that are saved in Microsoft Office format will only be optimized if their metadata indicates that the application that last saved them was a Microsoft Office application. This may help to avoid problems with files that were created by third-party applications ("third-party documents").

When this property is set to 'true', attempting to optimize a third-party document will yield the status code nxstatusNotSavedByMSOffice.

Many third-party applications create files with metadata indicating that they were created with Microsoft Office, so setting this property to 'true' does not guarantee that only files created with

Microsoft Office will be optimized.

Signature

C# Property Setting **COPYCODE**

```
public bool MSOfficeOnly
```

C++ Functions **COPYCODE**

```
void msOfficeOnly(bool msOfficeOnly);
bool msOfficeOnly() const;
```

Java Methods **COPYCODE**

```
public void msOfficeOnly(boolean msOfficeOnly) throws NXException;
public boolean msOfficeOnly() throws NXException;
```

Applies to:

Office files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

false

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
```

```
    opt.MSOfficeOnly = true;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.msOfficeOnly(true);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.msOfficeOnly(true);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

JPEG file specific properties

AllowResizingJPEG

Description

This property determines whether or not images will be physically resized according to the **LongestDimensionJPEG** value. If AllowResizingJPEG is 'true', any oversized images will be scaled down. If it is 'false', images will not be resized.

Signature

C# Property Setting **COPYCODE**

```
public bool AllowResizingJPEG
```

C++ Functions **COPYCODE**

```
void allowResizingJPEG(bool allowResizingJPEG);
bool allowResizingJPEG() const;
```

Java Methods **COPYCODE**

```
public void allowResizingJPEG(boolean allowResizingJPEG) throws NXException;
public boolean allowResizingJPEG() throws NXException;
```

Applies to:

JPEG files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

false

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.AllowResizingJPEG = true;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
```

```
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.allowResizingJPEG(true);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.allowResizingJPEG(true);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

MinPixelSizeJPEG

Description

This property determines the minimum size in pixels that a JPEG file must be before it will be optimized.

Signature

C# Property Setting **COPYCODE**

```
public int MinPixelSizeJPEG
```

C++ Functions **COPYCODE**

```
void minPixelSizeJPEG(int minPixelSizeJPEG);
int minPixelSizeJPEG() const;
```

Java Methods **COPYCODE**

```
public void minPixelSizeJPEG(int minPixelSizeJPEG) throws NXException;
public int minPixelSizeJPEG() throws NXException;
```

Applies to:

JPEG files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

int

Default value

1470000 (e.g. 1400 x 1050)

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.MinPixelSizeJPEG = 1000000;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.minPixelSizeJPEG (1000000);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
```

```
Optimizer opt = new Optimizer();
opt.minPixelSizeJPEG(1000000);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

JPEGQualityJPEG

Description

Determines the quality to be used when compressing images within files using the JPEG format. This is a number from 1-9 inclusive, where 1 is the lowest quality and 9 is the highest. This value will affect the size of your files as well as the quality of JPEG images. Higher quality will generally result in a larger file size, but better looking pictures.

Signature

C# Property Setting **COPYCODE**

```
public int JPEGQualityJPEG
```

C++ Functions **COPYCODE**

```
void JPEGQualityJPEG(int JPEGQualityJPEG);
int JPEGQualityJPEG() const;
```

Java Methods **COPYCODE**

```
public void JPEGQualityJPEG(int JPEGQualityJPEG) throws NXException;
public int JPEGQualityJPEG() throws NXException;
```

Applies to:

JPEG files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

int

Default value

8

Read/Write

Read and write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.JPEGQualityJPEG = 6;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.JPEGQualityJPEG(6);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.JPEGQualityJPEG(6);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

AllowRemoveMetadataJPEG

Description

If this is set to 'true' all of the metadata associated with the JPEG image will be stripped. If this property is 'false' JPEG images will retain their original metadata (e.g. EXIF data).

Signature

C# Property Setting **COPYCODE**

```
public bool AllowRemoveMetadataJPEG
```

C++ Functions **COPYCODE**

```
void allowRemoveMetadataJPEG(bool allowRemoveMetadataJPEG);
bool allowRemoveMetadataJPEG() const;
```

Java Methods **COPYCODE**

```
public void allowRemoveMetadataJPEG(boolean allowRemoveMetadataJPEG) throws NXException;
public boolean allowRemoveMetadataJPEG() throws NXException;
```

Applies to:

JPEG files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

bool

Default value

false

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
```

```
    opt.AllowRemoveMetadataJPEG = true;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.allowRemoveMetadataJPEG(true);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.allowRemoveMetadataJPEG(true);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

LongestDimensionJPEG

Description

Specifies the size in pixels of the longest dimension size for JPEG files. This is used in deciding the target size of images when scaling them down. This value is ignored if **AllowResizingJPEG** is 'false'.

Signature

C# Property Setting **COPYCODE**

```
public int LongestDimensionJPEG
```

C++ Functions **COPYCODE**

```
void longestDimensionJPEG(int longestDimensionJPEG);
int longestDimensionJPEG() const;
```

Java Methods **COPYCODE**

```
public int longestDimensionJPEG() throws NXException;
public void longestDimensionJPEG(int longestDimensionJPEG) throws NXException;
```

Applies to:

JPEG files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

int

Default value

1600

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    opt.LongestDimensionJPEG = 1600;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example **COPYCODE**

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
opt.longestDimensionJPEG(1600);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example **COPYCODE**

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
opt.longestDimensionJPEG(1600);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

Zip file specific properties

ArchiveEntryExtensionFilter

Description

By default, all supported file types are optimized within a zip archive; potentially this includes files that have no file extension. However, this filter enables callers to set precisely which files within a zip archive are to be optimized by specifying the file extensions to be filtered.

Extensions are set using a filter string that is in a colon-separated format. For example, to filter (and, therefore, optimize) certain types of Word and Excel files you might set the filter string to "doc:docx:xls:xlsx". Files with other extensions and with no extension will be ignored.

To include the optimization of files that have no file extension within the zip archive the special tag <no_ext> must be included in the filter string, for example "doc:docx:xls:xlsx:<no_ext>".

To ensure that no files in a zip archive are optimized, set the filter string to be an empty string.

To reset the filter to the default behavior, set the filter string to "*".

Signature

C# Property Setting **COPYCODE**

```
public string ArchiveEntryExtensionFilter
```

C++ Functions **COPYCODE**

```
void archiveEntryExtensionFilter(const std::wstring& wstrFilterString);
std::wstring archiveEntryExtensionFilter() const;
```

Java Methods **COPYCODE**

```
public string archiveEntryExtensionFilter() throws NXException;
public void archiveEntryExtensionFilter(string filter) throws NXException;
```

Applies to:

Zip files

Optimizer in C#

Optimizer in C++

Optimizer in Java

Value type

string

Default value

"*"

(Optimize all supported file types.)

Read/Write

Read & write

Remarks/Notes

NXPowerLiteException in C# code.

NXException in C++ code.

NXException in Java code.

Example code

C# Example **COPYCODE**

```
try
{
  using (OptimizerLibrary lib = new OptimizerLibrary(OptimizerLibrary.NXSDK_VER_MAJOR,
OptimizerLibrary.NXSDK_VER_MINOR))
  using (Optimizer opt = new Optimizer())
  {
    string filter = @"doc:docx:ppt:pptx";
    opt.ArchiveEntryExtensionFilter = filter;
    // Set more properties and do optimization here
    // ...
    // ...
  }
}
catch (NXPowerLiteException ex)
{
  Console.WriteLine("");
  Console.WriteLine("A NXPowerLiteException occurred with message " + ex.Message);
  Console.WriteLine("Error code: " + ex.NXError.ToString());
  Console.WriteLine("Error date: " + ex.ErrorDate);
  Console.WriteLine("Error time: " + ex.ErrorTime);
  Console.WriteLine("The stack trace is " + ex.StackTrace);
}
catch (Exception ex)
{
  Console.WriteLine("");
  Console.WriteLine("A General exception occurred with message " + ex.Message);
}
```

C++ Example COPYCODE

```
// No exception handling included - may throw nxpowerlite::NXException
nxpowerlite::Optimizer opt;
std::wstring filter = L"doc:docx:ppt:pptx";
opt.archiveEntryExtensionFilter(filter);
// Set other properties
NXStatus nxStatus = opt.optimize(inFileString.c_str(), outFileString.c_str());
```

Java Example COPYCODE

```
// No exception handling included - may throw NXException
Optimizer opt = new Optimizer();
string filter = "doc:docx:ppt:pptx";
opt.archiveEntryExtensionFilter(filter);
Optimizer.NXStatus status = opt.optimize(inFileName, outFileName);
```

Optimizer error & status codes

Error Code Descriptions

Error codes fall into two ranges. Error codes less than 1000 describe errors that are general or most of which can be corrected from within the application itself. Error codes in the second range (from 1001 upwards) describe internal errors, indicating that something is wrong with the system and probably can't be fixed from within the application itself.

Error Code Names	Value	Description
nxerrNoError	0	The file was optimized successfully.
nxerrAlreadyInitialized	1	The SDK library has already been initialized.
nxerrNotInitialized	2	The SDK library has not yet been initialized. See **OptimizerLibrary**.
nxerrFailedToInitialize	3	TheSDK library failed to initialize.
nxerrBadHandle	4	The object handle is not valid.
nxerrBadMemoryAllocation	5	Memory could not be created when, for example, creating a handle.
nxerrInvalidParameter	6	A parameter passed to the function is invalid (e.g. an 'out' parameter has been passed as NULL).
nxerrOutOfDiskSpace	10	There is insufficient disk space to complete the operation requested.
nxerrSDKIncompatible	11	The major protocol version numbers of the SDK 'include' files and the SDK library .dll differ, making them incompatible.
nxerrSDKTooOld	12	The minor protocol version number of the SDK 'include' files is greater than the SDK library .dll (CAPI).
nxerrIncompatibleComponents	13	There is a version mismatch between two SDK components (e.g. .NET wrapper and C API, or C API and Optimizer) .
nxerrUnexpectedException	15	An unknown error has occurred.
nxerrMoreData	20	The buffer is not big enough to hold output.
nxerrPropertyNotReady	21	Applies to the read-only properties **LastFileHadHiddenContent**, **LastFileHiddenContentRemoved** and **LastFileType**. If a read-only property is called before a successful call to Optimize() or OptimizeEx() then this error will occur.
nxerrFailedToGeneratePipeName	1001	A unique named pipe name could not be generated.

nxerrFailedToCleanupPipe	1002	A pipe connecting to another process could not be cleaned up.
nxerrFailedToSendMessageToOpt	1003	A message could not be sent to the Optimizer process.
nxerrFailedToReadMessageFromOpt	1004	A message sent from the Optimizer process could not be read.
nxerrFailedToFindFreeOpt	1005	There are no free Optimizer processes available (the maximum number of Optimizer processes is specified in the **OptimizerLibrary** constructor).
nxerrFailedToStartOpt	1006	The Optimizer process failed to start.
nxerrFailedToTerminateOpt	1007	An Optimizer process failed to terminate successfully.
nxerrFailedToCloseOptHandle	1008	There was an error closing the process and thread handles of the Optimizer process.
nxerrOptProcessTableIndexOutOfBounds	1009	The index to the table that holds data on available Optimizer processes, is out of bounds.
nxerrOptPipeReadingfilenameFailed	1011	The Optimizer failed to read the filename sent to it.
nxerrOptPipeReadingSettingsFailed	1012	The Optimizer failed to read the settings sent to it.
nxerrOptPipeReadingVersionProtocolFailed	1014	There was a problem obtaining the version from the Optimizer process.
nxerrFailedToDeleteOptFile	1015	The optimized file which is larger or the same size as the original file, couldn't be deleted.
nxerrFailedToCopyString	1019	Failed to copy an internal string.
nxerrFailedToSendProgressMsgRspToOpt	1020	A failure occurred sending a progress message response to the Optimizer.
nxerrFailedToReadProgressMsgFromOpt	1021	A failure occurred reading a progress message from the Optimizer.
nxerrOptPipeReadingProgressMsgRspFailed	1022	This error can only arise if progress notification has been enabled by calling OptimizeEx(). The error occurs when the Optimizer process sends a notification to the c-api process but does not get a response.
nxerrOptPipeWritingProgressMsgFailed	1023	This error can only arise if progress notification has been enabled by calling OptimizeEx(). The error occurs when the Optimizer process is unable to send the progress message.

Status Code Descriptions

Status Code Name	**Value**	**Description**
nxstatusOk	2001	Status is Ok.
nxstatusErrorOccurred	2002	An error occurred.
nxstatusUnsupportedFileFormat	2003	The operation requested cannot be performed on the file passed in as it is an unsupported type for that operation.
nxstatusOptBusy	2004	The optimizer process is busy.
nxstatusOptimizerFailure	2005	The file is already in use.
nxstatusAlreadyOptimized	2006	The file has already been optimized.
nxstatusCannotReduce	2007	The file cannot be reduced in size with the optimization settings used.
nxstatusOfficeFastSavedDoc	2009	The file is in the Office 'fast saved' format.
nxstatusInvalidFormat	2010	The file looked valid initially but on further investigation turned out to be invalid.
nxstatusJPEGTooSmall	2011	The JPEG being optimized is too small. See **MinPixelSizeJPEG** to change the minimum size.

nxstatusFileVersionTooOld	2012	The version of the file is too old (e.g. pre-Office 97).
nxstatusUnsupportedPDFSubset	2019	The file being optimized is a PDF subset which is unsupported.
nxstatusNotSavedByMSOffice	2020	The file being optimized was not last saved by Microsoft Office, and **MSOfficeOnly** is enabled.
nxstatusFileVersionTooNew	2021	The file version is too new (not yet supported).
nxstatusDigitallySigned	2022	The file being optimized is digitally signed.
nxstatusEncryptedOrPassword	2023	The file is encrypted or password protected and cannot be opened because a password is required.
nxstatusFileNotFound	2024	The file specified does not exist.
nxstatusOptCrashed	2025	The Optimizer process has crashed.
nxstatusTaggingFailed	2026	The operation to tag the optimized file has failed.
nxstatusPDFFailure	2027	An unexpected error was returned by the PDF Library.
nxstatusFailedToCopyInputFile	2028	The file being optimized could not be copied to the user temporary directory.
nxstatusCannotCreateOutputFile	2029	The output file cannot be created.
nxstatusOutputFileExists	2030	The output file already exists.
nxstatusOptimizerFailure	2031	Unexpected error returned by the Optimizer process.
nxstatusUserCancelled	2032	The user cancelled the optimization operation.
nxstatusUnsupportedTIFFFile	2033	The file being optimized is a TIFF that contains compressed data. The SDK will only optimize TIFF files with uncompressed data.
nxstatusUnsupportedTIFFTags	2034	TIFF files are made up of data structures called tags. A tag has been encountered that the SDK doesn't recognize.