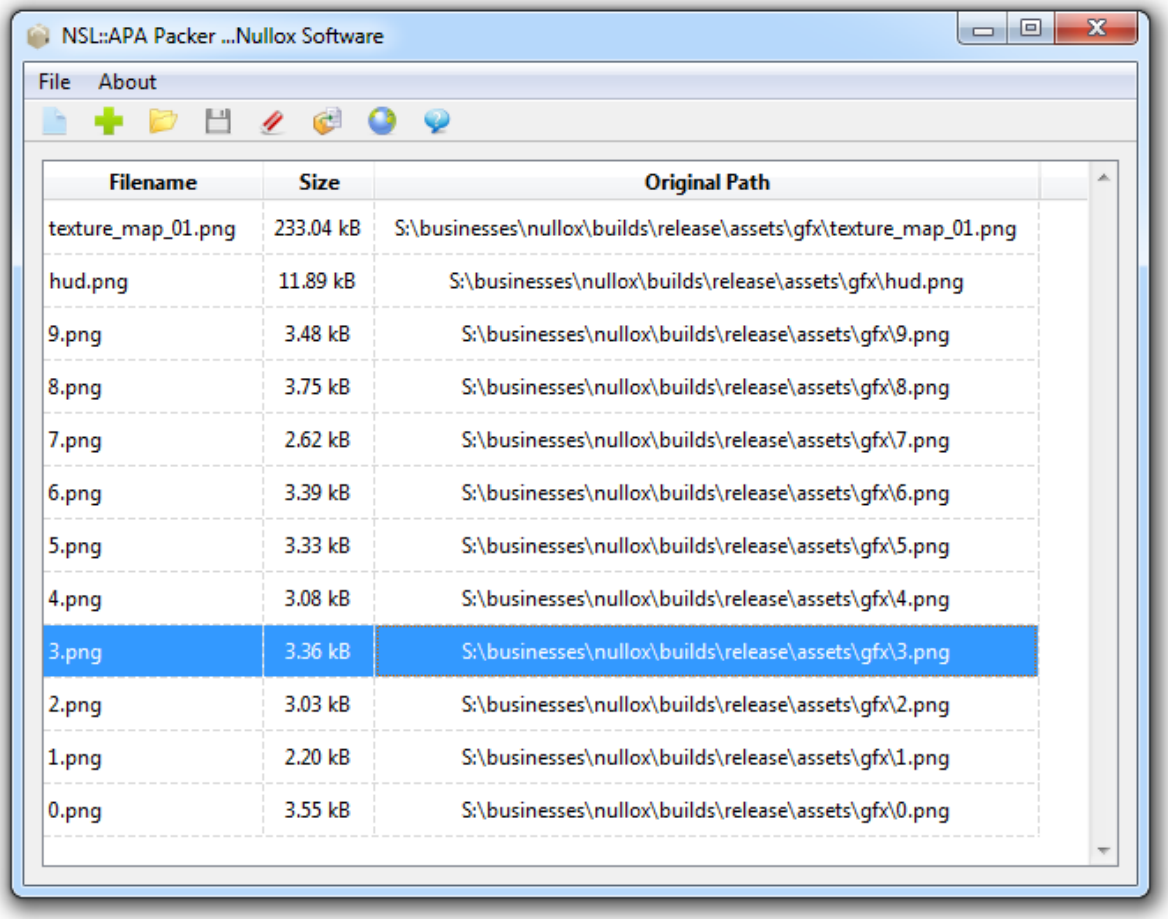
NSL::APA Packer

Asset Package Management Packer Solution

http://www.nullox.com/software/apa-packer/



Written by William Johnson

Enquiries => devteam@nullox.com

Version 1 - 14/06/2015

1. What is it?

APA (asset package archive) packer is an asset manager for all computational projects with distinct needs for media distribution. It can pack assets akin to a stack which omits having to distribute raw media files thus protecting intellectual property. It may be used in many scenarios. Its key objective is to ease the issue of managing assets in game development projects. With access to an API, one can easily access the raw binary to import and manipulate hundreds of assets seamlessly.

With APA packer, you never need distribute multiple RAW assets with your projects. Omit installation overhead by adopting the APA file format. Simply pack, distribute and interface with the C++ API or maybe even develop one of your own using the file format disclosed herein.

1.1. Why?

At Nullox Software, we've been developing software for years. What often happens with a project is that it acquires a large number of assets. Things such as icons, bitmaps, audio and video media each have their own file, size and add needless bulk to installation overheads and distribution effort.

If all these can be packed into a readable format and distributed as a single file and then efficiently unpacked and read at runtime then evidently the benefits become clear.

In addition, issues such as protecting intellectual property by not distributing valuable media in raw format which can be easily stolen is omitted since its packed.

In future research, we plan to embed compression and encryption support directly into NSL::APA thus taking it to the next level of digital package management.

2. File Format

Nullox has a library denoted as NSL, this is the Nullox standard library which is a large library governing all computation that takes place within Nullox Software and Nullox Game developments.

APA is a minute part of NSL thus it is indeed nested with the NSL namespace. The software APA does not discriminate between the potential distinct asset types nested within the APA file format. This provides the benefit of catering to boundless files and asset types in a single package. All file operations use binary mode thus contamination of assets is never an issue.

What follows is a description of the file format thus you gain an understanding of how the packing works and a glimpse of what is possible in future revisions such as compression and encryption initiatives.

2.1 Header

The header within a APA file comprises of six parts, these are:

Signature	Used to identify the file format, ensuring the file contents match the APA input algorithm and to differentiate between distinct file formats which may use the *.apa file extension
Endian Signal	Used to inform the decoder whether the file was encoded using big endian or little endian byte ordering
Version ID	Used to inform the decoder the version_id of the encoder
Asset Count	Informs the input algorithm of the number of assets encoded within the packed archive
Filename Length + Filename	An interleaving collection of filename lengths and filenames corresponding to each encoded asset. These filenames include the original path of the packed file from the distributor.
Asset Sizes	A linear collection of asset sizes

2.1.1 Signature

The signature comprises of six bytes aka magic numbers. The bytes correspond to the decimal vector [78, 83, 76, 65, 80, 65]. These bytes are always present in a legal NSL::APA file.

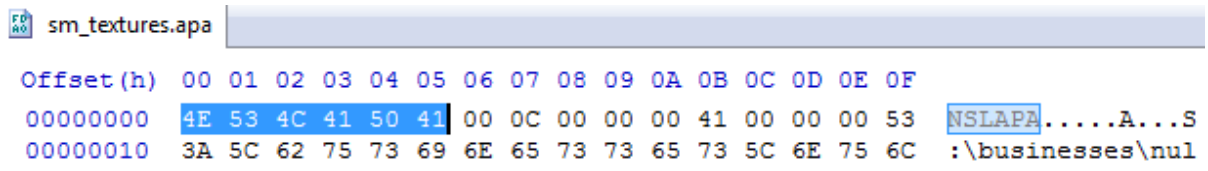


Figure 1, NSLAPA Signature

2.1.2 Endian Byte

The next byte after the signature (the seventh byte) indicates whether the encoding used little or big endian. If the value is equal to zero then the encoding used little endian. For any other value (one in a correct implementation), the encoding would alternatively have been big endian.

2.1.3 Version ID

The next four bytes (a long integer) inform the decoder the version number of the encoder.

2.1.4 Asset Count

The next four bytes (a long integer) indicates the number of assets packed within the archive.

2.1.5 Filename Length and Text

Next follows an arbitrary number of bytes encompassing filename lengths and filename text. The order of each filename .length() and filename .c_str() are equal to the order of the binary dump further on within the archive.

For each filename length and text pair, the length is always four bytes long (a long integer) followed by the textual string corresponding to its original path name. Repeat the extraction process for each asset, since you already know the number of packed assets, the process is feasible without a terminator symbol within the stream.

2.1.6 Asset Sizes

Next follows a linear collection of asset sizes. Each size uses four bytes (a long integer). These asset sizes can be used to offset within the main binary data to differentiate between the number of assets.

2.2 Asset Data

Immediately following the header is the raw binary data for all packed assets, one asset after another. Each asset is ordered thus aligned with the filenames and sizes within the header.