NINTENDO

NITRO-SampleTools

QuickStart for TXLib

Ver 1.1.0

The contents in this document are highly confidential and should be handled accordingly.

NTR-06-0083-001-A
Released: 05/21/04

Table of Contents

List of Figures

NTR-06-0083-001-A
Released: 5/21/04

Revision History

Version	Date Revised	Revision Contents
1.1.0	4/23/2004	- Corrected errors. - Changed the header from NITRO to NITRO-SampleTools
1.0.0	3/5/2004	Initial release

1 About the TXLib Package

These are the main items packaged in the TXLib folder.

- QuickStart for TXLib
- TXLib Sample program source (`TXLib\build`)
- TXLib Manual (`TXLib\doc\TXLib_manual.pdf`)
- ntexconv Executable program (`TXLib\bin\Release\ntexconv.exe`)
- ntexconv Manual (`TXLib\doc\ntexconv_manual.pdf`)
- TextureViewer (`TXLib\TextureViewer`)

For information on the TXLib tree structure, refer to the directory map in `TXLib\ReadMe.txt`.

2 How to Proceed

QuickStart for TXLib explains the procedures required to set up the environment that you will need in order to program using TXLib. It does not describe TXLib. If you do not want to program, but simply want to make textures, proceed to “ntexconv Manual.”

The procedure for programming is as follows:

(1) Read this Manual.
(2) Read the ntexconv Manual.
(3) Try out the ntexconv executable program.
(4) Try out the TextureViewer.
(5) Read the TXLib Manual when you are programming using TXLib.

3 Development Environment Required When Programming Using TXLib

We have confirmed that it is currently possible to build with TXLib in the following Windows environments.

- Microsoft Windows 2000 Professional Service Pack 4

You must use the following tool to build and debug TXLib:

- Microsoft VisualC++ .net 2003

4 Procedure for Installing TXLib

(1) **Unzip the TXLib Package.**
This expands the TXLib package, which is compressed with WinZip. Use a decompression tool to expand it in the appropriate location.
This will create a folder named “TXLib.”

(2) **Move the TXLib Package to a Work Directory.**
Move the expanded TXLib folder to a working directory.

(3) **Set the Environment Variable.**
Set the TXLib environment variable as shown below.

Variable name:	TXLIB
Value:	The absolute path to the directory you moved to. (Example: `D:\home\SampleTools\TXLib`)

(4) **Build the TXLib Tree.**
Double click “`all.sln`” in `TXLib\build`, which will start Visual Studio .net 2003. After it starts, you will see a screen like the one shown below. Build will begin when you select Menu Bar > “Build” > “Build Solution.”

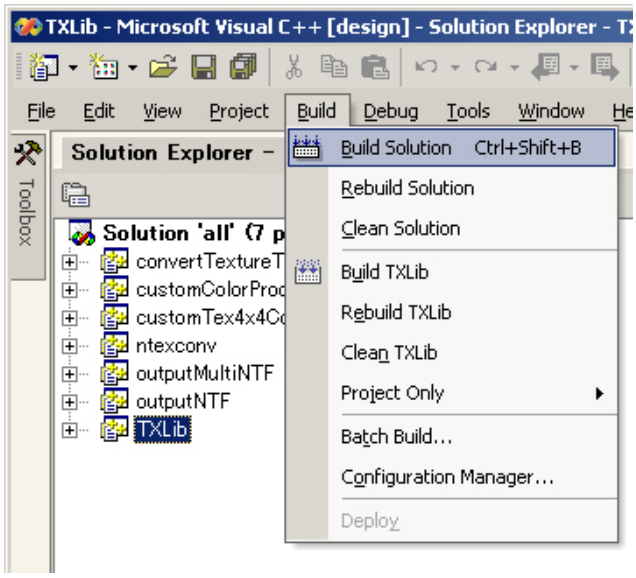

Figure 4-1 The Screen Opened by all.sln

(5) **Confirm TXLib Sample Program Operation.**

Run the sample program to confirm that the build succeeded.

By selecting Menu Bar > “Debug” > “Start Without Debugging,” all sample programs will start.

Some of the sample programs will request user input. In such cases, proceed by typing the appropriate response. If there are no problems, after each program completes “Press any key to continue” will indicate that the program has run successfully.

For information on running one sample program as well as other information on operating VisualStudio, see VisualStudio Help.

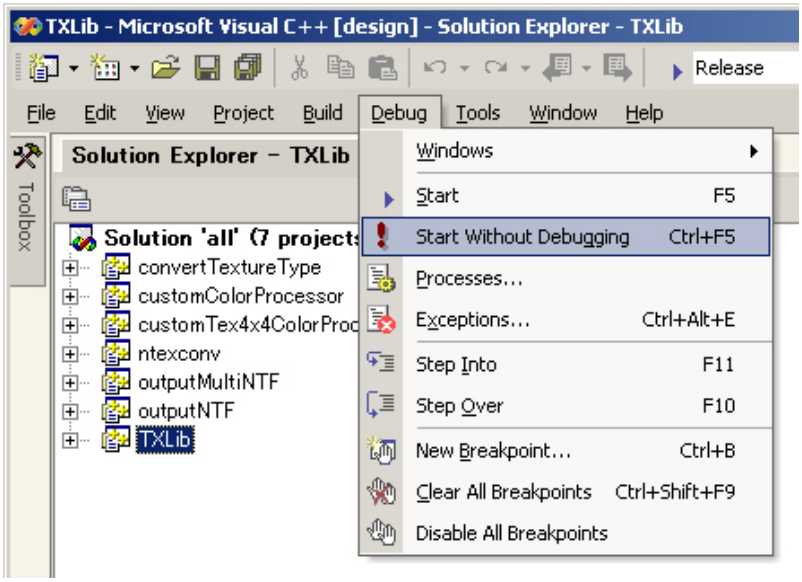

Figure 4-2 Confirming Sample Program Operation

5 Procedure for Creating TXLib Sample Programs

5.1 Adding a New Project to the Solution

A Solution manages all of the TXLib Projects. Sample programs, including TXLib, are bundled in one Solution file: `all.sln`. When you make a new sample program, start by adding a new Project here.

With Solution “all” selected, right-click to open the pull-down menu, and select “Add” > “New Project.”

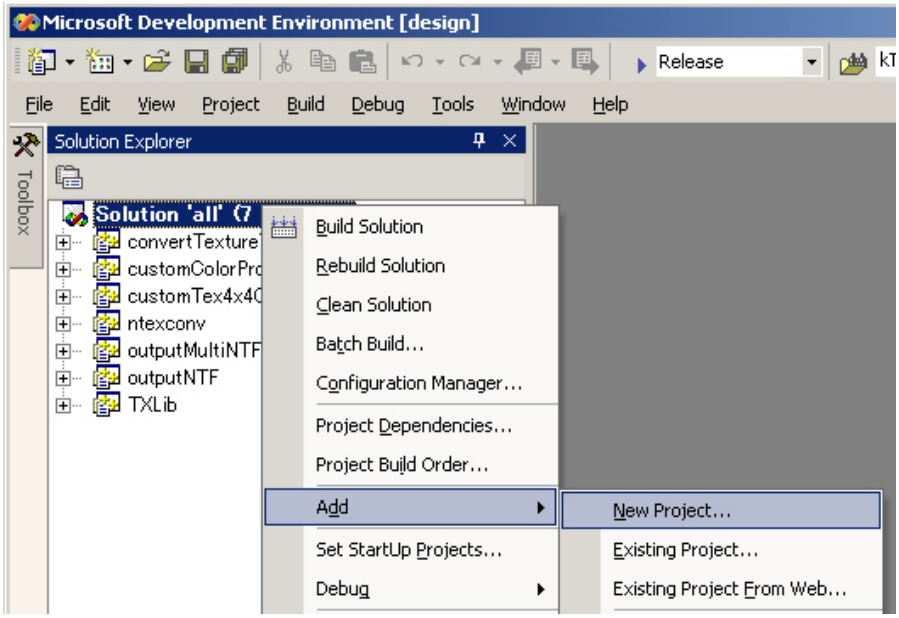

Figure 5-1 Adding a New Project to the Solution

When the dialogue box appears, select the type of Project that you want to create.

To create a simple sample, select “Win32 Console Project.”

If you want to save the Project, save it in “`TXLib\build\samples`.”

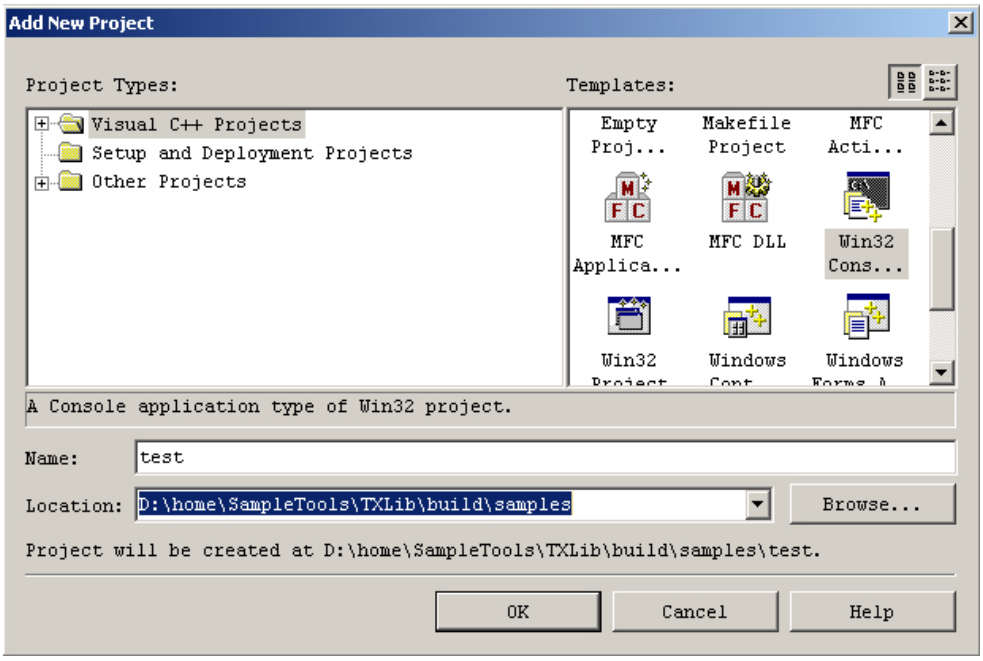

Figure 5-2 Selecting the Project Type

Next, the “Application Type Setting” dialogue box appears. Select "Console application". Uncheck the “Empty project” check box. Notice that if you do check it, “`stdafx.h`” will not be created. This is explained in the following paragraphs.

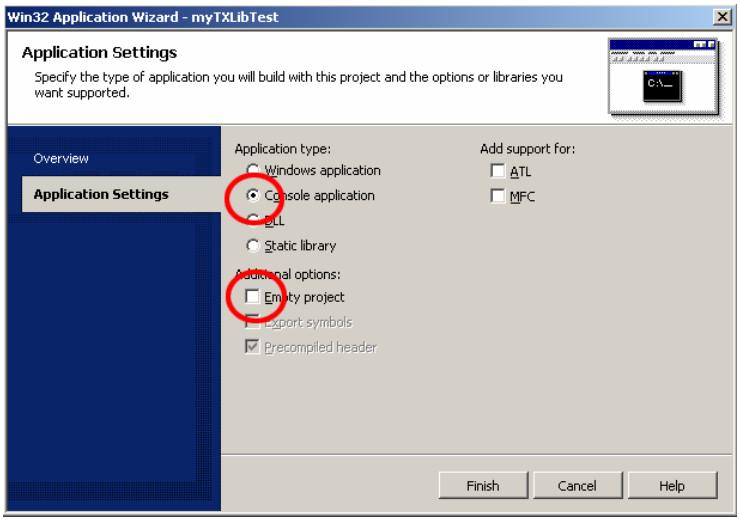

Figure 5-3 Application Settings

5.2 Confirming the Files in the Project

Once the Project has been created, the Solution Explorer will look like Figure 5-4. Confirm that "`stdafx.h`" is there. When you use TXLibUtility, an error will occur if this file is not in the Project. This is only necessary when using TXLibUtility.

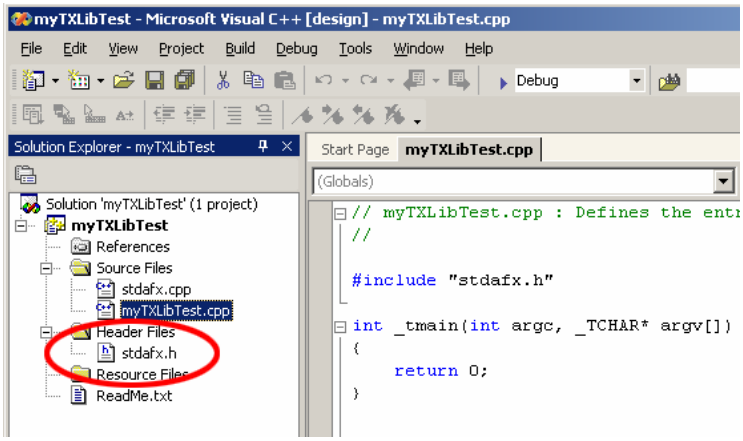

Figure 5-4 Confirming the File "stdafx.h"

5.3 Setting the Include and Library Directories

Set the directories that contain `TXLib.h` and `TXLib.lib` in the test Project.

With the "test" Project selected, right-click to open the pull-down menu, and then select Properties.

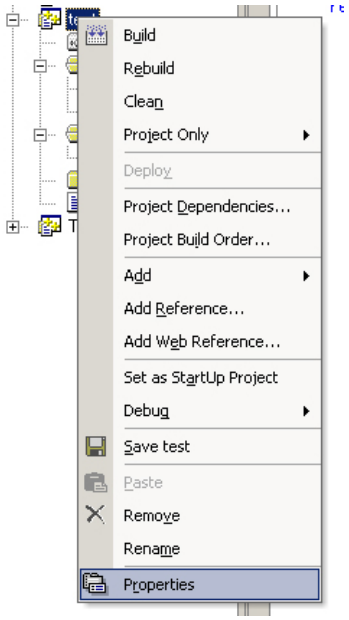

Figure 5-5 Setting the Include and Library Directories 1

In the Project Setting dialogue box that appears, configure the settings as follows.

- “C/C++” > “General” > “Add Include Directory”: `$(TXLIB)\include`
- “Linker” > “Input” > “Additional Dependant File”: `$(TXLIB)\lib\TXLib.lib`
- Set both of these in “Configuration” > “Debug,” and “Configuration” > “Release.”

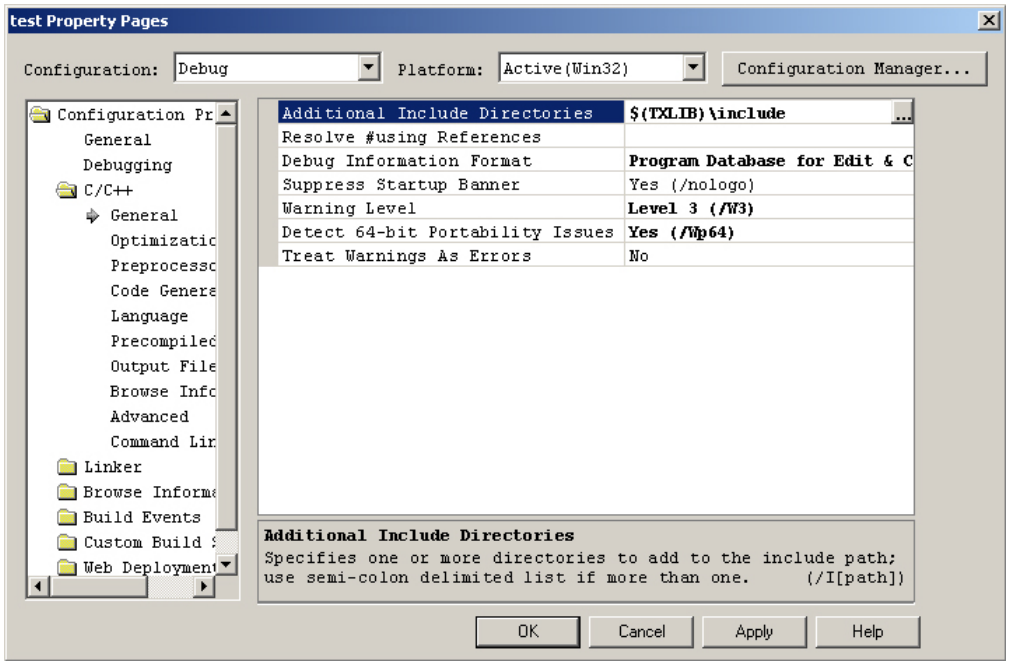

Figure 5-6 Setting the Include and Library Directories 2

5.4 Setting TXLib.dll Copy

Set to copy `TXLib.dll` to the current Project folder at build time.

This is set in “Build Event” > “Events After Build” > “Command Line.”

The figure below shows the setting for copying to the debugging folder the `TXLib.dll` used for debugging. If you do not set this, you will have to manually copy `TXLib.dll` to the current Project folder. There must be a `TXLib.dll` in the folder for the sample program to run.

As in the previous section, you must set this in both Debug and Release.

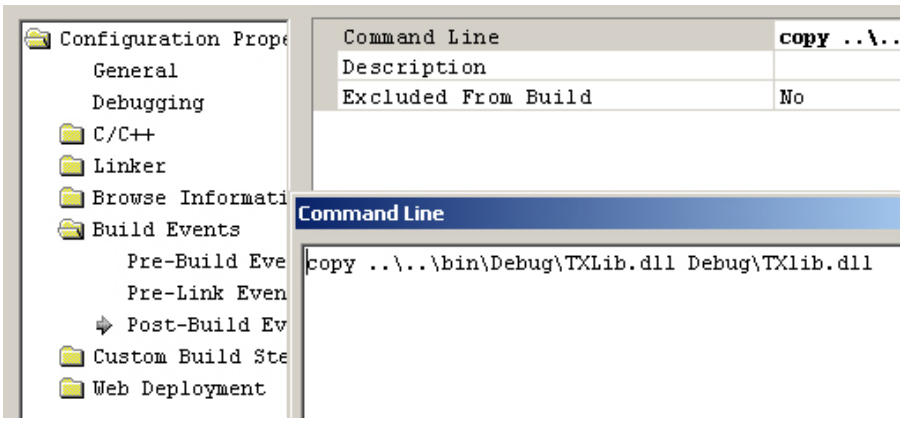

Figure 5-7 Setting TXLib.dll Copy

5.5 Setting Project Dependency Relationships

Since the Project is dependent on TXLib, you must set build accordingly.

With the Project “test” selected, right-click to open the pull-down menu, and then select “Project Dependency Relationships.”

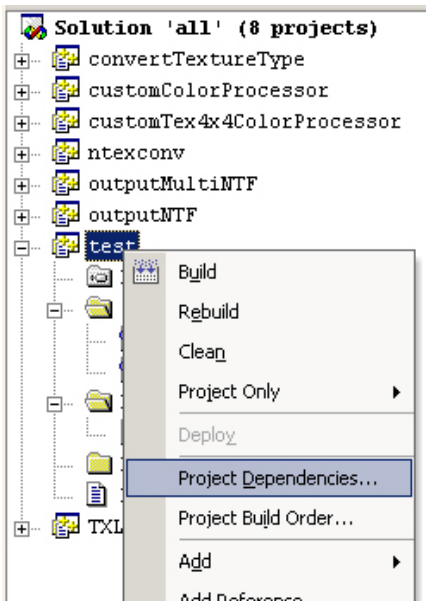

Figure 5-8 Setting Project Dependency Relationships

Next, a dialogue box like this one will appear. Check TXLib to set the dependency to TXLib.

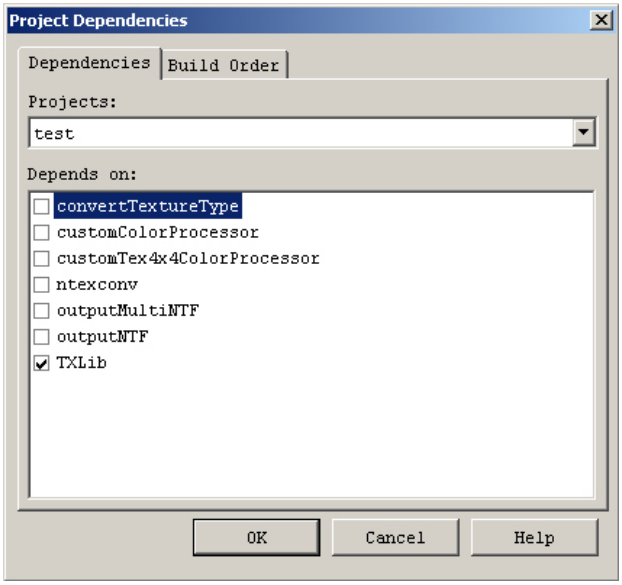

Figure 5-9 Setting the Project to be Dependent on TXLib

5.6 Copying TXLibUtility to the Project Folder

When using TXLibUtility, add its source and header files to the Project. This is only required if you are using TXLibUtility.

It is easiest to copy individual folders from a Project that is in the same Solution.

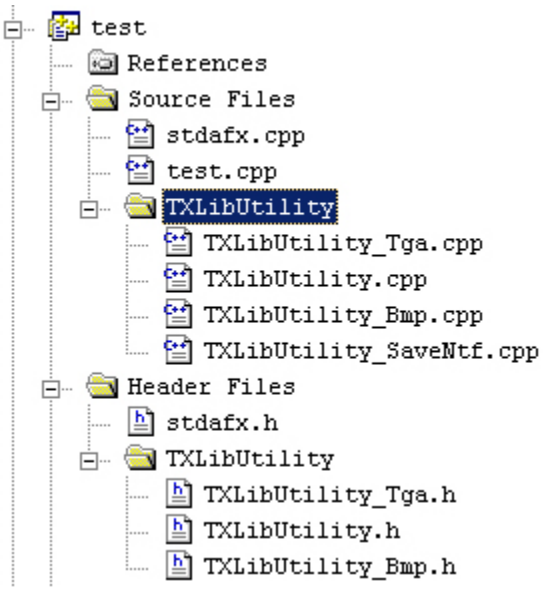

Figure 5-10 Copying TXLibUtility

5.7 Write Code in Source Files

Write code in a source file.

By copying the content of ”outputNTF .cpp“ of the “outputNTF” Project that is in the same Solution, you can easily check behavior.

The TXLibUtility path for #include is a relative path when the current Project folder is under `TXLib\build\samples\`.

test.cpp

(Globals) _tmain

```
// test.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <cstdio>
#include <TXLib.h>
#include "../TXLibUtility/include/TXLibUtility.h"

int _tmain(int argc, _TCHAR* argv[])
{
    TXLib::CTexture texture;

    // Input image files -> CTexture conversion
    if( !TXLibUtility::Load(
                &texture,                                                    // CTexture
                "../common/inputFiles/mario_128x128_palette256.tga" // Input file
    ) )
    {
        printf("Failed to read image file.\n"); exit(1);
    }

    // NTF output
    if( !TXLibUtility::SaveNtf(
                "outputFiles/after.c",      // output NTF path name
                &texture,                   // CTexture
                false,                      // output text format
                NULL,                       // input image file path name
                false,                      // not output TextureViewer file
                true                        // alignment NTF image size
    ) )
    {
        printf("Failed to output NTF file. \n"); exit(1);
    }

    printf("Succeeded. \n");

    return 0;
}
```

Figure 5-11 Example of Source File Code

5.8 Build

When you are finished coding, build the Project.

5.9 Run

If no errors occur during the build, run the program to see whether it behaves properly.

Because the code does not contain the NTF output destination folder, an error will occur. If you create an “outputFiles” folder under the Project and then run the program, NTF will be output to outputFiles with no problems.

The current Solution is set to “Multi Startup Projects.” Change this to “Single Startup Project” in the following way:

(1) With Solution “all” selected, right-click to open the pull-down menu and select Properties.

(2) When the dialogue box appears, select Single Startup Project, and specify the Project name “test.”

5.10 Customizing Source Code

In order to get used to TXLib, try customizing the source code.

For example, begin with some easy operations like changing the input image file names, outputting in binary format, or changing arguments.

We have prepared a number of samples in the Solution for your reference. We have also described the sample programs in the next chapter. See the descriptions and programs for more information.

6 About the Sample Programs

We have prepared a number of sample programs in `TXLib\build\samples` as examples of using TXLib. We explain those sample programs here.

Of these sample programs, we created “ntexconv” as a real tool for creating NITRO texture data. For details on how to use it, see the included document “`ntexconv_manual.pdf`.”

- **TXLibUtility**
  This sample library is very useful when using TXLib. This library is above TXLib, and has classes for handling BMP and TGA, and functions for handling files.
  This sample library can be used by all of the sample programs in the “samples” folder.

- **TXLibUtilityEx**
  This library has functions in addition to those in TXLibUtility. Currently it has functions for outputting data files and header files for TextureViewer, and a class for handling BG output. Currently only “ntexconv” can use this library.

- **ntexconv**
  This sample program is a practical console base for outputting NTF. From the command line, this sample program reads BMP and TGA files, converts texture formats, and outputs NTF. You can place a script in the command line, to run multiple command line commands as a group.

- **outputNTF**
  This simple sample program reads 1 BMP file and outputs NTF.

- **outputMultiNTF**
  This sample program reads multiple TGA, BMP files, and outputs NTF.

- **convertTextureType**
  In accordance with the command line, this sample program reads BMP and TGA files, converts texture, and outputs NTF.

- **customColorProcessor**
  This sample program has a customized color processing class. The functions that are customized reduce color when they convert direct color images into palette format texture.

- **customTex4x4ColorProcessor**
  This sample has a customized color processor class. The functions that are customized extract colors 0-4 from blocks when converting to tex4x4.