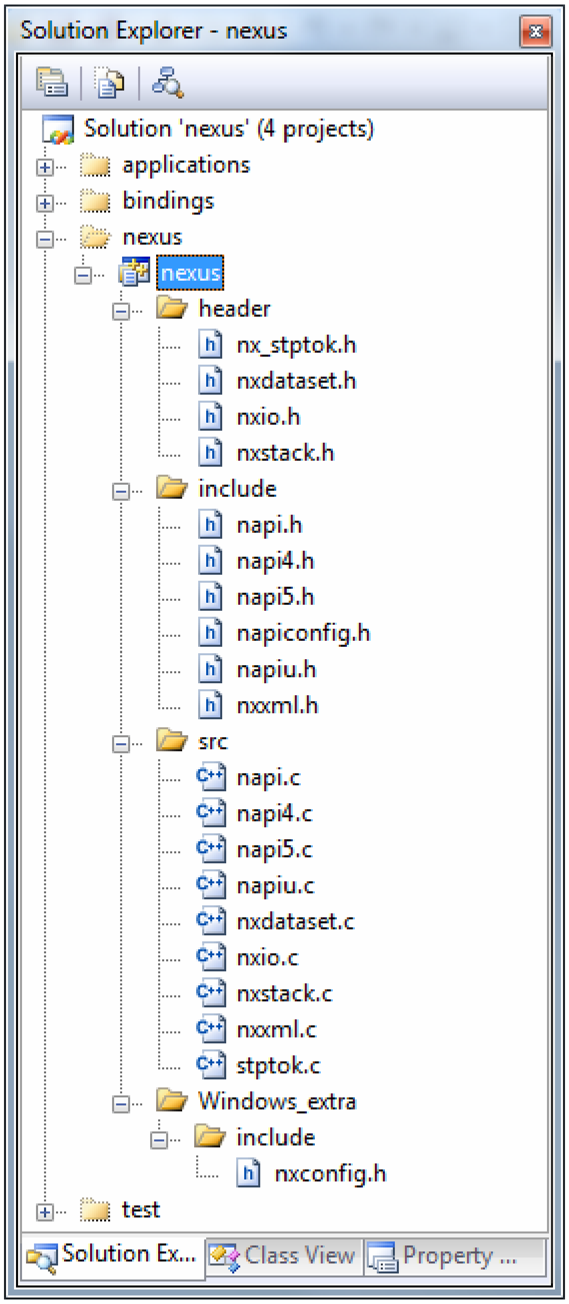
Building NeXus with Visual Studio 2008

Pedro Vicente (pedro.vicente@space-research.org)



1 Introduction

Open the file /windows/nexus.sln with Visual Studio.

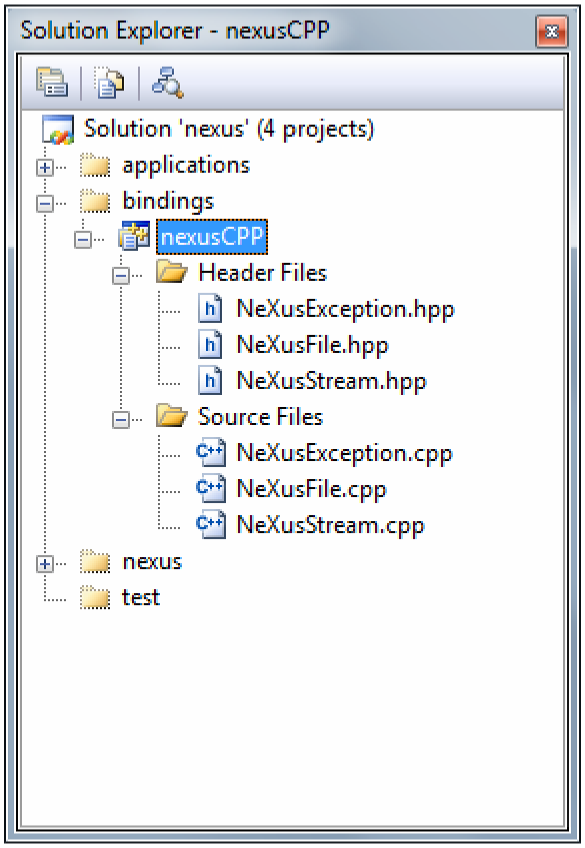
2 Layout

The solution layout is shown in the figure in the left, a Visual Studio solution with 4 folders: applications, nexus, bindings and test.

2.1 nexus folder

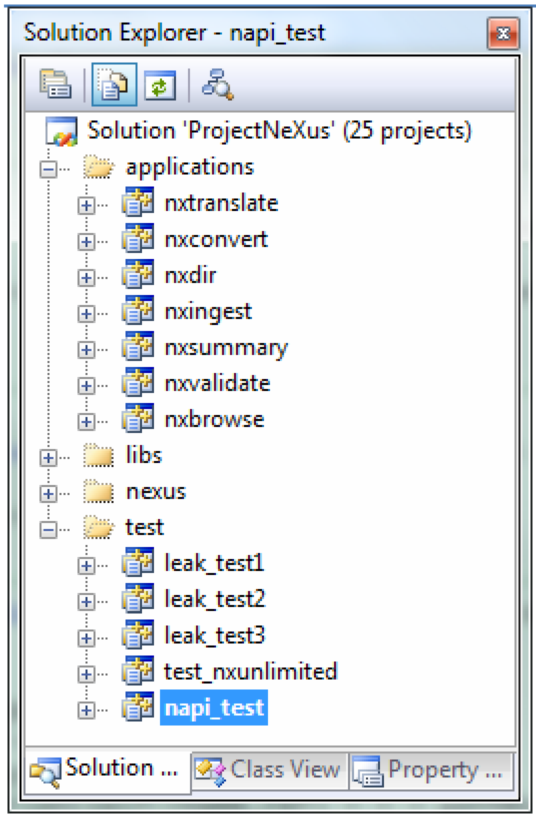
The folder nexus contains a project named nexus that contains the C source code for the NeXus API. The folder contains 4 subfolders named ‘header’ , the C header files currently located in /src of the NeXus distribution, ‘include’ , the C header files currently located in /include of the NeXus distribution, ‘src’ , the C source files currently located in /src of the NeXus distribution and ‘Windows_extra/include’ that mimics a folder with the same name in the NeXus distribution.

The ‘Windows_extra/include’ folder contains the ‘nxconfig.h’ file, that contains several Visual Studio system dependent macros for use in the NeXus API. Unlike the UNIX systems, this file is not generated automatically by the configure process in those systems.



2.2 bindings folder

The bindings folder contains a project named ‘nexusCPP’ that contains the C++ binding of the NeXus library.



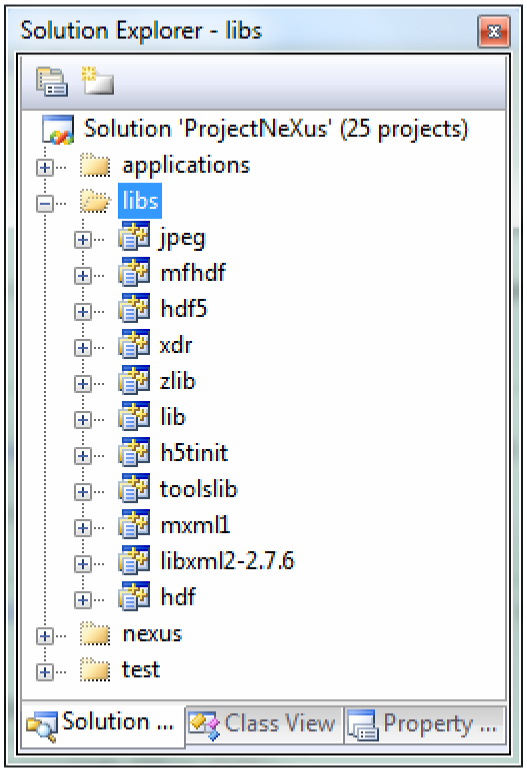
2.3 applications folder

The figure on the left contains the projects regarding the applications and test folders. For applications the projects are nxtranslate, nxconvert, nxdir, nxingest, nxsummary, nxvalidate and nxbrowse.

2.4 test folder

For the test folder, the projects are leak_test1, leak_test2, leak_test3, test_nxunlimited and napi_test.

These names duplicate the programs currently built for the UNIX systems.



3 Add additional libraries

A feature of the Visual Studio IDE is that it allows inserting and deleting projects by means of a Graphical User Interface. Thus, it is possible to include in the solution projects for the base libraries that NeXus depends, such as HDF5 (and its dependencies SZLIB and ZLIB), HDF4 (and its dependency JPEG), and the XML libraries. This allows advanced developers to have direct access to the code of the underlying libraries, for debugging purposes, for example. Since these external libraries are not distributed with NeXus, these projects are not proposed to be included in the Visual Studio Solution, but can be made available for interested developers.

4 Building NeXus with the base libraries

To actually store NeXus files on physical media, different low-level file formats are available, namely HDF4, HDF5, and XML. The NeXus library may be configured to support all of them, or any nonempty subset. Applications that create NeXus files need to decide (or let the user decide) in which low-level format data shall be stored. This section shows how to modify several Visual Studio project settings to build the NeXus API and applications with the underlying libraries HDF5, HDF4 or MXML.

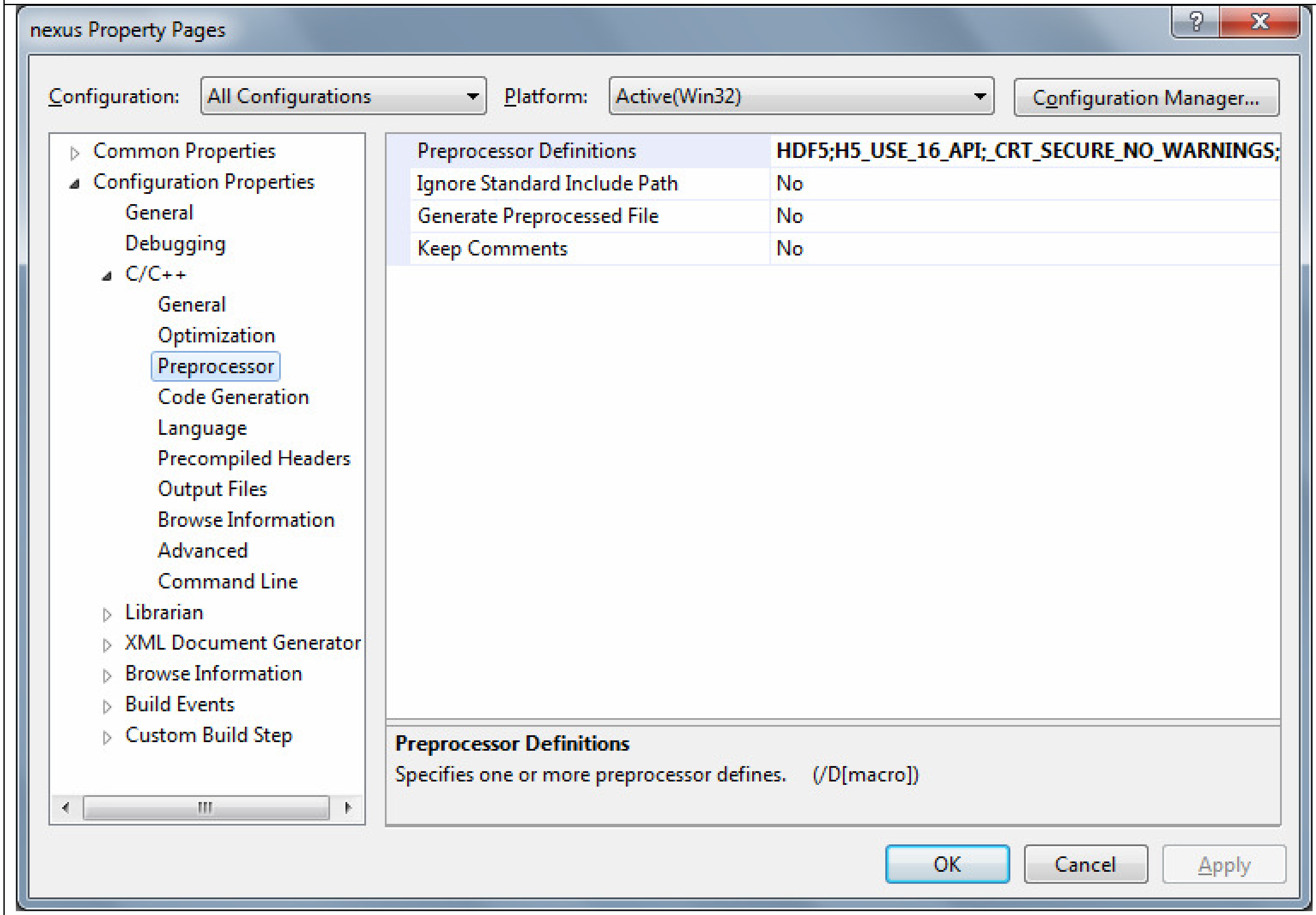
To compile and link a NeXus application with any underlying library, two actions must be done:

- add a preprocessor definition for the underlying library for the NeXus library Visual Studio project
- add the underlying library name and path for each application Visual Studio project

4.1 NeXus library Visual Studio project settings

The preprocessor definitions are entered in the C/C++, Preprocessor, Preprocessor Definitions Property Page, shown below

Figure 4.1: the C/C++, Preprocessor, Preprocessor Definitions

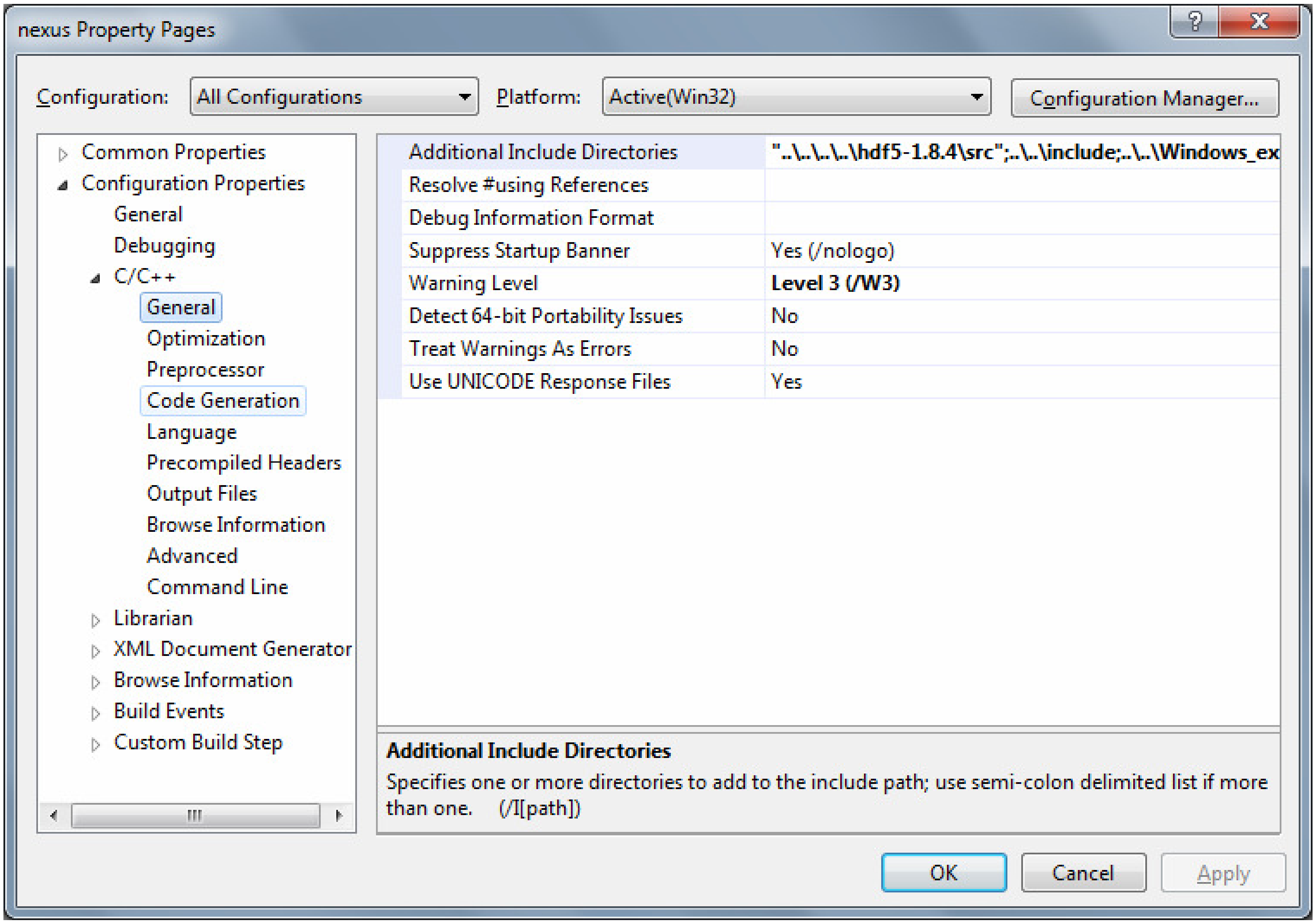


Preprocessor Definitions	HDF5;H5_USE_16_API;_CRT_SECURE_NO_WARNINGS;
Ignore Standard Include Path	No
Generate Preprocessed File	No
Keep Comments	No

Preprocessor Definitions
Specifies one or more preprocessor defines. (/D[macro])

The library path of the underlying library must also be entered. This is done in the C/C++, General, Additional Include Directories of the **nexus** project file, shown below.

Figure 4.2: the C/C++, General, Additional Include Directories



Additional Include Directories	"..\..\..\..\hdf5-1.8.4\src";..\..\include;..\..\Windows_ex
Resolve #using References	
Debug Information Format	
Suppress Startup Banner	Yes (/nologo)
Warning Level	**Level 3 (/W3)**
Detect 64-bit Portability Issues	No
Treat Warnings As Errors	No
Use UNICODE Response Files	Yes

Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

4.2 Applications project settings

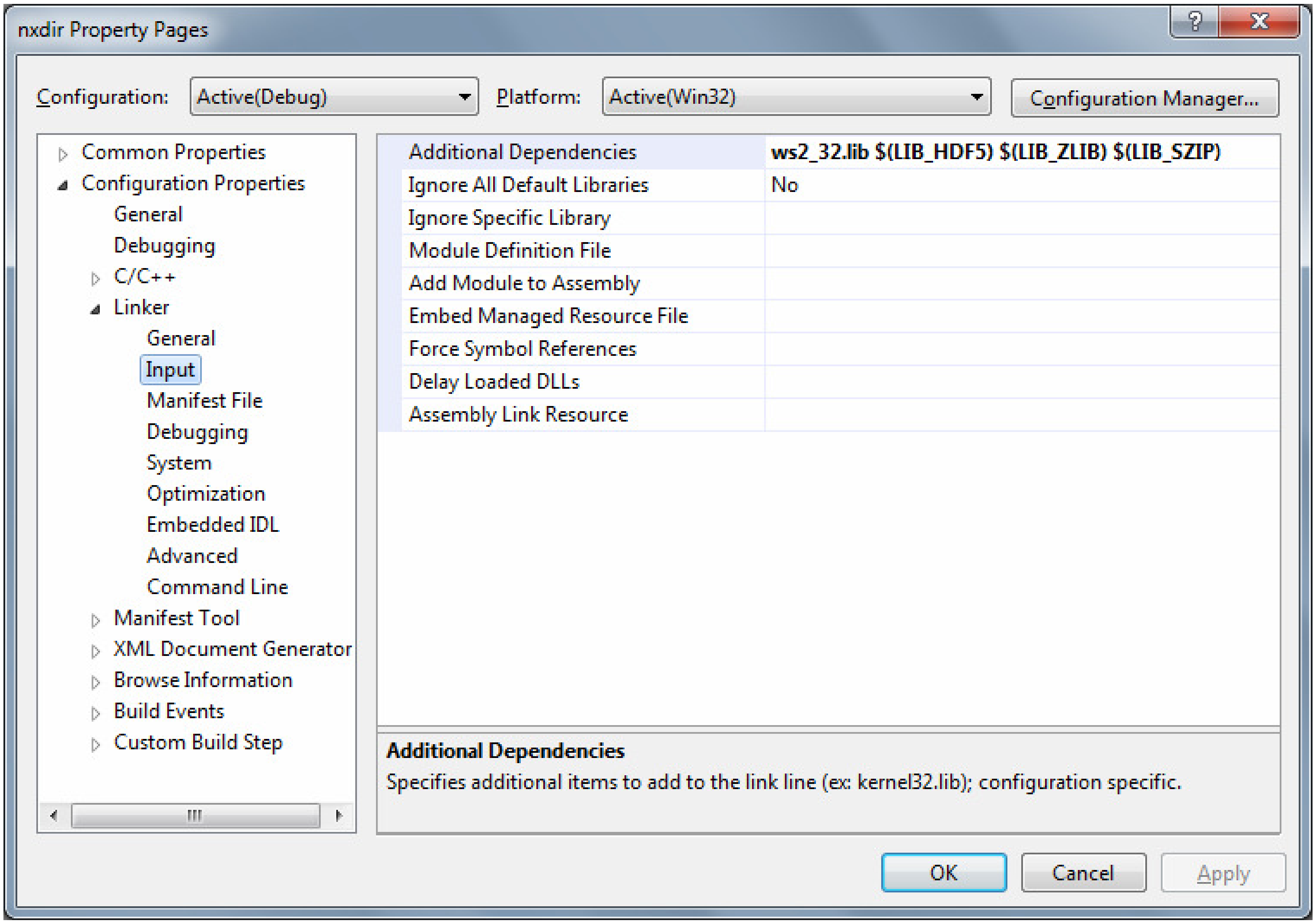
To library name to be linked is entered in the Linker Input Property Page (Figure 4.3), in the Additional Dependencies field. Either the literal library name or a Windows system environment variable[1] with the

[1] To define an environment variable in Windows 7, click Start, right-click Computer, and select Properties. Select Advanced System Settings. Then, click the Environment Variables button.

library name or name with full path can be used. Figure 4.3 shows an example of NXBrowse linked with the HDF5 library: The following environment variables were defined, for the HDF5, ZLIB and SZIP libraries[2], LIB_HDF5, LIB_ZLIB, LIB_SZIP with the names of those libraries. Then, in the NeXus application Linker Input Property Page, in the Additional Dependencies field, the following must be entered

$(LIB_HDF5) $(LIB_ZLIB) $(LIB_SZIP)

Figure 4.3: Linker Input Property Page and additional libraries in Additional Dependencies



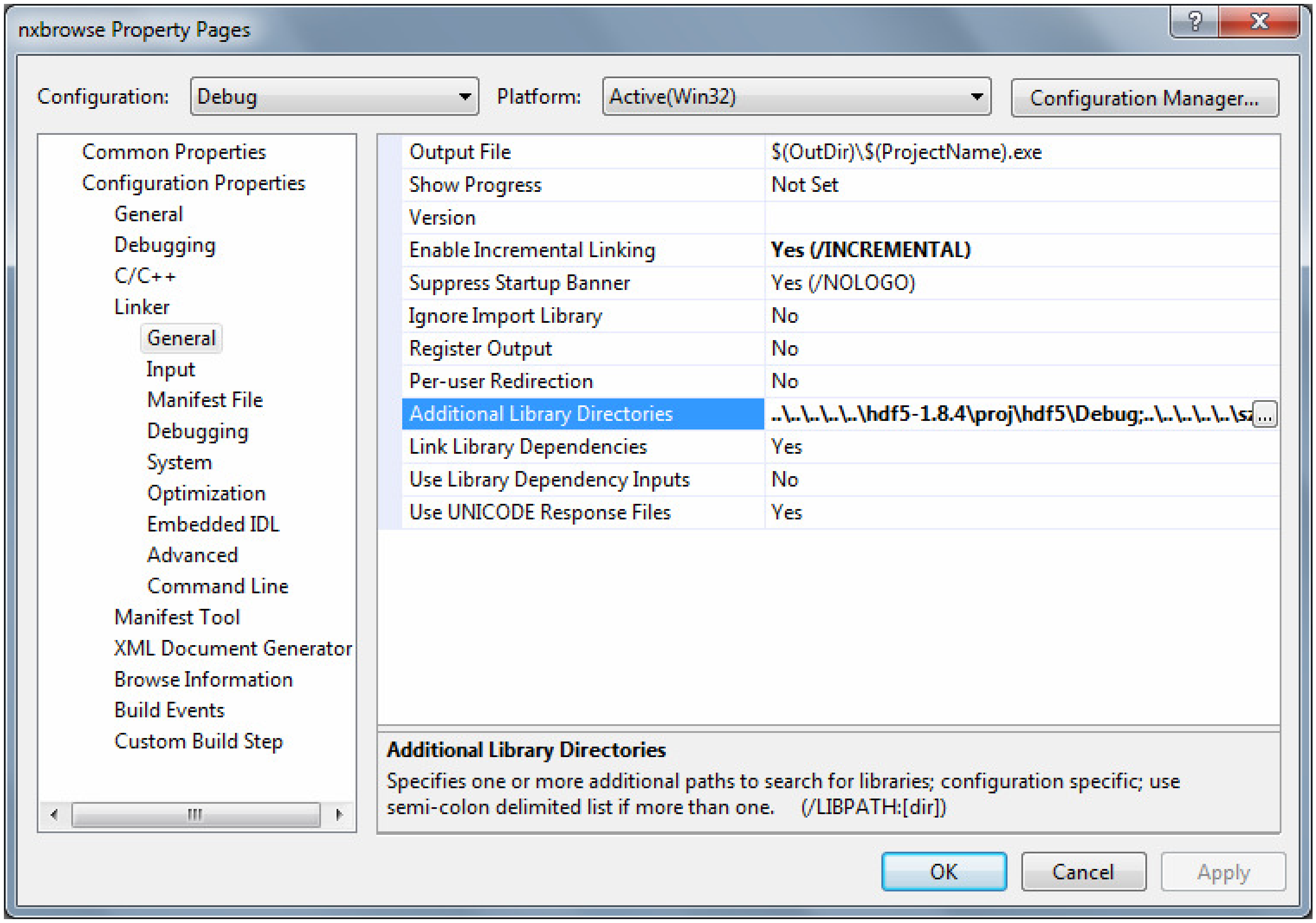
Additional Dependencies	**ws2_32.lib $(LIB_HDF5) $(LIB_ZLIB) $(LIB_SZIP)**
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

An additional step is to add the library path to be linked for each application. This is done in the Linker General, Additional Library Directories field, shown in Figure 4.4. This step can be omitted if the environment variable points to the full library path.

[2] The HDF5 library needs the SZIP and ZLIB libraries

Figure 4.4: Linker General, Additional Library Directories



Output File	$(OutDir)\$(ProjectName).exe
Show Progress	Not Set
Version	
Enable Incremental Linking	**Yes (/INCREMENTAL)**
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	**..\..\..\..\hdf5-1.8.4\proj\hdf5\Debug;..\..\..\..\sz**
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Use UNICODE Response Files	Yes

Additional Library Directories
Specifies one or more additional paths to search for libraries; configuration specific; use semi-colon delimited list if more than one. (/LIBPATH:[dir])

Both the HDF5 and HDF4 base libraries are distributed from the HDF Group (http://www.hdfgroup.org/). These instructions assume building using the HDF Group's distributed Visual Studio libraries.

4.3 Building NeXus with HDF5

The following table shows the preprocessor definitions and libraries needed to compile with the HDF5 library. The table also shows example names for environment variables for each underlying library. HDF5 depends also on external libraries (SZIP, ZLIB).

Preprocessor symbols	Libraries needed	Environment variables
HDF5, H5_USE_16_API	hdf5	LIB_HDF5
	szip	LIB_SZIP
	zlib	LIB_ZLIB

4.4 Building NeXus with HDF4

The following table shows the preprocessor definition and libraries needed to compile with the HDF4 library. The table also shows example names for environment variables for each underlying library. HDF4 depends also on external libraries (SZIP, ZLIB, JPEG) and one library included in the HDF4 distribution, XDR. The HDF4 Windows distribution actually generates 2 libraries: the single file HDF library, and the multi file HDF library, shown here as the LIB_HDF4 and LIB_MFHDF4 environment variable names.

Preprocessor symbols	**Libraries needed**	**Environment variables**
HDF4	hdf4	LIB_MFHDF4, LIB_HDF4
	xdr	LIB_XDR
	szip	LIB_SZIP
	zlib	LIB_ZLIB
	jpeg	LIB_JPEG

4.5 Building NeXus with MXML

The following table shows the preprocessor definition and library needed to compile with the Mini-XML library (http://www.minixml.org/). The table also shows an example name to use in the definition of an environment variable.

Preprocessor symbols	**Libraries needed**	**Environment variables**
MXML	mxml	LIB_MXML