Guide to Building and Installing OpenEBTS

OpenEBTS is a library that enables the reading and writing of NIST files. It is available as a regular C library and as a set of Java classes.
OpenEBTS is simple to build, install and run. OpenEBTS can run on Windows and Linux, 32-bit or 64-bit, in UNICODE or MBCS. It can now also run on Android.

Table of Contents

Windows

On Windows OpenEBTS is a 32-bit or 64-bit Dynamically Linked Library. For an application to use it all that is necessary is to include **`OpenEBTS.h`** and optionally **`OpenEBTSErrors.h`**, link to the import library **`OpenEBTS.lib`** (or **`OpenEBTSa.lib`**) and then make **`OpenEBTS.dll`** (or **`OpenEBTSa.dll`**) accessible to the calling application.

Note that in the OpenEBTS package's folders you will see files with extensions **`.sh`**, **`.mk`**, **`.cproject`**, and **`.project`**, and files with the name **`makefile`**.

These files are solely used for building on the Linux platform, so can be ignored or even deleted if working only under Windows.

Building

OpenEBTS is easily built by opening the **`OpenEBTS.sln`** in Visual Studio 2008 SP1. There are 4 projects within the solution:

1. **OpenEBTS**

   This project builds the main OpenEBTS DLL. It statically links with Boost's regex library and the NBISWSQ which supports the WSQ image format. It dynamically links to the FreeImage library which provides image format conversion support and to the cURL library which is used for FTP downloads. Boost regex is version 1.35, NBISWSQ is based on NBIS sources from Release 3.3.1, FreeImage is version 3.15 and cURL is version 7.20.1.

   Note that the Boost regex and NBISWSQ binaries are already provided for both Debug and Release configurations, but is desired they can be built by using **`Regex.vcproj`** and **`NBISWSQ.vcproj`**, respectively.

2. **OpenEBTSSample1**

   This Win32 Console application demonstrates basic functionality by creating a new FBI CAR transaction containing textual and image information. It also demonstrates verification against a Verification File.

3. **OpenEBTSSample2**

   This Win32 Console application demonstrates the image format capabilities of OpenEBTS by creating an EBTS file containing images of all possible formats and subsequently re-opening it an exporting all the images within the EBTS file to disk to test the successful functioning.

4. **OpenEBTSViewer**

   This is an MFC application that shows the contents of EBTS files. It may be practical to map the .ebts extension (or .eft, etc..) to this application to allow quick viewing of their contents.

Note that to run OpenEBTSSample1 or OpenEBTSSample2, the environment variable ***OPENEBTSSAMPLESFOLDER*** must first be set to the absolute path of the folder containing the sample images and Verification File, situated under ./OpenEBTSSamples/Samples relative to the OpenEBTS package root.

This environment variable **is not** required to be set for the regular functioning of OpenEBTS or the OpenEBTSViewer.

There are 4 configurations for each of these 4 projects, **`Debug`**, **`DebugA`**, **`Release`**, and **`ReleaseA`**, and there are 2 supported platforms, Win32 and x64, so performing a Batch Build of all projects and configurations will create 32 binaries.

The configurations ending in 'A' build the MBCS versions of the binaries and the ones without this suffix build the UNICODE versions. The MBCS binaries are suffixed with an 'a', e.g., **`OpenEBTSa.dll`**.

To determined ex post facto what type of binary was built, the OpenEBTS dll's version information will contain configuration and platform in the *File Description* field, such as “Release x86” or “Debug x64”, for example.

To remove all files created by the build process you can run the **`clean_All.bat`** file from the root of the OpenEBTS package.

Installing

Installation is achieved by copying **`OpenEBTS.dll`** (or **`OpenEBTSa.dll`**) in a location accessible by the operating system, such as the SYSTEM32 folder, the calling application's folder or any folder listed under the PATH environment variable.

The same must be done for the Dynamically Linked Library **`FreeImage.dll`**.

Testing

To ensure the build and/or installation went smoothly you can run **`OpenEBTSSample1.exe`** and **`OpenEBTSSample2.exe`** (or **`OpenEBTSSample1a.exe`** and **`OpenEBTSSample2a.exe`** if testing the MBCS versions).

Make sure the environment variable ***OPENEBTSSAMPLESFOLDER*** is properly set.

The successful run of the first sample should output

```
Successful creation of OpenEBTSSample1_out.ebts
OpenEBTSSample1 has completed.
```

and create the file **`OpenEBTSSample1_out.ebts`** in the samples folder. This file can then be viewed with the OpenEBTSViewer to ensure its contents are accurate.

The successful run of the second sample should output

```
Successful creation of OpenEBTSSample2_out.ebts
Successful extraction of images.
```

```
OpenEBTSSample2 has completed.
```

and create 50 image files and **`OpenEBTSSample2_out.ebts`** in the samples folder. Since this sample program took images of various formats and bit depths and converted them to various other formats, analyzing the resulting images can be useful in determining if there are any image-related problems. Once again OpenEBTSViewer can be used to view **`OpenEBTSSample2_out.ebts`**.

Running **`removeOutputFiles.bat`** will purge the samples folder from any files created during the execution of the sample programs.

Linux

On Linux OpenEBTS is a 32-bit or 64-bit Dynamic Shared Object. For an application to use it all that is necessary is to include **`OpenEBTS.h`** and optionally **`OpenEBTSErrors.h`**, link to the import library **`libOpenEBTS.a`** (or **`libOpenEBTSa.a`**).

Building OpenEBTS with a 32-bit version of Linux will create the 32-bit version of OpenEBTS and building OpenEBTS with a 64-bit version of Linux will create the 64-bit version of OpenEBTS.

Note that in the OpenEBTS package's folders you will see files with extensions **`.sln`**, **`.vcproj`**, **`.rc`**, and **`.bat`**.

These files are solely used for building on Windows platforms, so can be ignored or even deleted if working only under Linux.

Also note that the OpenEBTSViewer is based on MFC and as such is a Windows-only application. If working only under Linux the entire **`OpenEBTSViewer`** folder can be ignored or even deleted.

Building

Before building on Linux, you must make sure that 3 libraries are already installed: cURL, Boost and FreeImage. The first two are most likely already installed. To get more information on these libraries please visit their respective website:

cURL:	http://curl.haxx.se/download.html
Boost:	http://www.boost.org/users/download/
FreeImage:	http://freeimage.sourceforge.net/download.html

FreeImage will probably not be installed – however after downloading the freeimage sources from sourceforge.org all that is required to build everything is '**`make`**' and all that is required to install is '**`make install`**' with su privileges. See the **`README.linux`** in the root of the downloaded FreeImage package.

OpenEBTS is then easily built by running the shell script **`build_All.sh`**. This will build the 4 projects: the NBISWSQ static library, the OpenEBTS shared library and the two sample programs OpenEBTSSample1 and OpenEBTSSample2.

Note that the shell scripts, such as **`build_All.sh`**, **`clean_All.sh`** and **`removeAllSamples.sh`** may need to have their permissions set to executable prior to running.

The build process builds Debug and Release configuration of the NBISWSQ static library, and builds 4 configurations of each of the other 3 binaries, namely **`Debug`**, **`DebugA`**, **`Release`**, and **`ReleaseA`**, so the build process actually creates 14 binaries in total.

The configurations ending in 'A' build the MBCS versions of the binaries and the ones without this suffix build the UNICODE versions. The MBCS binaries are suffixed with an 'a', e.g., **`libOpenEBTSa.so`**.

To remove all files created by the build process you can run the **`clean_All.sh`** file from the root of the OpenEBTS package.

Finally, to build OpenEBTS with the Eclipse CDT, the 4 projects NBISWSQ, OpenEBTS, OpenEBTSSample1 and OpenEBTSSample2 can easily be imported at once by using the menu **File**, menuitem **Import**..., section **General**, selection **Existing Projects into Workspace**. All 4 projects will be added at once, since the **`.project`** and **`.cproject`** files are present for each project in their respective subdirectories.

Installing

Installation can be done in one of two ways.

If actively developing the OpenEBTS library, then the OpenEBTS library can be left where it gets built, and the environment variable **`LD_RUN_PATH`** then set to the absolute path of its location, for example **`/home/bob/workspace/OpenEBTS/Debug`**. This will allow applications to locate the library at run-time.

If wanting to install the library in a more definitive manner then it should be copied to **`/usr/lib`** (or **`/usr/lib64`**) with su permissions. Copying **`libOpenEBTS.a`** (or **`libOpenEBTSa.a`**) to the same folder and copying **`OpenEBTS.h`** and **`OpenEBTSErrors.h`** to **`/usr/include`** is also recommended, as this will allow other applications to easily compile and link against the import library.

Testing

To ensure the build and installation went smoothly you can run **`OpenEBTSSample1`** and **`OpenEBTSSample2`** (or **`OpenEBTSSample1a`** and **`OpenEBTSSample2a`** if testing the MBCS versions).

Note that to run OpenEBTSSample1 or OpenEBTSSample2, the environment variable ***OPENEBTSSAMPLESFOLDER*** must first be set to the absolute path of the folder containing the sample images and Verification File, situated under ./OpenEBTSSamples/Samples relative to the OpenEBTS package root.

This environment variable **is not** required to be set for the regular functioning of OpenEBTS.

The successful run of the first sample should output

```
Successful creation of OpenEBTSSample1_out.ebts
OpenEBTSSample1 has completed.
```

and create the file **`OpenEBTSSample1_out.ebts`** in the samples folder.

The successful run of the second sample should output

```
Successful creation of OpenEBTSSample2_out.ebts
Successful extraction of images.
OpenEBTSSample2 has completed.
```

and create 50 image files and **`OpenEBTSSample2_out.ebts`** in the samples folder. Since this sample program took images of various formats and bit depths and converted them to various other formats, analyzing the resulting images can be useful in determining if there are any image-related problems.

Running the script **`removeOutputFiles.sh`** will purge the samples folder from any files created during the execution of the sample programs.

Java

JNI functions are automatically exported from the OpenEBTS library, so using OpenEBTS from Java is as simple as including **`OpenEBTS/Java/OpenEBTS/OpenEBTS.jar`** in a Java project.

Object oriented Java classes have been created to wrap the C OpenEBTS interface making any OpenEBTS Java function simple to perform. The classes provided are:

`NISTFile`
This main class encapsulates a NIST transaction file.

`NISTRecord`
`NISTField`
`NISTSubfield`
`NISTItem`
These 4 classes represent the hierarchical data structure of the NIST file contents.

`NISTVerification`
This class encapsulates an OpenEBTS Verification File.

`NISTTransactionCategories`
`NISTTransactionCategory`
`NISTTransactionList`
`NISTTransaction`
`NISTRecordOccurrencesList`
`NISTFieldDefinitionList`
`NISTFieldDefinition`
`NISTFieldValueList`
These last 8 classes represent the contents of an OpenEBTS Verification File.

There are 3 Java samples that demonstrate the use of these classes: *SampleCreateFile*, *SampleReadFile* and *SampleReadVerification*. The best way to explore them is to instruct Eclipse to Switch Workspaces to the **`OpenEBTS/Java`** folder and import all 4 projects from that same folder.

These samples have been tested with Eclipse Helios and JRE 6 on both Windows 7 and Fedora Linux.

Android

OpenEBTS is available for Android 2.2 and up. This section describes the steps to take to run the library using the Android emulator.

Building the Samples

Above all, building OpenEBTS for Android requires setting up the proper Android development environment. Google's Android Developer site is well written and contains invaluable information: http://developer.android.com/guide/developing/index.html

The steps involved in getting set up quickly are:

1. Install the Android SDK.
2. Install Eclipse, if not already installed.
3. Install JDK 6, if not already installed.
4. Install Google's Eclipse ADT (*Android Development Tools*) plug-in.
5. In Eclipse set the Android SDK folder via menu **Window**, menuitem **Preferences**, section **Android**, setting **SDK Location**.
6. In Eclipse ensure that the Java compliance level is set to 1.5 via menu **Window**, menuitem **Preferences**, section **Java**, subsection **Compiler** setting **Compiler compliance level**.

There are 3 sample Android applications (*activities*) provided, which are almost identical to the 3 Java samples from the **`OpenEBTS/Java`** folder. They are *CreateFile*, *ReadFile* and *ReadVer*. To build them simply instruct Eclipse to Switch Workspaces to the **`Android`** folder and import the 4 projects from under this same folder.

Note that you may receive several "Unable to open class file [...]R.java: No such file or directory" messages. To solve this just close and re-open Eclipse.

This gives Eclipse the opportunity to realize that the ADT has automatically re-created the **`gen`** subfolders.

Installing & Testing

Before an Android Activity can be run an AVD (*Android Virtual Device*) must be created. The appropriate settings needed to define an existing device can be easily found on the internet.

Also, before running the samples some sample files must be copied to the AVD's sdcard under a separate folder, **`OpenEBTSSamples`**. To create the folder on the emulator's sdcard, use the "**adb shell**" command from the Android SDK's **`platform-tools`** folder, then simply do:

```
cd sdcard
mkdir OpenEBTSSamples
```

Then in Eclipse, from the **File Explorer** of the **DDMS perspective**, the new folder will be visible as a subfolder of **`sdcard`**, and the sample files can be copied into it. The required files are the Verification File **`EBTS9.1_ENUS.txt`**, its four associated **`T2_*.txt`** files, **`finger8.bmp`**, and **`NISTFile.ebts`**, which can all be found in the **`OpenEBTSSamples/Samples`** folder.

On success, *CreateFile* will create the file **`SampleCreateFile_out.ebts`** in the **`sdcard/OpenEBTSSamples`** folder, *ReadFile* will output the textual record contents of the file **`NISTFile.ebts`** to a new file **`SampleReadFile_out.txt`**, and *ReadVer* will output detailed information about the EBTS 9.1 Verification File to the file **`SampleReadVerification_out.txt`**.

There are many things that could go wrong as a successful run requires several things to be done with precision, and this document cannot cover all possibilities, but thankfully the internet is a great place to find Android development discussions. The **LogCat** window in Eclipse will also be invaluable in following the exact actions of any running Android activity.

Building the Library

This step is optional and is only necessary if the OpenEBTS Android library needs to be rebuilt, as the 2 AMR processor versions of this library are already provided prebuilt under the **`Android/OpenEBTS/libs`** folder.

To build the library the Android NDK (*Native Developer Kit*) must first be installed.

Then the NDK command **`ndk-build`** must be called from the folder **`Android/OpenEBTS/jni`**, either from **Cygwin** on Windows or from Linux. Note that building on Linux with the NDK is several times faster than when using Cygwin from Windows.

All tests have been performed with the Android emulator emulating the Dell Streak, Samsung Galaxy and Motorola Xoom, with the following setup: Windows 7 x64, Eclipse Helios, Android SDK 10, ADT 10 and NDK 5b.