# Native File Operations - Report

Adam Farah Ashton Woollard

April 13, 2017

COMP3000

Table of Contents

# 1 Introduction

## 1.1 Context

To create a modification to the default file system capable of performing simple compressions and file conversion. Converting into formats such '.zip', '.tar.gz' and '.rar'. And conversion of audio, video, and image file types. This project uses compression and conversion algorithms from open source data where available, and open source libraries for proprietary algorithms.

The Native File Operations (NFO) uses an easy to understand layout of file type operations on the right click menu. The file type operations adjust accordingly to the file type of the object that is being selected. This can be seen in Figure 1. NFO was coded in C, using a Virtual-Machine (VM) running Ubuntu 16.04.

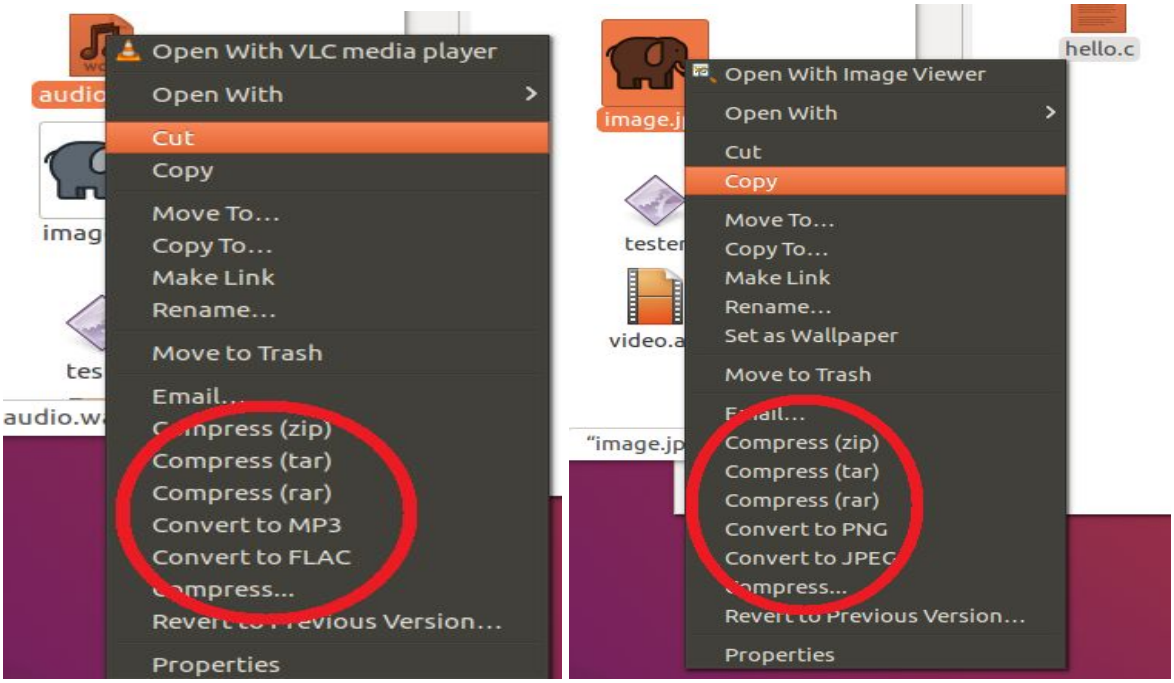


Figure 1: Menu adjusting to file type selection

Figure 1 shows the menu options on selected files adjusting based on the file type of the selected object. Also can be observed is the variety of file operations incorporated into the program.

## 1.2 Problem Statement

One of the challenges faced when incorporating the Native File Operations into the Operating System (OS) was including an acceptable number of files types for all media types, and an acceptable number of compression formats. With too few files types and compression formats, the difficulty of our project may seem too simple and not worthwhile. Ideally, the NFO set of conversion subroutines would incorporate all the most common file types when converting, as well as all the most common

compression formats. For image files, the chosen file types include ‘.png’, ‘.jpg’, and ‘.jpeg’. For video files, the chosen file types include ‘.mp4’ and ‘.flv’. For audio files, the chosen file types include ‘.flv’, ‘.mp3’, and ‘.wav’. For compression formats ‘.zip’, ‘.rar’, and ‘.tar.gz’ were chosen. These files types can be referred to and seen in Table 2 and Table 3. Ideally, more types of files, such as text and pdf files would be included into the program in the future.

Another challenge incorporating the NFO into the OS was understanding and researching the many codecs used for all types of files and compression formats. For image Files, this includes the ‘Portable Network Graphics’ codec for ‘.png’ files, and the ‘JPEG’ codec for ‘.jpg.’ and ‘.jpeg’ files. For video files, this includes the ‘MPEG-4 Part 14’ codec for ‘.mp4’ files and the ‘Flash Video’ codec for ‘.flv’ files. For audio files, this includes the ‘Free Lossless Audio Codec’ for ‘.flac’ files, the ‘LAME’ codec for ‘.mp3’ files, and the ‘Waveform Audio Format’ codec for ‘.wav’ files. For Compression formats, this includes ‘DEFLATE’ algorithm for ‘.zip’ and ‘.tar.gz’ files and the ‘Roshal Archive’ algorithm for ‘.rar’ files. These conversion and compression algorithms can be referred to in Table 2 and Table 3.

Another issue with the implementation of the NFO subroutines was the compartualization of the project. There are five main parts to the system that allow it to operate smoothly. The first set of parts are the conversions and compression subroutines, these are capable of converting image, video, and audio files from one file type to another, as well as being capable of compressing and decompressing files. The second part is the primary logic program, this handle the internal logic of the program, passing files to their correct modules for operation. The final part was the file explorer subroutines, allowing the user to easily make calls to the primary logic program from the right click context menu of an object. Once these core parts of the program were created, the next challenge was incorporating them into each other to create a streamlined program.

## 1.3 Result

Native File Operations in its current state includes conversion subroutines for three different file types (image, video, and audio). Each of these data types includes three file types that the program can convert files between, these can be seen in Table 2. It was felt that this was an acceptable number of file types and conversion formats to successfully demonstrate the capabilities of the program.

NFO currently uses, with the help of certain open-source libraries, all seven of the originally planned conversion codecs, as well as an additional eighth. Refer to Table 2 for these files types and conversion associated conversion algorithms. NFO also incorporates both the planned compression algorithms. Refer to Table 3 for these compression algorithms.

The compartilization and streamlining of the NFO was completed using certain open-source libraries, as well as using best programming practices learned in COMP 3000: Operating Systems and other Computer Science courses. In NFOs current state, when a user right clicks a file and selects one of the new options, the file explorer will send a command to the primary C program, which accepts the

command, parses it and enacts one of the subroutines to complete the command. From there, the program will take the converted or compressed file and save it in the originating folder.

## 1.4 Outline

The other sections in this report will outline crucial information pertaining this program, and will highlight points of evaluations through testing. Section 2: Background Information will detail background information necessary for understanding the operation of NFO. Section 3: Results will describe the work that has been achieved and how a solution was created. Section 4: Quality Assurance and Testing will analyze the results of our testing and highlight the quality of the test cases. Section 5: Conclusion will summarize the information in this report, as well as detail modularbility and future plans for NFO if it were ever commercialized.

# 2 Background Information

To understand how the subroutines work in the Native File Operations program, certain algorithms need to be understood. The following is a brief overview of what each algorithm does without getting too specific into each individual algorithm.

Firstly, a brief description of how compression and conversion algorithms work in general. For this, the ‘LZ77’ algorithm will be used as an example. This algorithm is a fairly basic and straightforward compression algorithm, which the ‘DEFLATE’ algorithm was later based on. This compression algorithm uses lossless data compression, which means that no data is thrown away when the compression occurs. As compression begins, the algorithm will search and read over the file one byte at a time. When a series of bytes appears that the algorithm remembers appearing earlier in the program, the series of bytes will be replaced by a pointer. This pointer has the relative jump back to the location the byte string was first seen, as well as a length value detailing how many bytes are repeated, and the next byte in the string. See Figure 2 for an example of this algorithm.

| Original Sequence: | |
|---|---|
| BCCBCABCCCCABCCBACBA | |
| | |
| Next Sequence | Code |
| B | (0,0,B) |
| C | (0,0,C) |
| CB | (1,1,B) |
| CA | (3,1,A) |
| BCC | (3,2,C) |
| CCA | (2,2,A) |
| BCCBA | (12,4,A) |
| AC | (5,1,C) |
| BA | (3,1,A) |
| | |
| Final Sequence: | |
| BC(1,1)B(3,1)A(3,2)C(2,2)A(12,4)(5,1)C(3,1)A | |

Figure 2: LZ77 Algorithm Example

Figure 2 shows a brief example on how the LZ77 Compression algorithm works on a string of bytes. The next sequence column shows the next string of bytes to be compressed, each string is made up of all repeated bytes except for the last byte. In the code column is the code that will be encoded.The final sequence in this case is longer than the original sequence, however in a longer file with longer strings of repeated bytes, this would not be the case.

As for the compression algorithms, Native File Operations in its current states uses two compression algorithms. For '.zip' and '.tar.gz' files, the 'DEFLATE' algorithm is used. This is a lossless data compression algorithm originating and based on two algorithms that came before, the 'LZ77' algorithm and the 'Huffman Coding' algorithm. In a brief explanation, identical patterns in the file's bit array are removed and replaced with single copies of the pattern and the amount of times the pattern repeats and the location of the repeating pattern. For '.rar' files, the 'Roshal Archive' algorithm is used. This algorithm is a licensed algorithm and as such, can only be created with commercial software and libraries. The software license agreements forbid the reverse engineering of this algorithm. NFO makes use of publicly available libraries in order to be able to include the '.rar' file format, if this software were ever to be become a commercially licensed software, '.rar' files would have to be removed from the available compression formats until a license agreement is secured.

As for image conversion codecs, NFO incorporates two codecs. For '.png' files, the 'Portable Network Graphics' codec is implemented. This codec is a lossless image compression codec. In a brief

overview of how this codec works, an image is encoded by converting the image into a set of data chunks. These chunks are labelled either critical chunks or ancillary chunks. Critical chunks include data such as the first and last data chunk, marking the beginning and end of a file, a list of colours used in the file, and the actual image file, which is further subdivided into smaller chunks. The ancillary chunks can include a lot of information including background colour, metadata, pixel size and more. An image can generally speaking still be viewed even if ancillary data is lost. Conversely, the 'JPEG' codec is used for '.jpg' and '.jpeg' files. As opposed to the previous codec, this is a lossy compression algorithm, which means the algorithm throws away some information when encoding the image in order to decrease the file size. The algorithm works best on images that have smooth colour transitions and few abrupt edges. It works by converting the image into a series of blocks, each of these blocks are given a hue or hue shift, these blocks can then be stitched back together into an image, containing less detail than the original but being of smaller size. Figure 3 demonstrates the compression of an image using the "JPEG" compression algorithm.

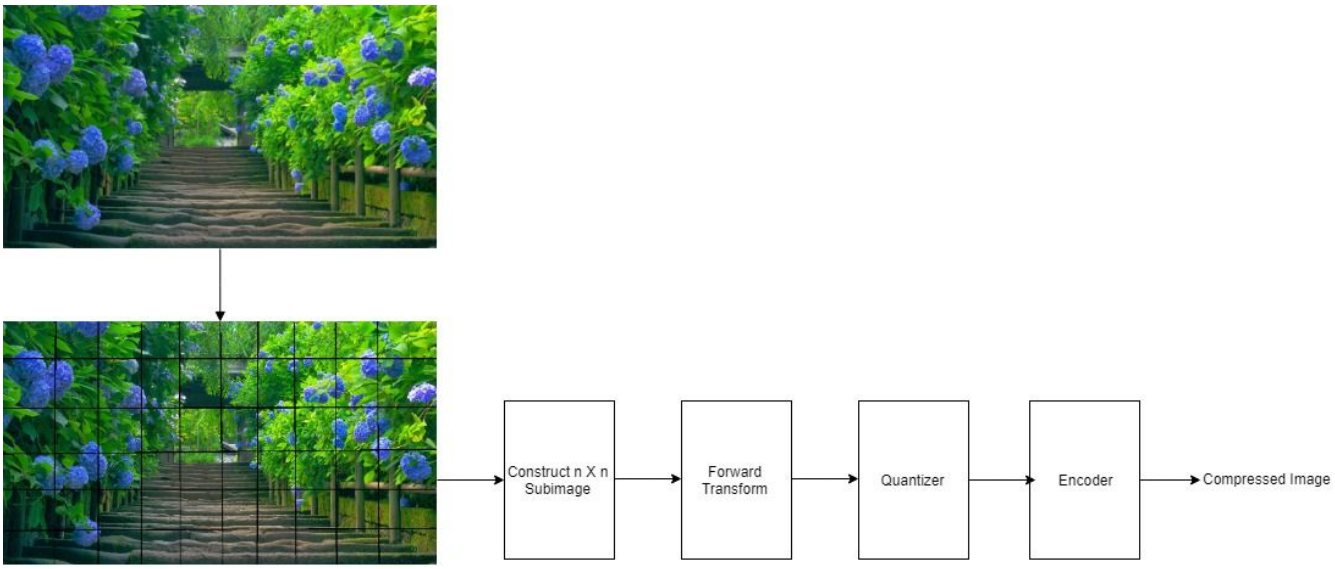


Figure 3: Illustrating Compression of a JPEG image

In figure 3, a quick demonstration of how an image is compressed is shown. This figure is using a compression algorithm such as the 'JPEG' compression algorithm.

As for the audio conversion codecs that NFO makes use of. Two of these codecs include the 'Free Lossless Audio Codec' for '.flac' files and the 'Waveform Audio Format' for '.wav' files, both of which are lossless file types. These two codecs are very much different sides of the same coin. The 'Waveform Audio Format' is the uncompressed audio file, containing all of the original information in its original state. As such, there is no real algorithm to explain with this codec. On the other side is the 'Free Lossless Audio Codec', which is also a lossless audio format. This codec works very similarly to the DEFLATE algorithm talked about earlier, except that it takes advantage of the specific characteristics of audio files in order to compress the file much further than the DEFLATE algorithm is capable of, while still retaining the entirety of the data. As for the final audio codec, 'LAME' is used for

‘.mp3’ files and is a lossy audio format. This algorithm works by making calculations on the audio file, and trying to find which parts of the file are not able to be heard by the human ear, whether the pitch is too high or low for normal hearing, or if the algorithm finds that a louder noise in the file overshadows a quieter noise. This algorithm is also a digital format compared to the the previous two being analog formats. What this means is that the audio is split up into small sections. Each section has only one sound incorporated with it, and for the entirety of the section, only that sound is played. When each section is played back to back, it gives the illusion of a smooth transition through the audio files. In figure 4, an example of this can be seen.

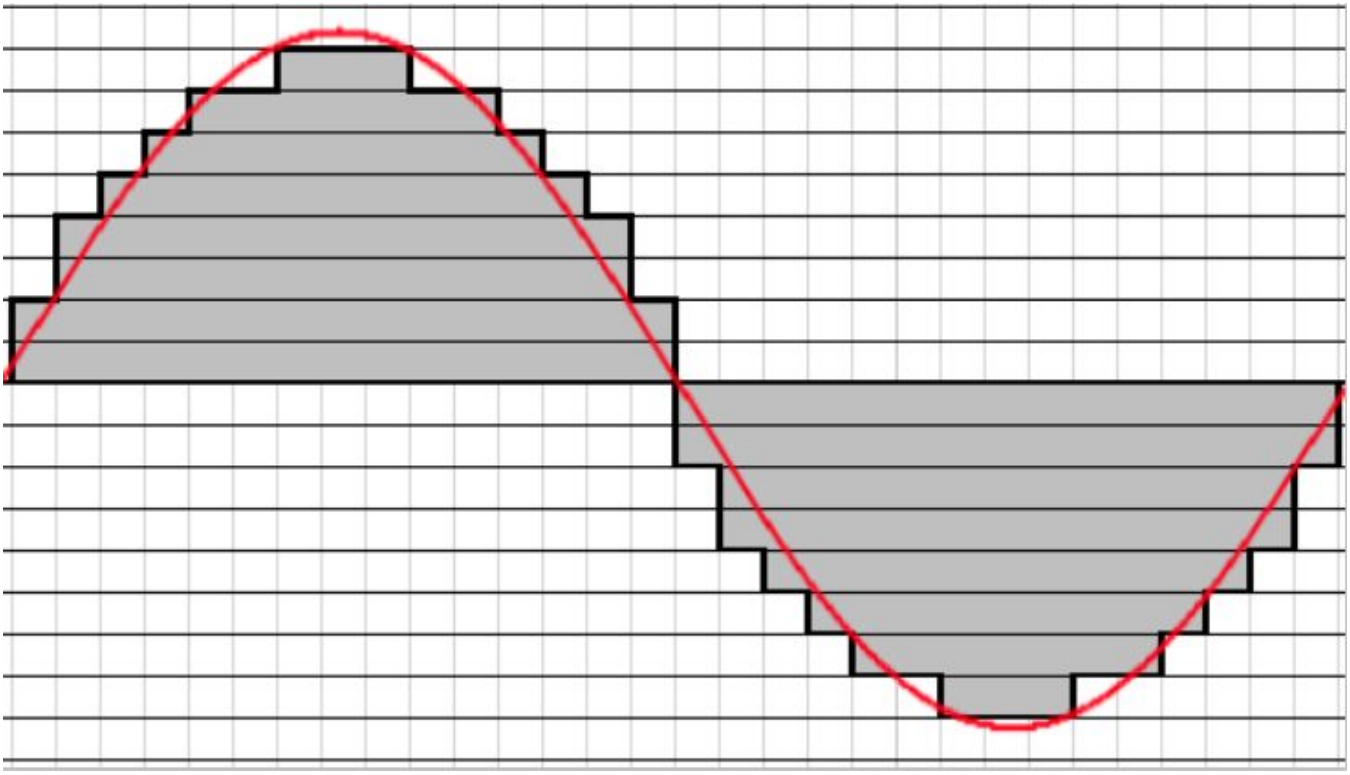

Figure 4: Analog and Digital audio represented on cartesian planes

Figure 4 demonstrates the difference between a lossless audio file (.flac or .wav) and a lossy audio file (.mp3). The sine wave demonstrates the lossless audio file, no transitions are thrown away and the data is read in an analog fashion. The square wave is how a system reads a lossy file, making almost indistinguishable transitions in the playback of the file, but is incapable of playing in a truly analog fashion.

As for video conversion codecs that are implemented in the NFO program, these include three conversion codecs. The first of which is the ‘MPEG-4 Part 14’ codec used for ‘.mp4’ files. This audio codec contains patented technology which requires certain software licenses to be purchased in certain countries. These licenses are largely ignored in this current day and age, and currently, one such lawsuit involving this algorithm is between AT&T and Apple, where the former is trying to sue the latter over the unlicensed use of this algorithm. However, considering this, finding details on how this algorithm works is very difficult. NFO uses a publically available library in order to make use of this codec. Much

like the ‘MPEG-4’ codec, the ‘Flash Video’ codec for ‘.flv’ is also a patented and licensed algorithm, these licenses are again largely ignored by the public and by companies. Information on the operation of this algorithm is hard to find, but in simple terms, it works by using bit streams for video, audio and other information like subtitles. Each of these bit streams carry the data for the section it is responsible for. In other words, a ‘.flv’ file can be distinctly broken up into each frame of the video, as well as the audio of the file. NFO again uses a publically available library in order to perform its conversions with ‘.flv’ files. The final video codec, the ‘Audio Video Interleave’ codec for ‘.avi’ is a lossy open source codec. It works much like the ‘Portable Graphics Network’, dividing the video into chunks, both mandatory and optional. The first chunk contains data such as video height, width and framerate. The next chunks contain the audio and visual data of the file.

# 3 Result

The program, Native File Operations currently has subroutines for multiple conversion and compression algorithms. Refer to Table 2 and Table 3 for a list of these algorithms. 11 of these algorithms were planned from the beginning, and with our available time we were able to implement a 12th conversion codec, the ‘Audio Video Interleave’ codec.

The NFO program uses a compartmentalized approach in order to operate. Three main compartments exist within the program. The file explorer subroutines (handling the integrated nature of the program), the primary logic program (handling the file logic portion of the program), and the conversion/compression modules (handling the file conversions and compressions). Generally speaking when a user wants to call the NFO conversion/compression subroutines on a file, the call is first passed through the file explorer subroutine, then into the primary logic program, which passes it along to the correct conversion/compression module, before the file is passed back to the logic program which saves the file in system memory. This flow of the program can be observed in Figure 5.

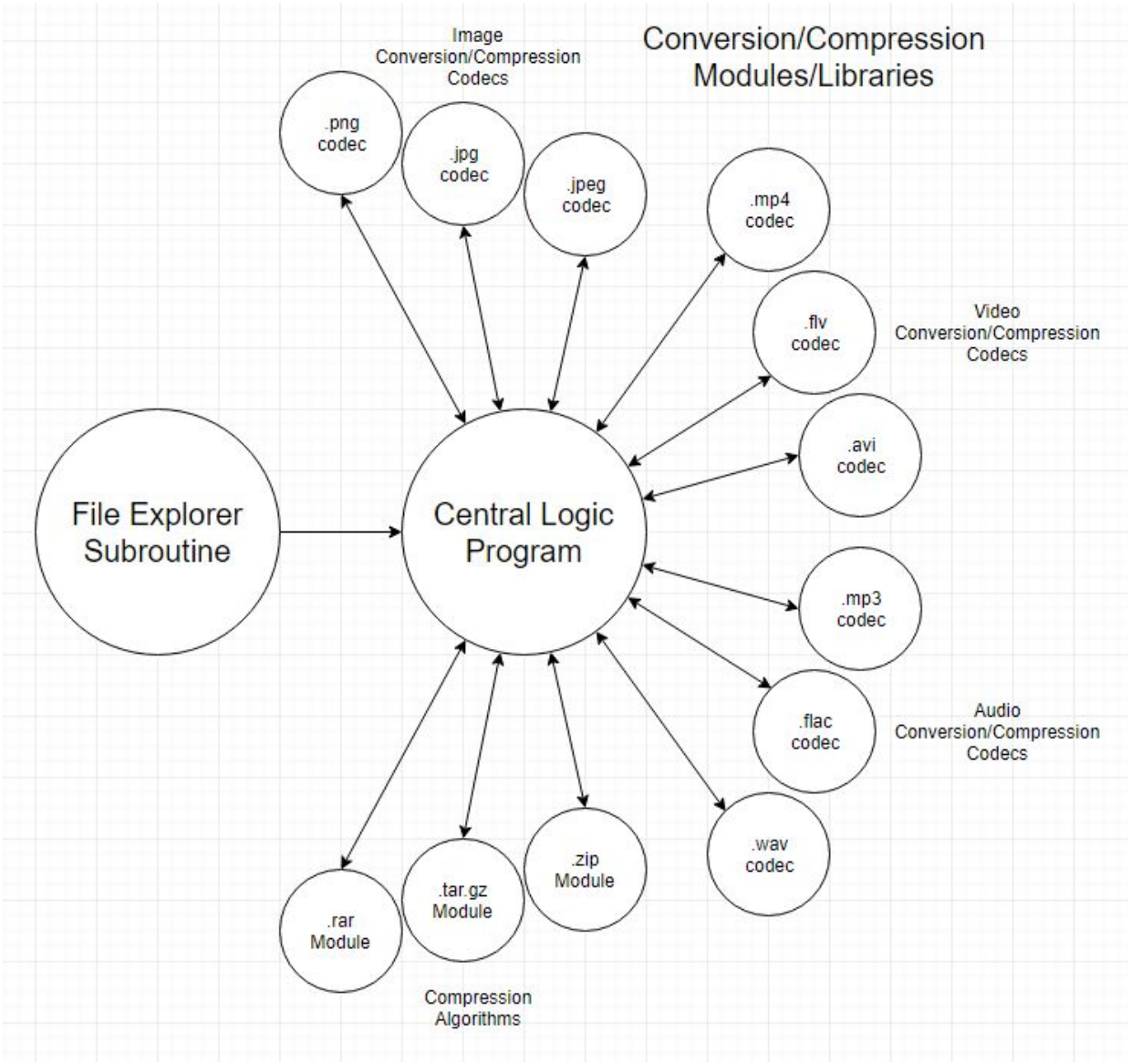


Figure 5: Flow Diagram of Native File Operations

Figure 5 details the flow of the Native Files Operations program. As stated above, it starts in the file explorer subroutines which makes a system call, the primary logic program listens to the call, parses it, and passes the file to the conversion/compression modules, which then perform their operation on the file before passing it back. The primary logic program will then save the file to the system memory.

The first user-program interaction begins with the file explorer subroutines. In the current iteration of NFO, this is a modification to the Nautilus file explorer included in the Linux distro, Ubuntu 16.04. These subroutines add in a variety of dynamic right click options to files. These options will detect the filetype of the selection and change automatically in order to serve only relevant options. In the case of expanding this program to work on other Operating Systems and with other file explorers, these subroutines would have to be modified.

The file explorer subroutines will pass system calls to the main logic program. The logic program will handle these calls, import the selected file, parse and pass along the selected file to the correct modules in order to handle the file conversion or compression. The logic program will then receive the modified file back from the module before saving it to system memory in the originating folder. In the case of using this NFO on an OS without the file explorer subroutines, this central program can be called directly from the command line.

The final part of NFO are the variety of modules. Every conversion codec and compression algorithm is bundled in its own module. Refer to Table 2 and Table 3 for conversions codecs and compression algorithms. The module will accept the file from the primary logic program, convert/compress the file before returning it to the main logic program. Some modules integrated into Native File Operations are open source modules. This is because some conversion/compression algorithms are proprietary and can not be legally replicated by ourselves. These proprietary algorithms include “MPEG-4 Part 14”, “Flash Video”, and the “Roshal Archive”.

# 4 Evaluation and Quality Assurance

A group of 10 participants with varying degrees of technology knowledge were selected in order the test the usability of the Native File Operations versus trying to convert or compress the file without.

First, participants were tested on their ability to perform these conversions and compressions without Native File Operations in 5 minutes or less. Participants were allowed to install software, use command line, and research how to perform the operation online. All participants were able to correctly compress files to a ‘.tar.gz’ in under 1 minute without installing software, without using the command line, and without researching how to. All participants were able to compress files to ‘.zip’ in under 5 minutes without installing software, with using the command line, and with researching how to. For file compression to ‘.rar’, 6 of 10 participants were able to correctly compress the files in under 5 minutes, after installing software, using the command line, and researching how to. For image file conversion, 8 of 10 participants were able to perform the conversion correctly in under 5 minutes without installing software, with using the command line, and with researching how to. For video and audio conversions, no participants were able to perform the conversions correctly in under 5 minutes.

Next, participants were tested on their ability to perform these conversions and compressions with NFO correctly installed. For every compression and conversion, all participants were able to correctly perform the operation in under 1 minute, without installing additional software (past NFO), without using the command line, and without researching how to.

To ensure that files were being correctly converted and compressed, a suite of test cases were created in order to verify a number things. One, the files were being properly converted from one file type to another. Two, the files were not being corrupted in the conversion/compression process. And three, the file size was reduced in cases where a compression algorithm was being applied. Refer to Table 1 for a list of test cases.

| Image Conversion | Audio Conversion | Video Conversion | File Compression | File Size Changes |
|---|---|---|---|---|
| .png to .jpg | .mp3 to .wav | .mp4 to .flv | .zip compression | .wav to .flac |
| .png to .jpeg | .mp3 to .flac | .mp4 to .avi | .rar compression | .wav to .mp3 |
| .jpg to .png | .wav to .mp3 | .flv to .mp4 | .tar.gz compression | .png to .jpg and .jpeg |
| .jpg to .jpeg | .wav to .flac | .flv to .avi | .zip uncompression | .mp4 to .flv and .avi |
| .jpeg to .png | .flac to .mp3 | .avi to .mp4 | .rar uncompression | .zip and .tar.gz compression |
| .jpeg to .jpg | .flac to .wav | .avi to .flv | .tar.gz uncompression | .rar compression |

Table 1: Native File Operations Test Cases

Table 1 lists the variety of test cases performed on files, testing each aspect of the program. Some test cases are combined (eg. in File Size Changes, .png to .jpg and .jpeg), this is because the same algorithm is being used for both files, and therefore only needs to be tested once.

# 5 Conclusion

## 5.1 Summary

Understanding the file types and how compression and conversion algorithms affect them was the main background concepts needed for this. This program delivers ease of access file operations, directly integrated into the right click options on files. This allows the user to not only compress files but also convert files from one file type to another. This element of conversion and compression helps for people who are looking for an easier and integrated way of adjusting multiple file types in environments that are otherwise restricted.

## 5.2 Relevance

The relevant aspects related to this course (COMP3000) dealt with in the NFO program are module creation and installation, as well as OS modifications. The conversion and compression

algorithms use topics discussed in COMP2402. The OS modifications can be shown in the integrated right click menu, see Figure 1 as an example. Modules have been a vital topic of this course and are used in NFO when calling the compression and conversion algorithms on specific files.

## 5.3 Modulability

Native File Operations has been created with the possibility of modulability in mind. The program has been created in three parts, two of which; the primary logic program, and the conversion/compression modules, can be transferred to another OS (Windows or Linux) with only minimal modifications necessary. The third part of the program; the file explorer subroutines, would have to be completely recreated for other Operating Systems and other file explorers.

## 5.4 Future Work

A number of modifications and improvements would be made to the Native File Operations before it was brought to into commercial distribution. A simple modification would be a pop up progress bar during the conversion and compression sequences. Currently, no such dialog is implemented. It was decided during the project that the implementation of such a dialog window would use too much of the development time and was overall, not the point of the program and did not demonstrate what the goal of the program was effectively. That development time was much better suited to creating the core of the program. Additionally, the integration of more conversion and compression types would be an obvious and welcome addition to the Native File Operations Program. Such conversions as image to pdf and text file conversion could be implemented in the future.

Another modification would be the expansion to other Operating Systems. This was clearly not feasible in the timespan of the project, and it was also not the intention of this project. Other Operating Systems this could be brought include versions of Windows, such as Windows 7, Windows 8/8.1, and Windows 10. This could also be brought over to other versions of Linux very easily, versions such as Ubuntu (other versions), RedHat, and Debian. A further stretch would be to incorporate this onto Android devices, where these modules/libraries are currently not well developed or non existent. Operating Systems such as MacOS and IOS would be considered as well, however with Apple's current stance on OS modification software (and lack thereof), this would be prove difficult to achieve.

One feature that may have to be taken out of the program if it was commercially released is the integration of proprietary algorithms. Before a commercial release, a decision would have to be made whether to leave these proprietary algorithms in NFO or not. Many companies ignore these patents and disregard them and use the algorithms as if they were open source, however the fact remains that they are not. The lawsuits surrounding some of these proprietary algorithms are mostly frivolous and are targeted towards larger companies.

The final improvement to NFO that would be ideally implemented before a commercial release is an all inclusive installer for the program. Currently, a user needs to install multiple open source libraries and make the modifications to their file explorer themselves in order to operate the program

successfully. A list of these libraries can be found in the References section. Before a commercial release, an installer would need to be created for NFO to ease the installation process.

# Appendix

## Appendix A: Tables

| **File type** | **Mime type** | **Compression Algorithm** |
|---|---|---|
| .png | Image | Portable Network Graphics |
| .jpg | Image | JPEG |
| .jpeg | Image | JPEG |
| .wav | Audio | Waveform Audio Format |
| .flac | Audio | Free Lossless Audio Codec |
| .mp3 | Audio | LAME |
| .mp4 | Video | MPEG-4 Part 14 |
| .flv | Video | Flash Video |
| .avi | Video | Audio Video Interleave |

Table 2: File types and Conversion Codecs/Compression Algorithms

Table 2 lists the file types that Native File Operations can convert, as well as their mime type and relevant compression algorithm. Files of identical mime types can be converted between.

| Compression Type | Compression Algorithm |
|---|---|
| .zip | DEFLATE |
| .tar.gz | DEFLATE |
| .rar | Roshal Archive |

Table 3: Compression types and Compression Algorithms

Table 3 lists the compression archives that are implemented in Native File Operations, as well as the compression algorithm it incorporates.

# Appendix B: Acronyms

AVI - Audio Video Interleave

DEFLATE - Deflate (Not actually an acronym, but looks like one)

FLAC - Free Lossless Audio Codec

FLV - Flash Video

IOS - Iphone Operating System

JPEG - Joint Photographic Experts Group

LAME - LAME Ain't an MP3 Encoder

MacOS - Mac Operating System

NFO - Native File Operations

OS - Operating System

PNG - Portable Network Graphics

RAR - Roshal Archive

WAV - Waveform Audio Format

## Appendix C: Contributions of Team Members

A. Ashton

-Completed and installed/made general compression algorithms, image conversion modules, audio conversion modules and file explorer subroutines.

-Wrote section 2:Background Information, section 3:Results, section 4:Quality and Test Insurance.

-Created Figure 2: LZ77, Analog and Digital audio represented on cartesian planes, Figure 5: Flow Diagram of Native File Operations, Table 1: Native File Operations Test Cases, Table 2: File types and Conversion Codecs/Compression Algorithms, and Table 3: Compression types and Compression Algorithms

B. Adam

-Completed and installed/made general compression algorithms, video conversion modules, file compression modules and the primary logic program.

-Wrote Section 1: Introduction and Section 5:Conclusion, Appendix

-Created Figure 1: Menu adjusting to file type selection, and Figure 3: Illustrating Compression of a JPEG image

# References

[1] Bradley Sepos, and Scott. “Table of Contents.” HandBrake Documentation - Table of Contents, handbrake.fr/docs/en/1.0.0/table-of-contents.html. Accessed: April 4, 2018.

[2] “Ffmpeg Documentation.” Ffmpeg Documentation, www.ffmpeg.org/ffmpeg.html. Accessed: April 4, 2018.

[3] “GNOME/Nautilus-Actions.” GitHub, github.com/GNOME/nautilus-actions. Accessed: April 4, 2018.

[4] ImageMagick Studio LLC. “Command-Line Processing @ ImageMagick.” ImageMagick, www.imagemagick.org/script/command-line-processing.php. Accessed: April 4, 2018.

[5] “PuppyLinux: UnRAR.” PuppyLinux: UnRAR, puppylinux.org/wikka/UnRAR. Accessed: April 4, 2018.

[6] “WinRAR Archiver, a Powerful Tool to Process RAR and ZIP Files.” WinRAR Archiver, a Powerful Tool to Process RAR and ZIP Files, www.rarlab.com/. Accessed: April 4, 2018.

# Final Word

This project and the course as a whole enlightened us in the practicality and designs of working on low level system work and adding fully integrated features into the Ubuntu OS. With the possibilities ahead for future development this feature can be pivotal in the conveniency of  the clients working on their own projects in an environment that use multiple file types.