UNIVERSITY OF
ALBERTA

# Project Report

**Analysis of Current Portable App Creating Utilities and Fork of the Current Best Open Source Utility**

**Submitted By:**
**Rajendra Ganipineni**

**TABLE OF CONTENTS**

## Table of Contents

## LIST OF TABLES

## List of Figures

## LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| BYOD | Bring Your Own Device |
| CD-ROM | compact disc |
| DLL | Dynamic Link Library |
| EXE | Executable |
| HDD | Hard Disk drive |
| HTML5 | Hyper Text Markup Language 5 |
| IIS | Internet Information Service |
| IT | Information Technology |
| MSI | Micro-Star International |
| MP3 | MPEG-1 Audio Layer III |
| MacOS | Macintosh Operating System |
| PC | Personal Computer |
| PaaS | Platform as a Service |
| RAM | Random Access Memory |
| SCOM | System Center Operations Manager |
| SxS | Side by Side |
| SDK | Software Developer’s Kit |
| UAC | User’s Account Credentials |
| USB | Universal Serial Bus |

## ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Leonard Rogers for the useful comments, remarks and engagement through the learning process of this Project. Furthermore, I would like to thank Leonard Rogers for introducing me to the topic as well for the support on the way. Also, I would like to thank everyone in my survey, who have willingly shared their precious time during the process of interviewing. I would like to thank my student friends, who have supported me throughout the entire process, both by keeping me harmonious and helping me putting the pieces together. I will be grateful forever for your support.

## EXECUTIVE SUMMARY

The main objective of this work is to analyse the current portable app creating utilities and to fork the best currently available open-source portable app creating utilities with some improvements. A portable application is a software product implemented to be portable from one computer network to another. They include portable versions of regular installable applications. There are plenty of portable app creating utilities. The work analysis was focused on the five major currently available portable app creating utilities namely Cameyo, VMware ThinApp, Spoon Studio, Enigma Virtual Box and Evalaze. Among these, the best open source utility is chosen which is Cameyo based on various factors. After forking the best open source portable app creating utility, some recommendations are provided to make improvements in the performance and to meet the future needs.

# 1 INTRODUCTION

A portable application[1] is a software product implemented to be portable from one computer network to another. They include portable versions of regular installable applications. It is also, an application which doesn’t need any installer.

All the sources required to run the app reside in a solitary folder, which can be stored or placed on any disk on the system. If the folder of a portable app is copied or moved to a different location, the application will exhibit the same kind of performance without any change.

Instead of downloading a portable app like a normal app download it as a ZIP file. Extract the portable app ZIP file to a folder and execute the .exe file for the app. Any changes that were made or need to be saved to any configuration settings are done so, inside the same folder.

Some of the examples of portable applications are google chrome, Firefox, apache open office, VLC media player.

The important advantage of using portable apps is from the name we can say it—they are portable. Download them on a USB or flash drive or any portable devices and it can be carried to any place, which means these portable apps are carried from system to system.

Of course, there are some downsides of using portable applications like Windows’ user access control does not work properly for the portable apps when compared to its working nature for installed apps, which means these apps are more subject to non-administrative processes.

## 1.1 Portable Applications

Portable applications[2] offer some distinct focal points when compared with the regular installable applications. They occupy less storage space or will create a smaller number of files, and they enable a user to move between computers while taking the apps and settings.

A portable app is one that doesn't utilize an installer. Every one of the files required to run the app dwell in a solitary folder, which can put anyplace on the system. On the off chance that it changes directory or folder location, the portable app will continue to exhibit the same performance.

Rather than installing a portable app, regularly download it as a ZIP file, extract all data or files related to the portable app into a folder, and execute the .exe file for the app. On the chance that data or configurations of the app are changed, it will allow you to save settings, those settings are saved in files directly inside a that folder.

The most critical[2] benefit of utilizing portable apps is that they are portable. Stick them on a USB drive, for instance, and they can be moved from computer to computer. They would not keep any impression on the PCs which run them on. Everything related to app like any settings or configurations that have been saved, will be placed only in the folder where the .exe file of the portable app is present on the system or on USB drive. It's fundamentally the same as the way things worked during usage of MS-DOS and Windows 3.1.

Portable apps can be useful regardless of whether it is not moving between computers, however. For a certain thing, they leave a little impression on the PC. They will in general be lighter load than most installable apps just by ethicalness of not being introduced. It can match up them (alongside their settings) to the different PCs utilizing something like Dropbox. A user can simply utilize an app once without stressing over it leaving residual files and data on the system.

Without a doubt, there will consistently be apps that are introduced. Either they are simply too enormous or advanced to run as a portable app, or they must exploit Windows multi-user or security capabilities. In any case, numerous apps come in two flavors, which implies it can pick between an installer and a ZIP when they download it.

Obviously, there are a few drawbacks[2] to utilizing portable apps. Windows' User Account Controls (UAC) don't work for portable apps the way they do for installable apps, implying that they are progressively liable to non-regulatory procedures. The drawback is that the IT division and any security conventions they have instituted may be less powerful.

Another drawback of portable apps is that they are not regularly worked with multiple users at the top of the priority list. This is likely not a major issue since they are most likely making a portable drive that they can bear only for it. Yet, if multiple users do need to utilize an app, they will either all need to utilize similar settings, or they must have a few duplicates of the app folder on the portable drive.

Ultimately, if they are running portable apps from a USB drive, they will need to take additional consideration to discharge the drive appropriately rather than simply hauling it out. Else, they can degenerate the apps or cause settings not to be saved appropriately. They can even run into this issue on PCs that don't handle USB drives well when they enter rest or hibernation.

All things considered, the benefits of portable apps for the most part exceed the hindrances especially if they move around to various PCs a great deal.

## 1.2 History of Portable Applications

The concept of portable apps[16] started out of a portable version of Mozilla Firefox in March 2004 by John T. Miller. Soon the open-source group of portable programs outgrew Haller's personal website and he moved it to a community site, PortableApps.com.

## 1.3 Current Portable Application Creating Utilities

Currently there are many application creating utilities[3] that are open and close sourced. Open sourced allows to create app without any security issues. whereas closed ones need credentials to use it. There are plenty of portable app creating utilities among them some of the major open and closed source utilities are[3]:

**Cameyo**: a lightweight and robust portable app creator.

**VMWare Thin App**: A powerful application virtualization software that is perfect for professionals to simplify application deployment and migration process.

**Spoon Virtual Application Studio**: Spoon Studio allows you to change Windows-compatible applications to self-contained virtual applications.

**Enigma Virtual Box**: Another application virtualization system for Windows. Combines all application files and registries into an executable file.

**Evalaze**: is compatible with Windows and Windows Server machines. Creating a portable app is easy by following the instructions given by the Evalaze assistant.

## 1.4 Comparing the Portable App creating utilities and Forking the best open source Portable App creating utility

The Internet reports that the current best open source portable app creator is Cameyo with the success rate equal to 85% and the best closed source portable app utility is VMWare thin apps with a success rate equal to 95%[7]. The apps created by using this Utilities will be compared with the apps created by the other Utilities to determine if this reporting is accurate.

## 1.5 Improvements to the best open source Portable App creating utility

The recommendations for the Cameyo open source utility are that it lacks an app menu that can be used to list and search for installed or portable apps already stored, so that can be recommended for the improvement of the application.

## 1.6 Outline

The material in this report is organized into 5 sections dealing with the following topics:

Section 2: Explains the analysis of portable application and portable application creating utilities.

Section 3: Explains the comparison of best open and closed source portable application creating utilities.

Section 4: Explains the forking of best open source portable application creating utility with some recommendations for improvements.

Section 5: Explains the conclusion of the complete analysis. The chapter also provides some recommendations and suggestions for any future work.

Section 6: In this section we will be discussing about the source code organization and flow.

# 2 ANALYSIS OF PORTABLE APPLICATION AND PORTABLE APPLICATION CREATING UTILITIES

## 2.1 Explanation on Portable Application

Portable apps[4] are applications that can be executed from a pen drive without installing them on the systems. What's incredible about this is that they don't have to introduce these apps on the PC.

Engineers make these apps with the end goal of a simple exchange starting with one computer then onto the next. Programs can likewise have a portable form that they can, without much effort, add to the computer apps.

These apps are open source and can be gift supported, minimal effort, or even totally free. They can regularly store these apps instantly on a USB drive, however they can likewise keep them in other media, for example, memory cards, memory sticks, and Smart Media. they may likewise store these apps on web utilizing Google Drive or Dropbox.

**Open Source Definition**[4]: Open source is any computer program where developers give a source code that is available for modification or use as users see fit. Developers make this program as an open coordinated effort and with the expectation of complimentary use.

## 2.2 Benefits of Portable Applications

There are various benefits of utilizing or switching to portable apps as follow[4].

- They are easy to convey and go with and are prepared to utilize whenever and anyplace.
- They occupy zero storage room on the computer.
- Portable renditions of regularly utilized apps are accessible online for nothing.
- It can store information in one area, so it's generally realized where to discover them.
- All that is to do is have to run them is now in the PC – simply click.
- These apps can erase transitory files consequently upon exit.
- The apps keep the entirety of their settings inside them. the custom settings will even now be there when it runs them on another computer.
- They don't introduce symbols or alternate routes that mess with or hinder the computer.

## 2.3 Cons of Portable Applications

Despite having their focal points, portable apps have various drawbacks[4] as well.

- They have limited storage capacity controlled by the size of the USB drive, which may not be advantageous if a portable app needs a great deal of room to save its settings.

- Portable apps are sensitive to voltage, where an unexpected or extreme voltage increment can wreck the app information. It should make backups to keep this from happening.
- In uncommon cases, some portable apps don't have every one of the elements of their work area rendition, which diminishes its exhibition.
- The drawback of pulling out an apps drive is that it can corrupt the drive. When the USB drive is lost, likewise so are the apps.

## 2.4 Portable App Creating Utilities

Portable app makers[11] typically utilize application virtualization innovation to make a portable adaptation of a product. A portable application doesn't require to be installed and can be brought anywhere. It scans the system when installing an application and investigates the changes. At that point, it changes over the introduced files, DLLs, and vaults into a solitary executable (.EXE) file.

### 2.4.1 Cameyo:

Cameyo[5] is a prominent application virtualization item. It enables to make portable application holders of the product. It catches programming establishment, and then transforms it into a standalone executable that contains the whole vault and file system of that product.

When they have that standalone EXE holder with the whole programming inside, they can transfer it to Cameyo cloud server and then play it at whatever point they like through any HTML5 program or utilizing their committed Android and Windows customers. They can even interface with Dropbox or Google Drive and work with the product legitimately on the files.

Cameyo is exceptionally incredible and has loads of cutting-edge commands for control users. However, it's very simple to begin with, and even enables to make virtual bundles of internet, utilizing only the program.

Cameyo[5] is virtualized into a solitary EXE file, permitting duplicate and execute on the computer without installing the application. They are additionally ready to distribute the own product to use cloud server and run it from any gadget with a HTML5 program without any modules. This capacity gives it the ability to demo the product to different users through the website.

It works by first taking a depiction of the system before installing an application and then another a while later. The program at that point thinks about the progressions to understand what changes are expected to the system. The Cameyo window will be as shown in Fig.2.1.

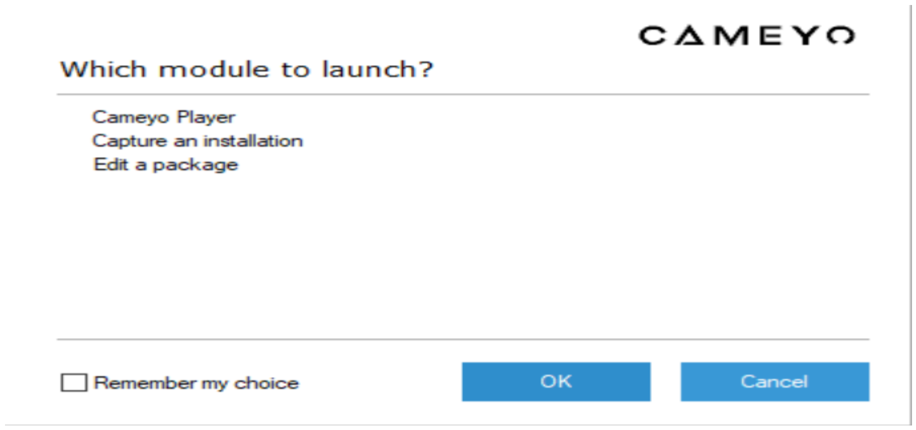


Figure 1: Cameyo Window

The Highlights that set Cameyo apart are many. Comparable utilities available don't have comparable highlights from this item, including information encryption, cloud bundling, and virtualization modes for both RAM and DISK. The application allows you to apply user limitations, for example, scripting and passwords. It can likewise save information and settings on a USB or Dropbox. At the equivalent, it is a reasonable option to VMware ThinApp and comparable projects.

### 2.4.2 VMware ThinApp:

VMware ThinApp[6] is an application virtualization device designed for the VMware Inc. Frame of Reference Suite and accessible as a standalone item. Work area administrators use ThinApp to port heritage apps onto new working systems, empower apps on high-security work areas and to streamline application refreshes and other administration tasks. The app virtualization device is specialist less, with nothing introduced on the user’s machine.

VMware ThinApp packages applications into executable files, MSI or .EXE group, which are exemplified from different apps and from the basic machines working system. ThinApp packages made by one variant of the device exist together on a similar user gadget with applications packaged by another adaptation. Application virtualization additionally enables administrators to entitle various users to various apps, which would then be able to be conveyed starting with one work area then onto the next by means of USB drives.

VMware ThinApp makes application holders by utilizing a form procedure to bundle application records and vault into a solitary packed executable document which can be run on an assortment of working frameworks without establishment. The application holder uses square based spilling with straightforward decompression into memory to execute all application capacities.

Applications can be executed from a client's work area, a network share, or removable storage like a USB streak drive or CD ROM. Applications run completely in client mode and are unmistakable to the working framework, as standard windows procedures to keep up the suitable security setting.

VMware ThinApp presents working framework assets and capacities to the virtualized application, permitting full usefulness and a consistent client encounter while yet encapsulating the application's records, library sections, COM/ActiveX controls, and administrations in a portable holder for use over various working frameworks. VMware ThinApp will be launched in windows as shown in Fig.2.2.

Welcome to Setup Capture

You will be guided through the process of creating a virtualized application.

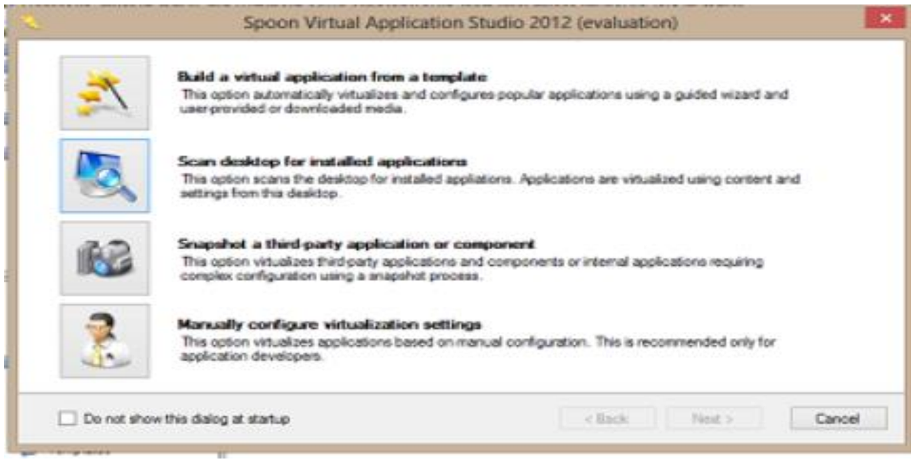
Figure 2: VM Ware Window

VMware ThinApps technology gives numerous benefits to end clients and the IT organization. The following list abridges essential benefits acknowledged by the clients[8]:

- Avoids Application Conflicts
- Minimizes Cost of Packaging and Deploying Applications
- Application Delivery without Agent.
- A Safe and Supportable Delivery Model for Applications
- Less Cost for Implementation and Ownership

### 2.4.3 Spoon Studio

Spoon Virtual Application[9] Studio converts Windows-based applications into single folder virtual applications. Application virtualization devices from Spoon bundles software applications into single executable documents that can be run on any PC utilizing the Windows platform, empowering applications to be run with no introductions, conditions, or clashes. Spoon Virtual Application software window is as shown in Fig.2.3.



Spoon Virtual Application Studio 2012 (evaluation)

**Build a virtual application from a template**
This option automatically virtualizes and configures popular applications using a guided wizard and user-provided or downloaded media.

**Scan desktop for installed applications**
This option scans the desktop for installed appliations. Applications are virtualized using content and settings from this desktop.

**Snapshot a third-party application or component**
This option virtualizes third-party applications and components or internal applications requiring complex configuration using a snapshot process.

**Manually configure virtualization settings**
This option virtualizes applications based on manual configuration. This is recommended only for application developers.

☐ Do not show this dialog at startup

< Back | Next > | Cancel

Figure 3: Spoon Studio Window [11]

The Key Features of Spoon studio are as follows[9]:

- Business Continuity: Runs inherit applications appropriately on Windows 7, 8 and 10 platforms, guaranteeing business progression.
- Compromise: Possesses multi-platform similarity and automatically settle inconsistencies between different working frameworks.
- Similarity: "AppVShop can virtualize 32-piece and 64-piece applications, IIS, SCOM and SxS services, and databases such as SQL server."
- Convey Anywhere: Able to send in EXE's, MSI's, intranets with Spoon Server, or on the web with spoon.net.
- Bother Free: No reboots, authoritative benefits or separate arrangement steps required with Spoon Studio Virtualization.

### 2.4.4 Enigma Virtual Box

Enigma Virtual Box[10] is utilized for file and registry frameworks virtualization and enables to solidify all files and registry setting utilized by the application into a single executable file, which means there is no need to extract these files or application related files to hard disk.

"With Enigma Virtual Box[10], any types of files, dynamic libraries (*.dll), ActiveX/COM objects (*.dll, *.ocx), video and music files (*.avi, *.mp3), text files (*.txt, *.doc) can be virtualized. [10]" Enigma Virtual Box[10] does not place interim files in the hard disk drive, file copying is performed in the active process memory only. The Enigma virtual box can be run in different MS Windows versions like 2000/XP/2003/Vista/2008/Seven/2012/8/8.1 and Windows 10.

Enigma Virtual Box is ideal for creating portable applications. Enigma Virtual Box includes a proficient loader into the main module of application which executes before the main code of the application runs. The loader intercepts framework calls to the disk for file read/compose, and if the objective file is virtualized, the Enigma Virtual Box will imitate the file in memory and return the necessary outcome. Enigma Virtual Box Application window is as shown in Fig.2.4.

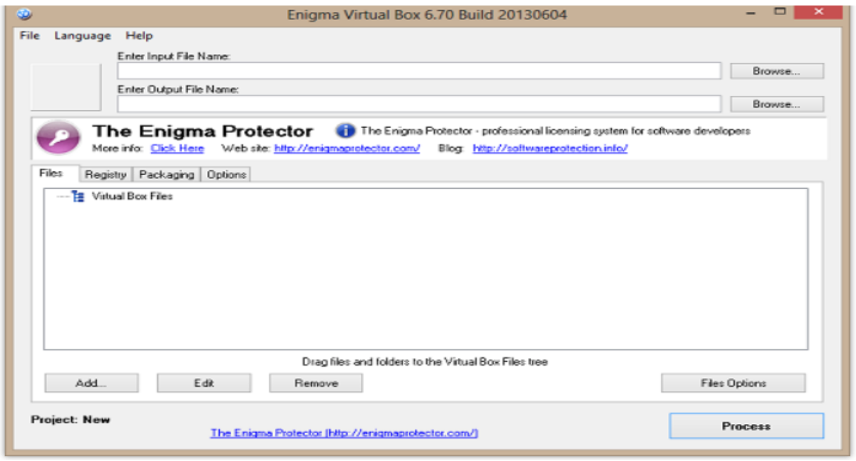

Enigma Virtual Box 6.70 Build 20130604
File
Language
Help
Enter Input File Name:
Browse...
Enter Output File Name:
Browse...
The Enigma Protector
The Enigma Protector - professional licensing system for software developers
More info: Click Here
Web site: http://enigmaprotector.com/
Blog: http://softwareprotection.info/
Files
Registry
Packaging
Options
Virtual Box Files
Drag files and folders to the Virtual Box Files tree
Add...
Edit
Remove
Files Options
Project: New
The Enigma Protector (http://enigmaprotector.com/)
Process

Figure 4: Enigma Window [11]

Main Benefits of Enigma Virtual Box are as follows[10]:

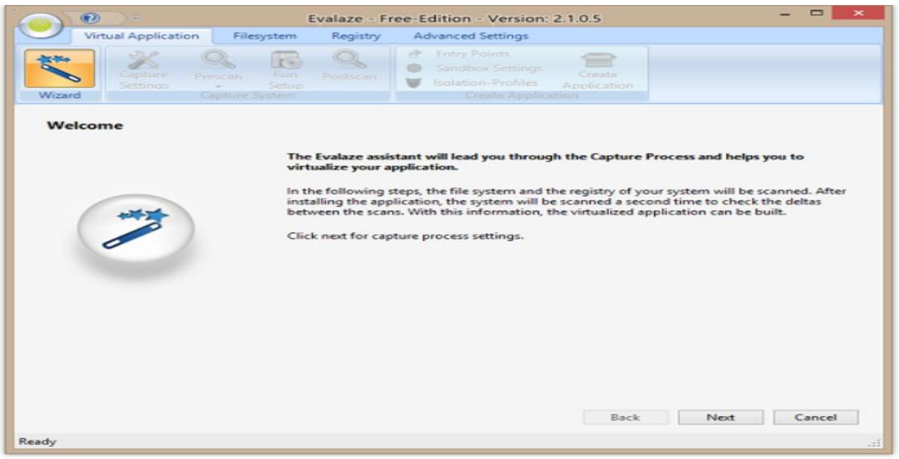
- Enigma Virtual Box does not store the data created by this portable app to the hard disk and does not make any interim files on the client's system
- Enigma Virtual Box supports x86 (32-piece) and x64 (64-piece) binaries that sudden spike in demand for all versions of Windows NT
- Enigma Virtual Box is a completely free application; don't need to pay anything to utilize it!
- Enigma Virtual Box supports any sort of files that needs to be virtualized
- Enigma Virtual Box permits compressing virtual files, in this manner to reduce the total size of an application a few times
- Enigma Virtual Box permits to register ActiveX/COM components without administrator access.

Enigma Virtual Box[10] does not give adequate assurance to virtualized files. To ensure the application files, Enigma Virtual Box is recommended. Enigma Virtual Box is very reliable and simple to use since does not require any additional technical knowledge or modification of the source code of the main executable source file. Simply run the Enigma Virtual Box, indicate the main executable file of the application, include files or directories it uses to the list, and snap Process.

Enigma Virtual Box is a completely free application that does not require any registration or installment. This is the sole freeware item with such usefulness.

### 2.4.5 Evalaze

Evalaze[11] is most easily compatible with Windows and Windows Server machines. The process of creating a portable app is easy because it can be done by following the instructions given by the Evalaze assistant. Before the installation of the application, Evalaze will run a pre-scan to scan registry and file system. Once the installation of application is over, a post-scan is performed to identify the changes on the computer and to create a portable version of the application of the installed application. Evalaze Application window is as shown in Fig.2.5.



Welcome

**The Evalaze assistant will lead you through the Capture Process and helps you to virtualize your application.**

In the following steps, the file system and the registry of your system will be scanned. After installing the application, the system will be scanned a second time to check the deltas between the scans. With this information, the virtualized application can be built.

Click next for capture process settings.

Figure 5: Evalaze Window [11]

Evalaze is quite easy to use so that a person having limited technical knowledge can easily create their own virtual applications and a free version is also available.

### 2.4.6 Portable App Suite

It is a location on the web where the collection of portable versions of installable applications like web browsers, gaming applications, audio players, VLC media players, email apps, e library apps, document editor applications. All these applications are configured in way that they work in the portable way. All we have to do is just go to that portable suite and search for the required portable applications just download them on to the USB or system and use it wherever you want.[18]

There are many portable application suits currently available, they are

1. Portable apps.com: The portable apps.com is a place where we can find thousands of portable applications for the normal installable applications. It contains major applications Microsoft office, cricket games, password manager, email applications, messaging apps etc.
2. LiberKey: It's a portable app suite which currently contains 289 applications which can be readily downloadable. It contains applications related to video, audio, gaming and messaging applications.[18]

3. Geek.Menu: It is a fork of the PortableApps.com menu system. It has many enhanced features such as support for true crypt, categories, custom buttons, multi wallpaper switching, auto-executing apps on start, ejection scripts, special behavior for authorized PCs, local/www search bar, etc.[18]
4. Pstart: It is portable app suite where it mainly contains the portable versions of all web browsers available.
5. Lupo Pen Suite: It’s a portable application suite where it contains portable versions of 180 programs and games. It can be available around 28 languages.
6. Code Safe: It is another software tool that turns any portable drive from a simple data carrier to a computer-on-stick. Carry your computer programs with you, manage them, launch them on any PC, and leave no footprint behind
7. Win wen Pack: It is another portable application suite which will contain portable versions of many applications

# 3 COMPARISON OF BEST OPEN AND CLOSE SOURCE PORTABLE APP CREATING UTILITIES

This chapter compares the major open and close sourced portable App creating utilities. The advantages of using each utility is discussed and compared to obtain the best open sourced portable App creating utility.

## 3.1 COMPARISON OF PORTABLE APP CREATING UTILITIES

The portable app creating utilities should have certain properties to stand alone in the crowd. Those features must make the process easier, faster and simpler. Each utility will have both pros and cons. The utility which is more powerful, and beneficiary wins the race. The comparison of these utilities was based on its ease, user friendly, robust, fast and effectively portable in any device.

Cameyo[17] is a cost efficient and reliable Platform as a service that helps companies to move their Windows apps to the cloud environment. Cameyo provides a premium on flexibility, because of which it can executed in multiple configurations: in a public cloud which is less secure or private cloud, in a hybrid cloud environment or on-premises environment. Thus, it can maximize the already present tools and avoid increase of cost for upgrades.

The benefit of Spoon Studio[12] is the reality that it as of now several applications accessible for use. So, the user is not simply paying for an item when buying or using Turbo-net but getting a service. VMWare’s ThinApp must be the best and generally utilized arrangement which makes applications as such. Spoon has given that other people who utilize this technique don't, has the capacity to change over ThinApp applications to the Spoon format.

Turbo.net gives considerably more capabilities than ThinApp while giving different benefits. The current ThinApp clients might be the ideal market to pitch this item to. Spoon offers the plausibility of utilizing portable applications and applications distributed via the Turbo.net webpage or own private website. It also offers the probability of utilizing their spilling profile capabilities to stream applications which would take out the requirement for introducing virtual application via MSI, which is constantly is the genuine benefit of app virtualization.

The real fact of the portable app utilities is that many of the application virtualization or portable app creating utilities like Cameyo, Spoon, Evalaze[13] and ThinApp are having similar features when compared to one another, in the processing, ease of use, benefits and for the most part limitations.

An interesting result is that Evalaze will support virtualizing 64-bit applications which VMWare ThinApp cannot support despite its technical developments. Although VMWare is continuously trying to release upgrades of ThinApps but with feasible and cheaper variants of portable app utilities out there like Evalaze and Cameyo they will need to do different things to set new standards in portable software, because Evalaze certainly is competition. “Cameyo[13] is a low-cost solution but perhaps for a business Evalaze would be the winner in this fight as the vendor provides support”[13] .

Finally, ThinApp will be the general virtualization solution for a small to medium sized company. But Cameyo excels with higher end benefits among the open source portable app creating utilities as it provides cloud services and mobile app services which are secure and user friendly.

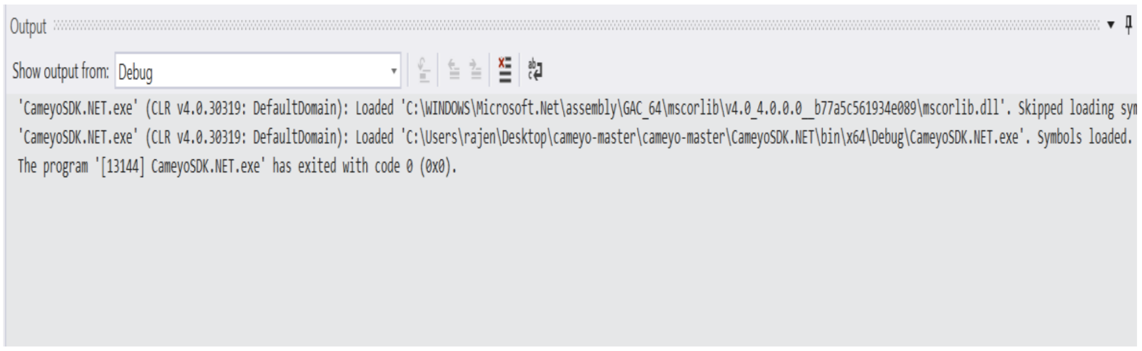
## 3.2 Executing different portable Software utilities:

In this section we will be discussing downloading and executing the different portable app tools like Cameyo, Enigma virtual box, Evalaze and spoon studio.

The main purpose of downloading and executing them was to find out size of software, amount time needed by tool to create portable version of an app and number of files it was creating while creating portable version of application. By assessing all these conditions and we also considering the aspects like number of downloads of tool, rating for each tool to compare the best open source portable app tool.

### 3.2.1 Cameyo Execution:

1. Cameyo.net file executed in visual studio showing cameyo.exe created in the Cameyo master folder: Here we will be executing the Cameyo open source code that is cameyosdk.net file in visual studio so that it will create cameyo.exe file the Cameyo master folder the running system.

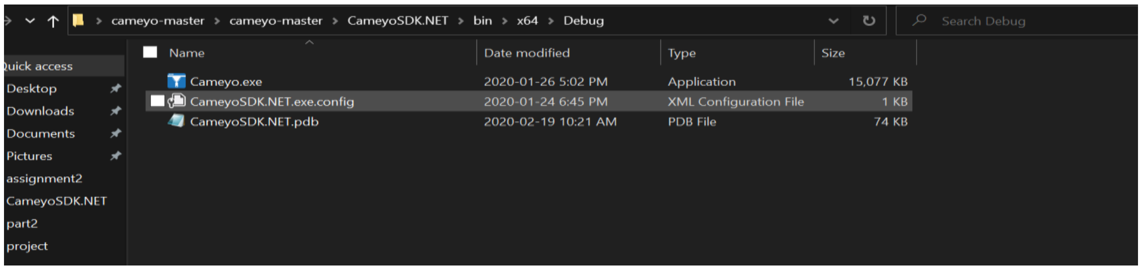


```
'CameyoSDK.NET.exe' (CLR v4.0.30319: DefaultDomain): Loaded 'C:\WINDOWS\Microsoft.Net\assembly\GAC_64\mscorlib\v4.0_4.0.0.0__b77a5c561934e089\mscorlib.dll'. Skipped loading sym
'CameyoSDK.NET.exe' (CLR v4.0.30319: DefaultDomain): Loaded 'C:\Users\rajen\Desktop\cameyo-master\cameyo-master\CameyoSDK.NET\bin\x64\Debug\CameyoSDK.NET.exe'. Symbols loaded.
The program '[13144] CameyoSDK.NET.exe' has exited with code 0 (0x0).
```

Figure 6: Cameyo.net executed

From the above picture we can see that Cameyo.exe was created in the path specified in the above screen shot.

2. Cameyo-master\cameyosdk.net\bin\x64\debug\cameyosdknet.exe: Here the below screen shot is showing after executing Cameyo sdk.net the Cameyo.exe file was created in the path specified above. This Cameyo.exe is an application use to run the Cameyo tool.



cameyo-master > cameyo-master > CameyoSDK.NET > bin > x64 > Debug

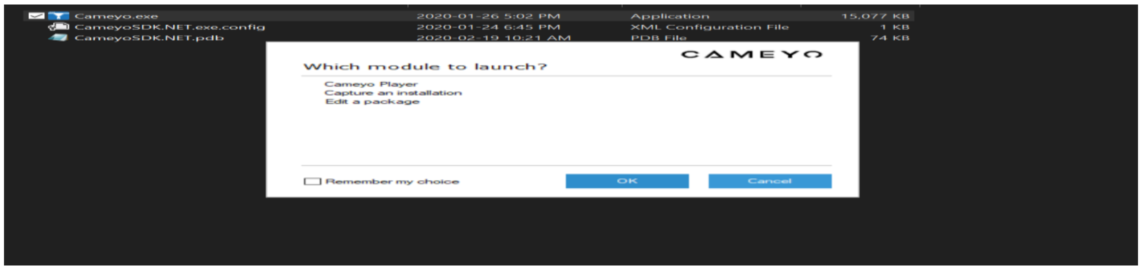
Search Debug

Quick access
Desktop
Downloads
Documents
Pictures
assignment2
CameyoSDK.NET
part2
project

| Name | Date modified | Type | Size |
|---|---|---|---|
| Cameyo.exe | 2020-01-26 5:02 PM | Application | 15,077 KB |
| CameyoSDK.NET.exe.config | 2020-01-24 6:45 PM | XML Configuration File | 1 KB |
| CameyoSDK.NET.pdb | 2020-02-19 10:21 AM | PDB File | 74 KB |

Figure 7: Cameyo.exe on system

3. Executing cameyo.exe file: Once the Cameyo.exe file was created we will be executing it. Once we click on the Cameyo.exe file it will show a pop-up window with the options like Cameyo player and capture the installation. We will use the option capture the installation to perform pre scan and post scan to find out changes made in the system, which will be help full in converting normal app to portable app.



| Name | Date modified | Type | Size |
|---|---|---|---|
| Cameyo.exe | 2020-01-26 5:02 PM | Application | 15,077 KB |
| CameyoSDK.NET.exe.config | 2020-01-24 6:45 PM | XML Configuration File | 1 KB |
| CameyoSDK.NET.pdb | 2020-02-19 10:21 AM | PDB File | 74 KB |

CAMEYO

Which module to launch?

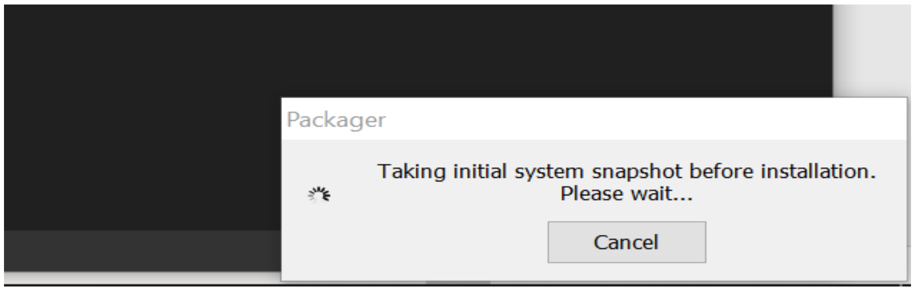
Cameyo Player
Capture an installation
Edit a package

Remember my choice

OK Cancel

Figure 8: Executing Cameyo.exe

4. Taking initial system snapshot: Once we click on the capture installation the Cameyo will start executing and we start making initial snapshots of system. It will take some time depending on the performance of system. To complete this task quickly I will suggest stopping using any other processors in the system which will slow up the process of initial snap shoting of system.



Packager

Taking initial system snapshot before installation.
Please wait...

Cancel

Figure 9: Initial snapshot

5. Here I am installing fire fox application to convert it to portable app: After the pre or initial snapshot of system was done we will be installing the fire fox or some other installable application to our system. So that when Cameyo is performing the final snapshot it will

observe the changes made to system and will show the changes or any added apps to the system in a pop-up window.

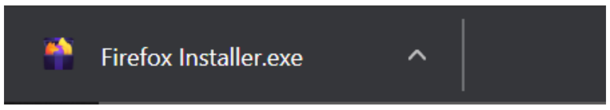
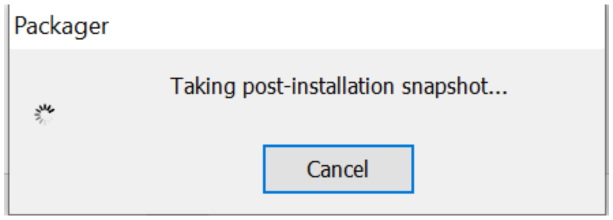


Figure 10: Firefox installer

6. Taking post installation snapshot of system after installing Firefox application: Once the fire fox or any other app that we wanted to convert into portable is installed into our system we press start post installation snapshot of system. Then Cameyo will start taking the post snapshot of system to find any changes made to the system. If it finds out any changes or any newly added apps to the system, it will show in separated window where we can see fire fox application as it is newly added. Once we click on fire-fox and choose option to convert it to portable the we can see portable version of fire-fox in Cameyo apps folder.



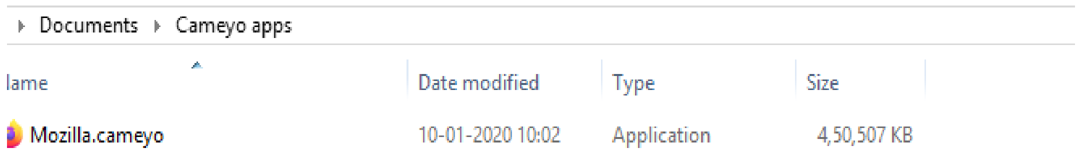
Packager

Taking post-installation snapshot...

Figure 11: Post Installation snapshot

7. Now Firefox is converted to portable application and it creates only one (1) file. Here after conversion we will get successful built messages as shown below:

   Here is the final output showing Firefox is converted to portable app:



Documents ▸ Cameyo apps

| lame | Date modified | Type | Size |
|---|---|---|---|
| Mozilla.cameyo | 10-01-2020 10:02 | Application | 4,50,507 KB |

Figure 12: Firefox converted to portable

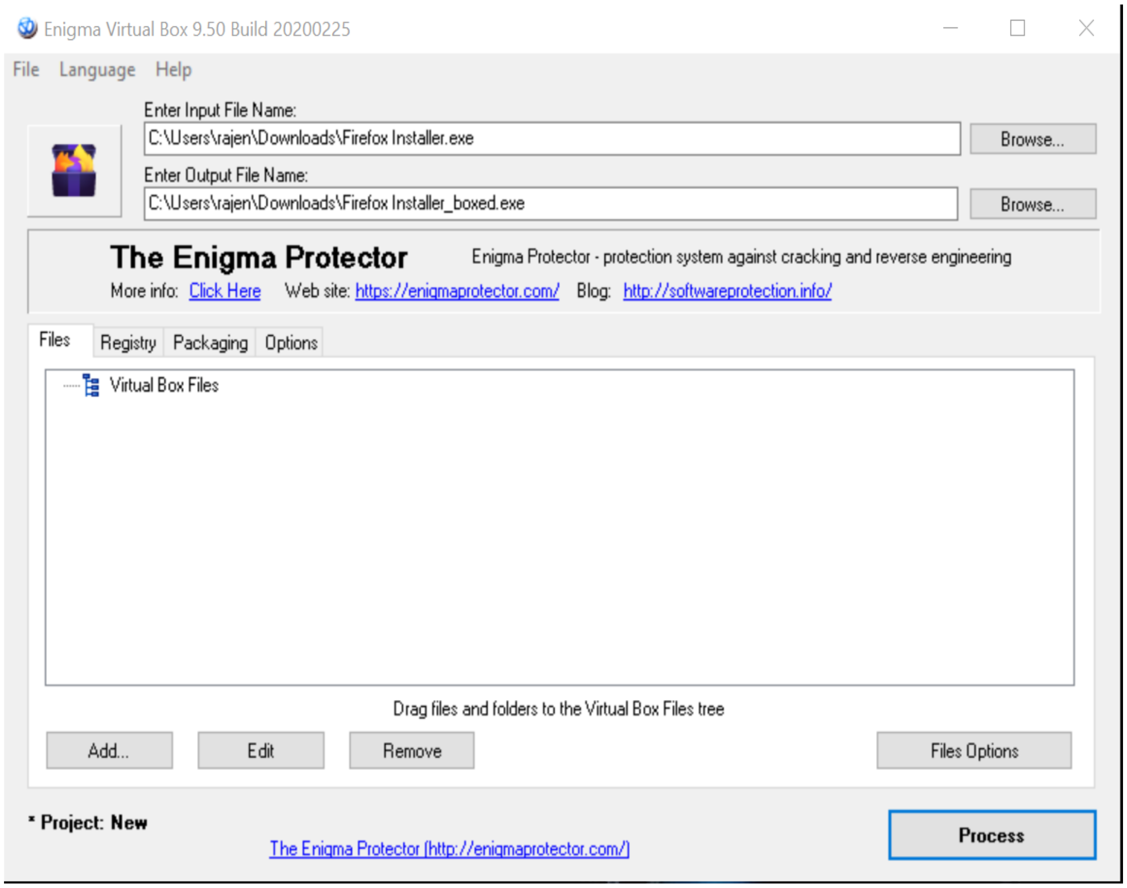
So, from the above screen shot we can see that how the installed application is converted to portable application.

While performing all this execution I have noted the time required to convert normal app to portable app which is around 4.2 minutes. The file size of Cameyo is around 14.7Mb and it will be creating on 1 app in the Cameyo apps folder.

### 3.2.2 Enigma virtual box execution:

We can download and install the Enigma virtual box tool from the internet. After installing the tool how it can be used is explained in the steps below.

1. Open the Enigma virtual box app which can set on desktop as desktop icon. As soon as opening the app we can see a window with different options as shown below



Enter Input File Name: C:\Users\rajen\Downloads\Firefox Installer.exe

Enter Output File Name: C:\Users\rajen\Downloads\Firefox Installer_boxed.exe

The Enigma Protector Enigma Protector - protection system against cracking and reverse engineering

More info: Click Here Web site: https://enigmaprotector.com/ Blog: http://softwareprotection.info/

Drag files and folders to the Virtual Box Files tree

* Project: New

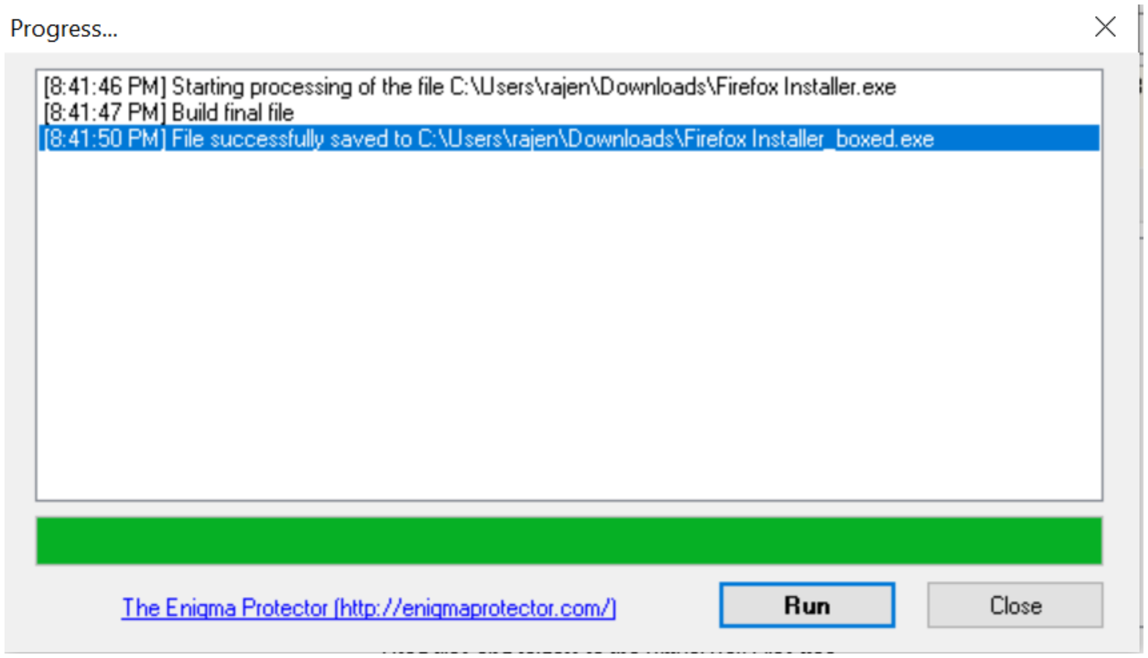
The Enigma Protector (http://enigmaprotector.com/)

Figure 13: Enigma virtual box window

After opening the app, we can see window as above screen shot. This Enigma virtual box will not take any pre or post snap shots of system. Here we need to upload the path of the app we need to convert to portable application. After adding path in text box enter input file name, we will press the process button at the bottom, then it will start processing the conversion and we will another window.

We can virtualize the applications in our system just by drag and drop. It means we can just drag the application that needs to be virtualized and just press process. Then a boxed application that is portable application is created in the folder path as specified.

2. After some time of processing we will see a pop-up window showing the result that portable version of the app is created that is it will show that a boxed version of the pp was created and stored in the path as shown in the screen shot below.



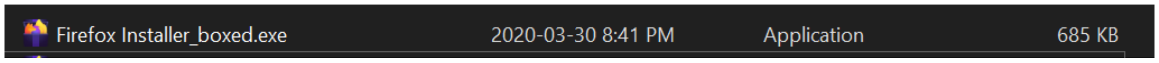
```
[8:41:46 PM] Starting processing of the file C:\Users\rajen\Downloads\Firefox Installer.exe
[8:41:47 PM] Build final file
[8:41:50 PM] File successfully saved to C:\Users\rajen\Downloads\Firefox Installer_boxed.exe
```

Figure 14: App converted successfully

After successful creation of the portable or boxed app we can run it from there if we want by pressing the run button at the bottom.

Here to covert an app to portable app I have used the normal version of fire-fox

3. Now we can observe the portable version of the fire-fox app in the path specified.



| | | | |
|---|---|---|---|
| Firefox Installer_boxed.exe | 2020-03-30 8:41 PM | Application | 685 KB |

Figure 15: Portable app by Engima

From the above screen shot we can see that fire-fox is converted to portable version of the application.

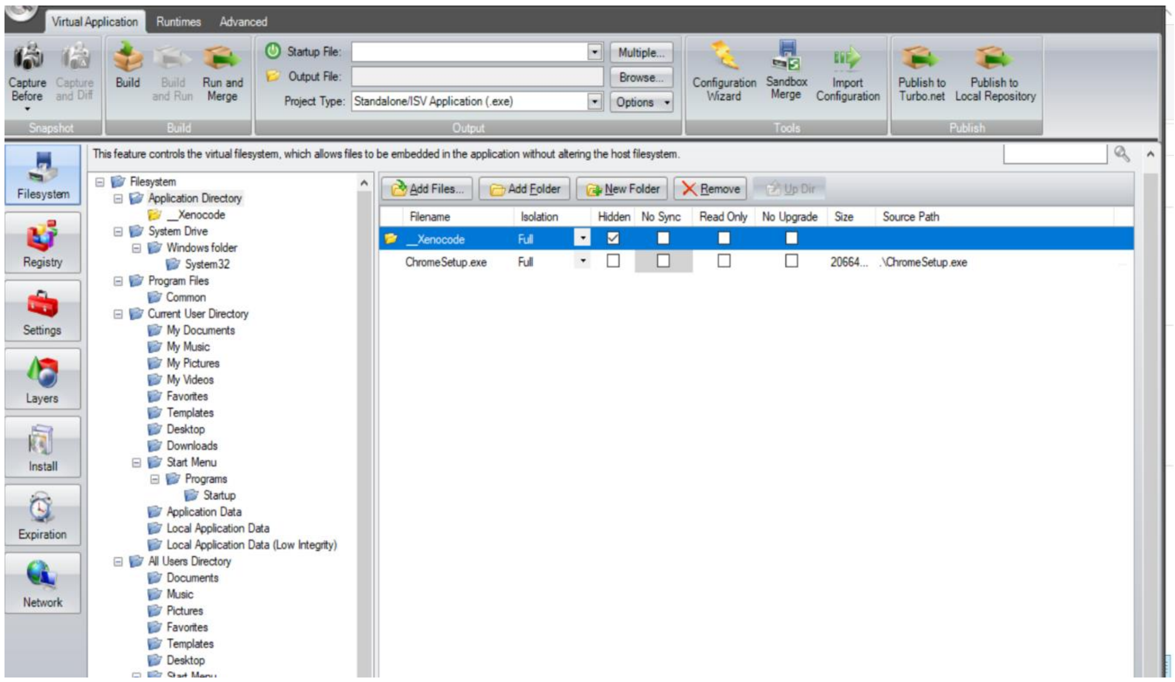
So, from the above steps we can see that how a normal installed application can be converted to portable version. While executing the Enigma virtual box I have noted the time required to conversion which is around 5.4 minutes and file size of Enigma virtual box is 6.8Mb. It will be creating only 1 file in the path specified.

The user rating for the Engima virtual box is 3.7 and total number of downloads for this software is around 10694.

### 3.2.3 Executing Spoon Studio:

Now we will see how the Spoon studio will be executed. First, we need to download and install the Spoon studio tool from the internet. Now a days the spoon studio is named as Turbo studio.

1. After installing the Spoon or Turbo studio we need create the account with username and password. After creating account and signing with the credentials we can see a window containing several options and directory files.



Project Type: Standalone/ISV Application (.exe)

This feature controls the virtual filesystem, which allows files to be embedded in the application without altering the host filesystem.

| Filename | Isolation | Hidden | No Sync | Read Only | No Upgrade | Size | Source Path |
|---|---|---|---|---|---|---|---|
| __Xenocode | Full | ☑ | ☐ | ☐ | ☐ | | |
| ChromeSetup.exe | Full | ☐ | ☐ | ☐ | ☐ | 20664... | .\ChromeSetup.exe |

Figure 16: Spoon studio window

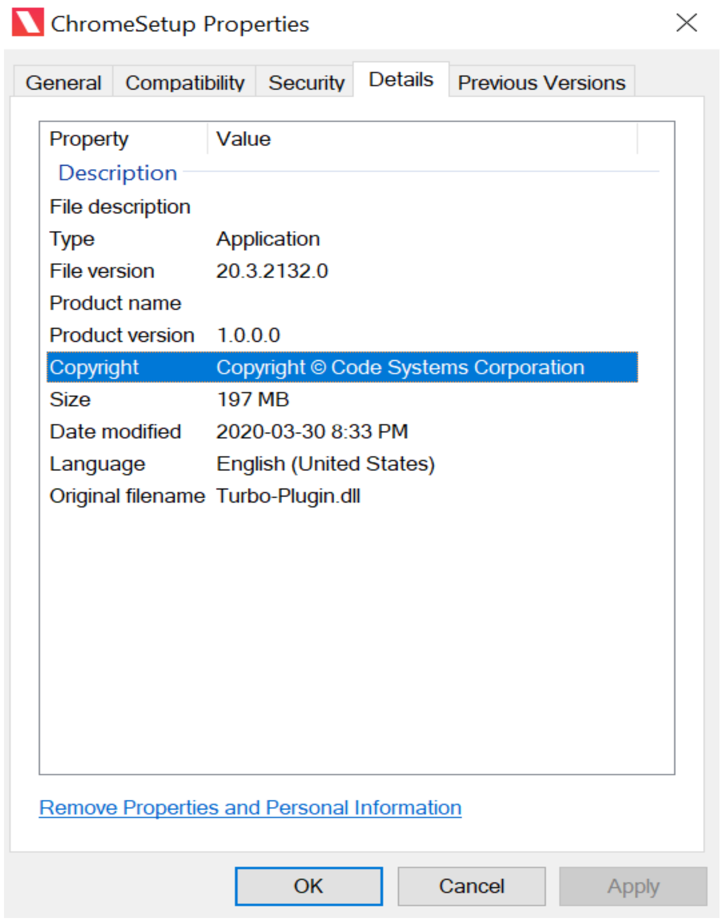
Now we need to select the filesystem on left pane and in that file system we need to select application directory. Then we will be shown with option to upload and run. After that we will be uploading the application from our system which we need to convert to portable application.

In this scenario I am using google chrome installer to be converted into the portable version as shown in the above figure.

After uploading the application to the spoon studio file directory, it will start processing and create the portable version of the application in the same path where the original application exists.

2. Once we upload the chrome installer and then we press build button on the top menu ribbon. The it will start processing and once the conversion was done, we will get a pop-up window showing successful creation of the portable version of the app.

   The portable version of the chrome installer created from spoon studio is as shown in the screen shot below.



| Property | Value |
|---|---|
| Description | |
| File description | |
| Type | Application |
| File version | 20.3.2132.0 |
| Product name | |
| Product version | 1.0.0.0 |
| Copyright | Copyright © Code Systems Corporation |
| Size | 197 MB |
| Date modified | 2020-03-30 8:33 PM |
| Language | English (United States) |
| Original filename | Turbo-Plugin.dll |

Figure 17: Portable application by spoon studio

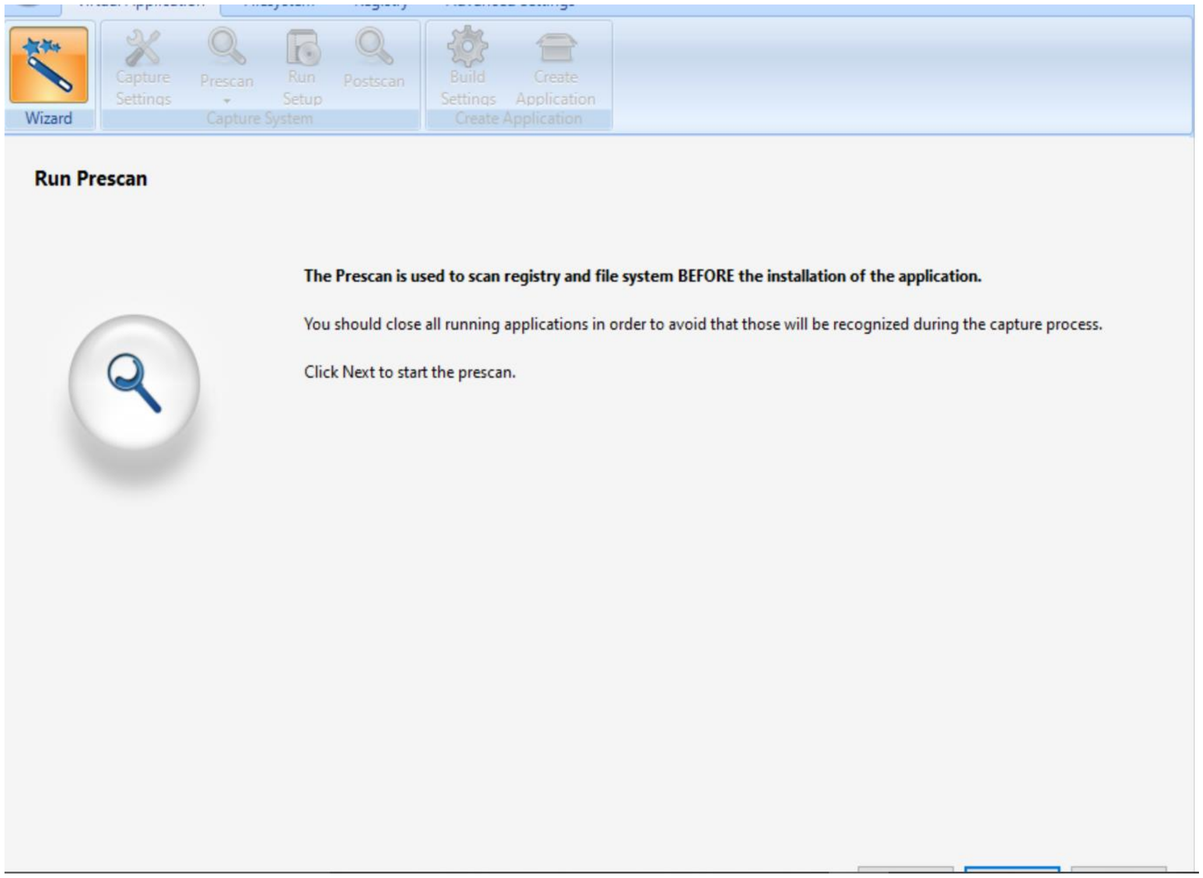
   From the above screen shot we can see that chrome installer was created from the turbo studio or spoon studio basically which are same. We can also see the logo of the chrome set has change stating that it was portable version created from the spoon/turbo studio.

So, Finally I will conclude with the time taken for the conversion of normal app to portable app using spoon studio is around 6.5 minutes and number of files created in the directory are 1. The size of Spoon studio tool is around 21.7Mb.

## 3.2.4 Executing Evalaze Tool:

To execute and observe the running of the Evalaze we need to first download and install the Evalaze tool. After installing the Evalaze we can observe the process of converting the installable app to portable app.

After opening the Evalaze tool we can see Evalaze application wizard which will perform all the pre scan and post scan process to find changes in the system or we can perform them manually.



Run Prescan

**The Prescan is used to scan registry and file system BEFORE the installation of the application.**

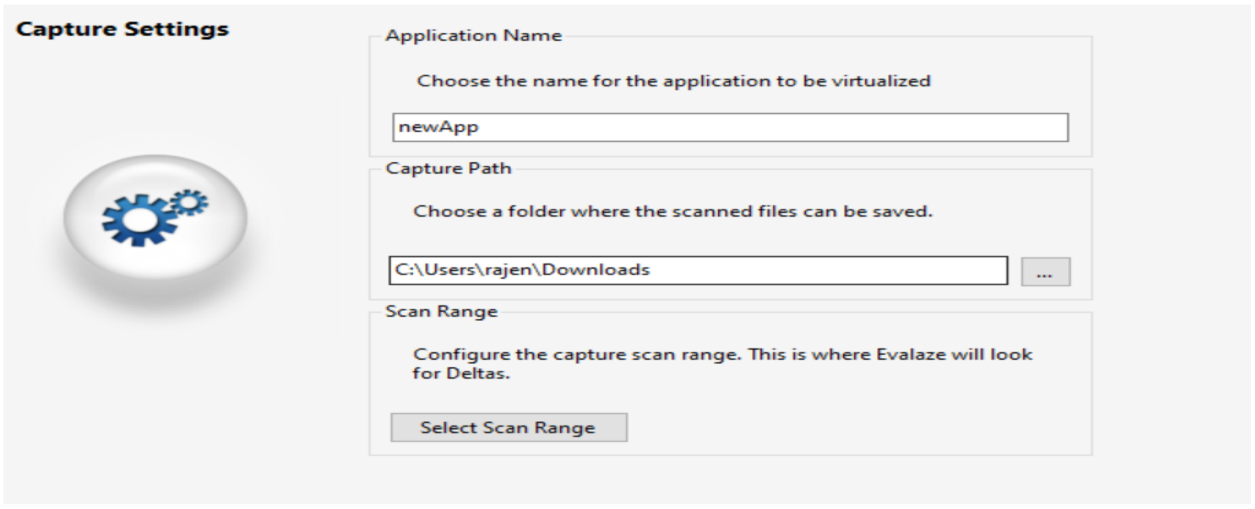
You should close all running applications in order to avoid that those will be recognized during the capture process.

Click Next to start the prescan.

Figure 18: Evalaze pre scan

After running the pre scan the Evalaze will create a capture folder in the c drive. This capture folder in the c drive will have the applications that are converted to portable applications using Evalaze. we can see the Screen Shot of portable fire fox application in the c drive capture folder which was provided in the later section.

By pressing next option at the bottom Evalaze will start pre scanning the system. Before pre scanning the system, we can set the range of scanning like to scan only c drive or to


**Capture Settings**

Application Name

Choose the name for the application to be virtualized

newApp

Capture Path

Choose a folder where the scanned files can be saved.

C:\Users\rajen\Downloads

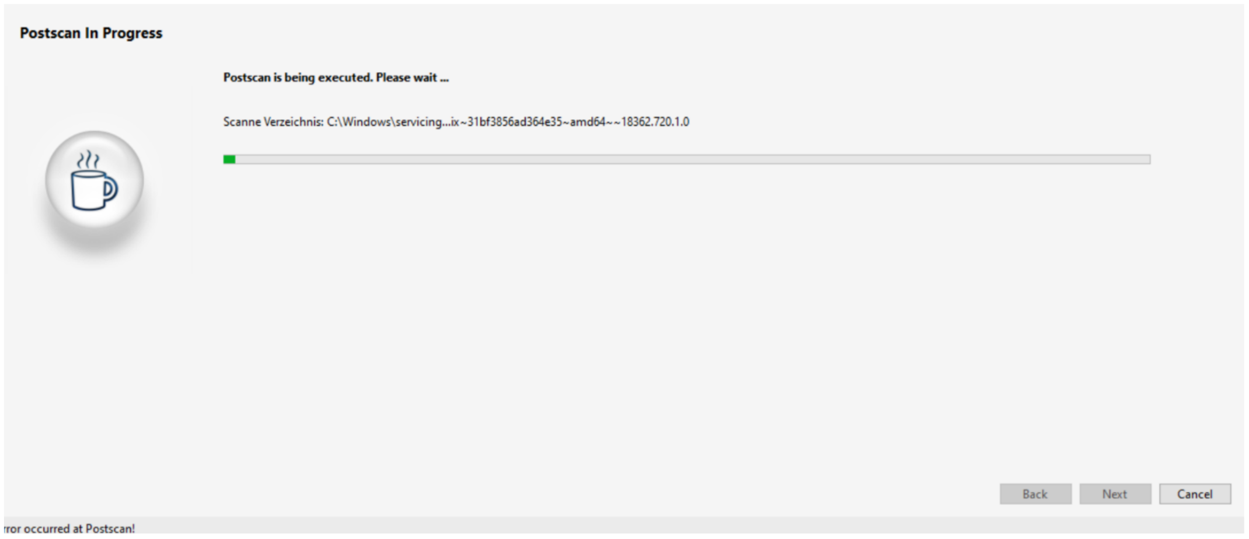
Scan Range

Configure the capture scan range. This is where Evalaze will look for Deltas.

Figure 19: Scan settings

By clicking the select scan range we can confine the pre scan range or we can increase the scan range of Evalaze assistant wizard. While this pre scan was done it will create capture folder in the c drive where we can find our portable apps created using Evalaze.

3. After pre scan was done next we need to install the application that needs to be converted into portable app is installed and then we will start post scanning of system as shown below.


**Postscan In Progress**

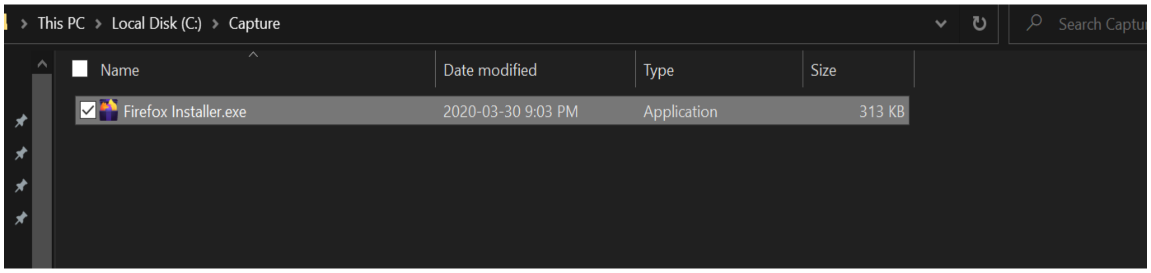
Postscan is being executed. Please wait ...

Scanne Verzeichnis: C:\Windows\servicing...ix~31bf3856ad364e35~amd64~~18362.720.1.0

rror occurred at Postscan!

Figure 20: Post scan spoon studio

Once the post scanning was done, we will see the window containing the list of applications that newly added after pre scan or while performing the post scan. In the list we will select the fire-fox and then will press next option at the bottom of window then the installed fire-fox will be converted to portable app and stored in the capture folder in the c drive as shown below.


This PC > Local Disk (C:) > Capture

| Name | Date modified | Type | Size |
|---|---|---|---|
| Firefox Installer.exe | 2020-03-30 9:03 PM | Application | 313 KB |

Figure 21: Portable app created

From the above we can see that fire-fox is converted to portable and we can place it in the USB drive and can be used any where we need.

So, finally after executing the Evalaze I came up with certain values like time need to conversion which is around 7.4 minutes and number of files created will be around 1. The size of Evalaze tool is 10Mb.

After observing the process of execution and different values like time for conversion, rating, size and number of files we can conclude that Cameyo is the best open source tool.

## 3.3 Comparison Between Different Portable App Tools:

In this section we create a comparison tables and indulge some important aspects of each portable app based on the performance and values we observed in the previous section by executing the various portable app creating tools.

### 3.3.1 Pros of using Portable App Tools:

1. Cameyo:
    - Cameyo is Open Source tool used for converting installable app to portable application. Once Cameyo tool was opened it will show a window with less and most important options for making portable apps using the effective graphical user interface.
    - The three options provided by Cameyo are Capture installation, Cameyo player and edit package. Here the capture installation is used to perform initial snapshot of system and final snapshot of system. Edit package is used to edit any installable application to portable application. Cameyo player is used to virtualize the applications.
    - As I have used the Cameyo tool for converting installable application to portable application, the best things I have observed is, it is very easy to use. The user without any prior knowledge can use these Cameyo Tools because there is nothing like uploading or giving paths. Just one need to open Cameyo tool and select the

one of the three options provided and the Cameyo tool will take care of the remaining things.

- The portable applications created using Cameyo will be saved with .Cameyo extension which represents a portable application.
- Coming to the performance of the tool aspects, the performance of Cameyo depends upon the system performance because to convert installable application to portable application Cameyo will take pre and post snapshots of the system which is a little lengthy process, but if the system on which Cameyo is running not performing any other tasks it will be done in less time.
- But the application created using the Cameyo are quite stable and can be easily used in other system by sticking on USB with out need to install any addition features in the system.

2. Enigma Virtual Box:
    - Enigma Virtual Box once downloaded will provide an option to set it up as desktop icon. Once it is installed, we will open the Engima virtual box it will show a window containing two text boxes and one process button at the bottom.
    - Here we need to upload the path of the installed application in input file text box and just need to press the process button. The Engima virtual box after process will show a pop-up window stating the successful creation of portable application. There in that window we can see run button which will used to execute the portable application.
    - After using the Enigma virtual box for portable app creation, I felt that it would be very easy to use as there is not much user interference needed as he just need to give path of the application that needs to be converted to portable application.
    - Coming to the performance aspects of the tool this tool is little bit slower when compared with the Cameyo in the terms of time for creating the portable application.
    - But the application created using the Enigma Virtual box are quite stable and can be easily used in other system by sticking on USB with out need to install any addition features in the system.

3. Spoon Studio:
    - Spoon studio once installed and open will show a big window containing lot of options which is bit confusing, but I thought it will easy to use after making proper observation to each of the option provided.
    - It provides an option to drag and drop the application or to upload the path of the application and then just press process or build button which will process the application and will create a portable application.
    - The spoon studio provides the feature of sand box environment and virtual file

system which will be created newly on top of the host file system without causing any problems to the host file system.

- As per the performance spoon studio will exhibit less better performance compared to Cameyo and Engima virtual box as per observation, because it is taking too long to create the portable application.

4. Evalaze:
    - After installing and opening the Evalaze we can see a window showing the Evalaze wizard assistant which will be helpful in the process of creating the portable application.
    - This Evalaze assistant will provide the step by step options to create the portable version of the normal application.
    - First this Evalaze wizard will pre scan the system, this time for the pre scan depends on storage or memory used in the system as it will scan the drives in the system.
    - But Evalaze will provide more advanced option like setting the scan range that is allowing Evalaze to scan only particular drive or folder or directory.
    - Thus, we can increase the performance of Evalaze by using the scan range option, as the range of scanning is less the performance will be more.
    - Later Evalaze will do the post scan to find the changes made and will show the newly added applications. We just need to click on the app we need a portable version and then the normal app will be converted to portable app and will be placed in the capture folder in the c drive.
    - As there is Evalaze assistant, it will be very ease to use it and performance of Evalaze again depends on the system of user.
    - But the application created using the Evalaze are quite stable and can be easily used in other system by sticking on USB with out need to install any addition features in the system.

### 3.3.2 Cons of using Portable App Tools:

There are some cons or disadvantages are observed during the use of tool and applications created by the tools.

1. Cameyo:
    - Cameyo performance will be depending upon the system performance, as if we are using system for performing other tasks while running the Cameyo, the performance of the Cameyo will be reduced.
    - Cameyo does not provide the desktop icon, every time we need to use Cameyo we need to open the Cameyo.exe file. I think they should be providing the desktop application or application which will increase the ease of using the Cameyo

2. Enigma Virtual Box:
    - The Apps created by using the Enigma virtual box was not working sometimes. These portable app created using Enigma virtual box is working properly sometimes without any mal functioning but sometimes the application is not even opening.
    - Even the enigma is providing the desktop application, this application is cracking sometimes there by we need to install it again.
3. Spoon Studio:
    - Coming to the spoon studio, after drag and drop the application to spoon studio or providing path of application in spoon studio then we need to press build button. But sometimes this build option is not working which means we need to again open the spoon studio and rerun it again.
    - The applications created by spoon studio will be working properly on the host system, but when we need stick it on USB and need to use it on other system, we need to provide login credential of our spoon studio account.
4. Evalaze:
    - After performing pre scan and post scan Evalaze will give list changes made in system after pre scan and before post scan. In that list we should find the newly installed application. But sometimes after post scan it will be showing the empty list without any applications which will be leading to run the pre and post scan again thereby wasting the time of user.
    - May be after running for second time we could sometimes find the application list but even after some cases it’s not showing which will be leading to running of Evalaze a greater number of times. This will even be affecting the performance of creating the portable application leading to greater times for converting to portable application.
    - Even the performance and scan time of system depends on the occupied storage space in drivers. If the drivers are fully occupied it will take more time for pre and post scan of systems by Evalaze.
    - The applications created by the Evalaze are quite stable.

Thus, above both sections we have discussed the advantages and disadvantages observed by me while running or executing these portable app creating tools and apps created by using these tools.

### 3.3.3 Comparing the Product related Information of Tools:

In this section we will be comparing the information related to the portable application tools like size of the portable application tool, price for using premium versions of tools and rating given by different users.

1. After downloading the free open source portable application tools, I have taken a note of the size of each tool. The size of each tool is represented in megabytes. The size of Cameyo tool is 14.7MB, spoon studio is 21.7MB, Evalaze is around 8.4MB and size of Enigma virtual box is around 6.8MB.
2. Later I have observed the pricing of different tools for professional versions. The Cameyo will be costing around 144$, Enigma virtual box will be around 199$, Evalaze there is no professional version, so there is no price for Evalaze and spoon studio will be costing around 96$ for the professional versions.
3. Even I have observed the user ratings for these tools in their official websites. The user rating for Cameyo is 4.5, spoon studio is 4 , for Evalaze is 3.8 and finally for Enigma virtual box is around 3.7.
4. Even some of the organizations of these tools provided the number of downloads made by the users. The number of downloads for Cameyo is 35067 and for spoon studio is 5446.

The below table will contain all the product data related to the portable application creating utilities.

| Tool Name | Price | Size of Tool MB | User Rating for Tool | Downloads No of Tools |
|---|---|---|---|---|
| Cameyo | 144 $ | 14.7 | 4.5 | 35067 |
| Spoon Studio | 65 $ | 21.7 | 4.0 | 5446 |
| Evalaze | 0 $ | 8.4 | 3.8 | 6703 |
| Enigma Virtual Box | 199 $ | 6.8 | 3.9 | 10694 |

Table 1: Product data Comparison

### 3.3.4 Output Data comparison:

In this section we will discuss some of the data that is gathered after running or executing the portable application creating utilities and using the portable app created from these tools.

Mainly the testing of these tools done for converting the fire-fox web browser application into the portable version. By using different tools for converting the fire-fox to portable version I have observed the various aspects like original size of fire-fox and the size of fire-fox after converting into portable application by using above mentioned tools. The observations made are,

1. The size of original fire-fox application is 313 Kilobytes. Now we will observe the size of fire-fox after converting into the portable version
2. The size portable version created by Cameyo is around 475 Kilobytes
3. The size of portable version created by Evalaze is around 580 Kilobytes, by Enigma virtual box is around 686 Kilobytes and by spoon studio is around 626 Kilobytes.
4. In this observation I have noted that size of portable version of the application actually increased, so of the portable versions created by tools is double the size of original application. I think this might be due creation of virtual file system.
5. Even I have noted the number of files each tool will be creating in the process of converting application to portable version. But my observations found that each tool will be creating only one application. No additional files are created.
6. While running these portable application tools I have noted the time taken by each tool for converting the fire-fox original application to the portable version.
7. The time taken by Cameyo to convert Fire-fox to original version is around 4.2 minutes, for spoon studio is around 6.5 minutes, for Evalaze is 7.4 minutes and for Enigma Virtual box is around 5.4 minutes.

The table below will represent the output data and observations made while converting original fire-fox application to portable version like original size of the fir-fox file, name of the tool, time needed for converting normal application to portable application in this case installed fire-fox application to the portable fire-fox application and finally number of files each tool was creating while converting normal application to portable application.

| Name of Tool | Original Size of fire-fox | Portable app Size of Fire-fox | Time for Conversion | Number of files Created |
|---|---|---|---|---|
| Cameyo | 313 KB | 475 KB | 4.2 Min | 1 |
| Evalaze | 313 KB | 580 KB | 7.4 Min | 1 |
| Spoon Studio | 313 KB | 626 KB | 6.5 Min | 1 |
| Enigma Virtual box | 313 KB | 686 KB | 5.4 Min | 1 |

Table 2 Output data Comparison

### 3.3.5 Ease of use for Different users:

In this section I will discuss about the easiness of using the portable application creating utilities by different users

Here for the Observation purpose the users will be divide into 3 categories, they are Beginner, Moderate and High-end user.

1. For the beginner I think using Cameyo, Evalaze will be very easy  because there will a smaller number of options to choose for and to perform the portable app conversion. Mainly Evalaze will have a wizard assistant which will be helpful for the beginners as it will provide the step by step assistance.
2. For the moderate level user who have little bit of experience in using the portable application tools, they will find it very easy using the Cameyo and Evalaze. They can feel quite comfortable with Enigma virtual box too. Because in Enigma sometimes we will be running into problems like crash of tools and improper working of created application, which can be corrected if we have moderate level of knowledge using these portable application tools.
3. For the high-end users they might feel comfortable using all the four portable application tools including spoon studio where it will provide us with some user defined high end options like creating or merging the sand boxes option, can edit the virtual file system. In order to use all these, option effectively I think a high-end user with ample of experience can feel it easy.

Here I am extend my observation based on easiness to toughness of using the portable application tool by above categorized users.

1. Beginner who just started using these tools will find easy to use Cameyo, little difficult to use Evalaze, will find harder to use Spoon Studio and Enigma virtual box
2. Intermediate experienced user who have intermediate knowledge about these tools will find easy to use Cameyo, Evalaze and will find little difficult in using Enigma virtual box and will find harder to use Spoon Studio.
3. For high-end user who has we defined knowledge using these tools will find easy to use Cameyo, Evalaze, Enigma virtual box and will find little difficult to use Spoon Studio.

The below table will show which tool is easy to and which tool is harder to use for different kinds of users discussed above.

| User | Cameyo | Evalaze | Enigma Virtual box | Spoon Studio |
|---|---|---|---|---|
| Beginner | Easy | Little difficult | Harder | Harder |
| Intermediate | Easy | Easy | Little difficult | Harder |
| High-end | Easy | Easy | Easy | Little difficult |

Table 3 users' comparison

So, finally these are my observation which I have made while using the portable application tools and creating the portable applications of installable applications by these tools.

# 4 FORKING THE BEST OPEN SOURCE PORTABLE APPLICATION CREATING UTILITY WITH SOME RECOMMENDATIONS FOR IMPROVEMENT

This section highlights the features of the best open source portable app creating utility and some recommendations for improvements in the performance to enhance further.

## 4.1 Forking the best Open Source App creating utility based on product information.

The best open source App creating utility is Cameyo[14] which yields a success rate of 85% i.e., most of the installable app can be converted to portable using Cameyo which yields success rate of 85%. Cameyo streamlines[15] the software needs by allowing to carry the favorite or fundamental applications. It likewise assists free with upping storage space on the computer while liberating from installing and uninstalling occasionally utilized applications. Portable packs can also be made for occasionally required applications and run their portable versions when it's really needed.

The made portable applications[15] can likewise run in HTML5 browsers like Firefox, Chrome, and Safari etc. Only thing needed is to associate with the Internet and dispatch those applications from Cameyo server on the objective computer.

Even by comparing download size of the tool with another open source tool, the size of Cameyo is 14.7Mb which is much lesser in size than spoon studio but exhibits better performance than spoon studio and even for beginners will find it easy to use the Cameyo. Yes it is true that other tools like Enigma virtual box and Evalaze are lesser in size compared to the Cameyo, but we observed in the previous section that user ratings for Cameyo is much higher than any other tool which means users felt easy to use Cameyo and got better results of converting a normal application to portable application when compared with other tools.

## 4.2 Best Open Source App Utility based on Output Information.

Cameyo makes portable bundles with cutting edge virtualization technology that makes them portable on different Windows frameworks[15].

- It is a free option in contrast to different business-grade expensive portable application utilities like VMware ThinApp, and Spoon Studio.
- Cameyo likewise give tough competition to different kinds of free portable app creating utilities or suites PortableApps.com, Evalaze, and Enigma Virtual Box, etc.

Based on the data provided in the previous section, I  have observed that Cameyo is taking much lesser time for converting the installed application to portable application which is around 4.2

minutes which is lesser than time taken by any other tools for converting application to portable application.

Coming to the usage of the tool, I felt it is very easy to use Cameyo than any other tools because of its confined options which are available is easy to understand. Once Cameyo is opened and once user selects on e of the three options provided, then that's it Cameyo will do the remaining work of converting application to portable version by less user intervention.

So, therefore by observing all the product related data and data that has been collected while running different portable application creating utilities, I can conclude that Cameyo is the best open source portable app creating tool.

The best out of all other utilities are as follows[15]:

1. Soloness: Portable applications made using Cameyo can be utilized on any framework without the need of downloading the software. It makes portable bundles, which don't require any plugins or specialists for executing on the objective Windows framework.
2. Cloud features: Cameyo can be utilized to make cloud bundles of applications, which can without much of a stretch run in any HTML5-compatible browsers. It enables to run portable applications on different objective platforms without the need of installing any outsider software like Wine, etc. The cloud packaging highlight is selective to Cameyo and is not accessible in other application bundles.
3. Information encryption: Cameyo enables to scramble the information in the portable bundle, subsequently securing it against unauthorized use. This component too is not accessible in other portable application creators.
4. Run on all well-known OS's: Cameyo cloud packaging highlight enables its applications to run on any mainstream operating framework, name it as Windows, MacOS or Linux and on any HTML5 program on these OS. This is beyond the realm of imagination in other portable application developers as their portable bundles can run only on Windows platforms.
5. Application information on USB/Dropbox: Portable applications made using Cameyo can store their information and settings on a USB drive which is local storage device or Dropbox/cloud online storage service.
6. BYOD-accommodating (Admin-less): The portable software bundles made by Cameyo are really BYOD-accommodating (Bring Your Own Device). It's conceivable because these

portable applications require neither administrator rights nor installation of certain special programs or plugins on the objective framework.

7. Portable application creation by limited clients: Cameyo permits even the limited clients and non-administrators to make portable applications on a framework. This component is either not accessible in other portable application creators or may require admin benefits for creating portable versions of certain applications.
8. Make client confinements: Cameyo enables engineers to make different types of client confinements on the portable bundles like active directory group, expiration, password, scripting, etc.
9. Various Virtualization Modes: The portable applications made by Cameyo can run in two modes of virtualization – RAM and hard disk. This enables individuals to run these portable applications on Windows frameworks having less RAM or memory. This double mode usefulness is not present in any other portable application creators.
10. Software Developer's Kit (SDK): With Cameyo allows to make SDK for portable applications effectively and deftly. It offers advanced APIs that enable designers to include/expel files from existing bundles, list virtual applications running on a framework or web service to trigger virtual packaging, etc.
11. Open application directory: Cameyo's online directory of portable applications can be obtained easily. This online library enables to download portable bundles of different well known and commonly utilized open source and free applications. It provides provision to essentially download portable bundles according to the prerequisites without the problem of creating them.

The portability of any application can be achieved using Cameyo. Portable applications made using Cameyo can be utilized on any framework without the need of installing the software.

SDK offers advanced APIs that enable designers to include/expel files from existing bundles, list virtual applications running on a framework or web service to trigger virtual packaging, etc.

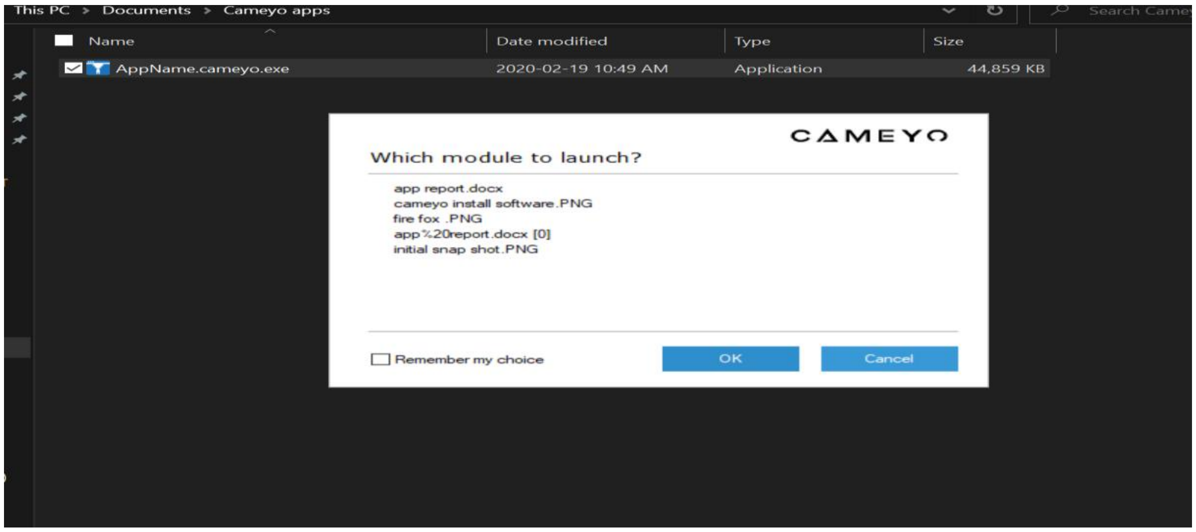
## 4.3 Improvements

There are some improvements that can be made to any invention to make it better. similarly, Cameyo and VMware Thinapp suffers from some flaws and certain improvements can be made.

### 4.3.1 Improvements

I will fork the Cameyo open source utility with some improvements as follows. The Cameyo portable app creator works only with low or mid-level apps. The improvements that needs to make are:

- Make it work with high level apps such as Adobe reader and Calibre (eBook management).
- Include an application menu for easy search operations. It's been including as Cameyo showing the list of things performed after taking system snapshot:



| Name | Date modified | Type | Size |
|---|---|---|---|
| AppName.cameyo.exe | 2020-02-19 10:49 AM | Application | 44,859 KB |

Which module to launch?

app report.docx
cameyo install software.PNG
fire fox .PNG
app%20report.docx [0]
initial snap shot.PNG

☐ Remember my choice

Figure 22: Menu in Cameyo

# 5 CONCLUSIONS AND RECOMMENDATIONS

## 5.1 Conclusions

The portable Applications are more advanced and essential one in many growing sectors. Thus, creating those applications to be portal is done using certain portable app creating utilities. Those utilities were explained and analyzed in terms of its performance. Then they were compared to find the best open source portable app creating utility. Cameyo is the forked to be the best open source portable app creating utility which provides 85% success rate and the best closed source portable app utility is VMWare thin apps with success rate equal to 95%.

## 5.2 Recommendations

The recommendation to make in the Cameyo open source utility is, it lacks an app menu that can be used to list and search for installed or portable apps already stored.

For the closed source VMWare thin apps, the recommendation is to make it work with 64bit applications.

The Cameyo portable app creator works only with low or mid-level apps. The improvements I would like to make are:

- Make it work with high level apps such as Adobe reader and Calibri (eBook management).
- The source code for most of the tools is closed source, I would recommend developing some more open source app creating tools.
- While taking pre and post snapshots of system to find the newly installed application Cameyo is taking more time which is not user friendly and the time for taking snapshot needs to be reduced.
- There is no portable app software which can make the portable operating system. I would recommend making a portable software that could convert the OS file to portable files, so that one can carry their own operating system in a USB drive.

Finally using and maintaining the portable app is very easy compared to installable app. So, the organizations like Microsoft can implement their own portable apps which makes users of MS office or some other software's easy to carry around and maintain them in a single folder with need of less memory.

# 6 SOURCE CODE STRUCTURE

## 6.1 Source Code Organization:

The source code the appendix is an open source code for the Cameyo software which was written using C# language. This Source code was executed using the Microsoft visual studio.

1. Firstly, the source code will be started using the pre-defined libraries to compile and run the program. The pre-defined libraries are which are available to use readily in the source code with out need to write any functions or classes to define them.
2. The pre-defined libraries used are console.IO which is used to handling the console input and output operations, Microsoft. Win 64/32 libraries are used to provide packages need to run on 64-bit or 32-bit systems.

### 6.1.1 Flow of Source Code:

Firstly, in this source code we will import all the libraries need for compiling and executing the program perfectly. There are several libraries being imported using Dynamic link libraries, they are:

1. DLL Imports: In this we will define functions to import the DLL libraries like package open, package save, package get property and package set property. These package libraries are useful for forming the packages in the Cameyo.exe when executed. line 127

   Some of the DLL imports are explained below

   - Package Open: Opens an existing virtual application package for read/write. Returns a handle that can then be used for manipulating virtual application packages. line 130
   - Package Create: Creates a brand-new virtual application package from scratch. Returns a handle that can then be used in the same way as Package Open. line 151
   - Package Close: Closes the virtual application package and cleans up the temporary files and registry created by package open. line 173
   - Package save: Saves the changes made to a virtual application package. line 189
   - Package Get Property: Retrieves the property assigned to the virtual package. line 205
   - Package set Property: Sets or used to modify the property or value assigned to package. line 227
   - Package Set Icon File: Used to modify the icon of the virtual package. line 246
2. Virtual File System Imports: In this we will import all the libraries need to form and operate the virtual file system on the top of the original file system. The libraries like Virtual file

system add, Virtfsadd, Virtfsextract, virtfsdir are being imported to perform the virtual file operations. line 284. Some of the Virtual file system imports are explained below.

- VirtFsEnum: Enumerates the files in the virtual package system. line 297
- VirtFsAdd: Adds a file to the virtual application package. line 316
- VirtFsAddEx: It is same as the VirtFsAdd but used to control the file flags of the added file. line 338
- VirtFsDelete: Deletes a file from the virtual application package. line 401
- VirtFsExtract: extracts a specific file from the virtual application package and saves it some where on the disk. line 382
- VirtFsGetFileFlags: It will return the file flags for a specific file or directory. 583
- VirtFsSetFileFlags: It will be used to set the file flags to specific file or directory. 602
- VirtRegGetWorkKey: It will provide the image of the virtual registry as contained in the virtual package. line 437
- VirtRegGetWorkKeyEx: It is same as VirtRegGetWorkKey function but with an additional event that caller can signal to abort the progress of this function. It will provide the image of the virtual file registry as contained in the virtual package. line 456
- VirtRegSaveWorkKey: Applies the changes to the modified registry keys obtained from VirtRegGetWorkKey function. The workflow for modifying the virtual registry is call VirtRegGetWorkKey (EX), modify the registry work hive, call VirtRegSaveWorkKey and call package save. line 478

3. Sandbox Imports: These sand box imports are used to import the sand box libraries which are useful to run the program in a sand boxed environment. Which means to execute the source code we need a restrictive environment which is provide by importing these sand box libraries. The sand box libraries like sandboxsetregistryfile, sandboxsetfileflags are used to provide the sand boxed environment. line 494.

   Some of the sand box imports used in this program are explained below:

   - Sandbox Get Registry Flags: It will return the registry flags from the specific registry in a sand boxed environment. line 495
   - Sandbox Get File flags: It will return the file flags for a specific file or directory in isolation mode. line 517
   - Sandbox set Registry flags: It will assign flags to a specific registry in a sand box environment. line 539
   - Sandbox set File flags: It will be used to assign the flags to specific file or directory. line 561

4. Deployed app Imports: These deployed app libraries are used to provide the all resources used to deploying the app and make available for users. Here these libraries are being imported to make this Cameyo program available for users and can able to run on all

systems. The libraries like deployedappenum. Deployedappgetdir are being imported to carry out the functions of deployment software. line 737

Here we will discuss some deployed app imports used in the source code

- Deployed app Enum: It enumerates virtual applications currently deployed on the user's profile. This routine enumerates the software or OS registry keys. line 742
- Deployed app Get Directory: It will be providing the data related to the directories in the currently deployed virtual applications. line 758

5. Running App Imports: These libraries are useful for the proper execution of the source code. The libraries like runningappenum and runningappenumkeepalive are used to carry out the operations later defined in the running app functions in the source code. line 787

   Now we will discuss the running app imports used in the source code:

   - Running App Enum: It will be used to enumerate the currently running virtual application in the current user session. line 788
   - Running App Enum Keep Alive: It is used to keep the session of running virtual application alive without being crashed. line 804

6. Debugging App: These debugging operations are useful for importing the debugging file which has implementation for the debugging app function used for debugging the app. Here we have imported to implementation files to carry out the operations of two debugging function where one is 64-bit and other one is 32-bit. line 835

   Here we will discuss the imports for debugging app:

   - Debugging App: This import function will be used to check for any internal errors in the running virtual application. line 839

All the libraries or file that are imported are will be in both 64-bit and 32-bit to carry out the functions and operations in the both 64 and 32-bit environments.

All the above implementation files or libraries are being imported to execute the operations in the functions defined at the later part of the code.

As we know in order to have a perfect organization and structure for the code, we need to define the classes. Here we will have the running app and deployment app class which will have call backs to the running app and deployment functions.

Now we will discuss the classes used in the source code, they are

1. Running app class: The running app class is user defined public class which will contains the functions as follows: line 870

   - Public string App Id: It will assign the unique virtual application Id.
   - Carrier Exe Name: It will be the name of executable used to launch the application.
   - Serial Id: It is serial number which is automatically increased for every launched virtual application.
   - Carrier PID: It will assign the process ID for the process which first started the virtual application.

- Start Tick Time: It is equal to get Tick Count () at the time of virtual application was created.
- Friendly Name: It is used for assigning more descriptive name for the application.

2. Deployed App Class: It is a user defined call which contains the following functions. line 1664
    - App Id: This function will give the unique app id of all deployed applications.
    - App Size: It will provide the size of each deployed application in current session.
    - Carrier Exe Name: It will provide the name of executables of each deployed application.
    - Occupied size: It will give the size occupied by applications in current session.
    - Exe Size: It will give the size of executables.
    - Base Dir Name: Provides the name of base directory containing these applications.

Now we will discuss the different kinds of user defined functions used in this source code:

1. Virtual File system Functions: In this first we will create a private function and then we will create public functions within that private function. The public functions like addfile, adddir, addfileextract are defined. These functions will carry out their operations using the files or libraries that are imported. These Virtual file system functions are useful to create the virtual file system on top of the hot file system. line 1163

    Some Virtual file system functions used in this source code are:
    - EnumFiles: It is a public function defined to enumerate and list the files in the virtual package of the system. line 1189
    - AddFiles: This public function is defined to add the files to virtual package by using source file name and destination file variable name. line 1198
    - AddFilesEx: It is similar AddFiles function but used control the files of added flags. line 1207
    - AddEmptyDir: It is public function used to define an empty directory in the system by using the source folder name which is newly adding directory and destination folder name to which this directory needs to be added. line 1225
    - AddDir: This function is used to add a directory of files by using source and destination variables. line 1240
    - ExtractFile: This function is used to extract a file from the virtual package by defining the file name which needs to be extracted and will be save on the disk where the location is specified by using targetdir variable. line 1265
    - DeleteFiles: This function is used for removing the unnecessary files in the package. line 1281
    - GetRegWorkkey: This function returns the image of the virtual registry as

contained in the virtual package. Returns APIRET_SUCCESS if successful, or APIRET_VIRTREG_DEPLOY_ERROR if the virtual registry key could not be written. APIRET_INSUFFICIENT_BUFFER is returned when the buffer size provided for WorkKey is not enough. line 1335

- GetRegWorkKeyEx: It works same as the GetRegWorkKey but will have an extra event where the user can abort the function process and re start it again if he wants. line 1309
- VirtRegSaveWork: This function is used to apply the changes made by above to functions to the registry and to save them. line 1343

2. Sand box functions: The sand box get and set functions which are user defined and public functions are used to create restrictive environment for program execution. Here the set property function is used to assign the certain values and properties and get property is used to get the return values from the imported files defined on the top of the source code. line 1359

   Some of the sand box functions are explained below:

   - Get Registry Sandbox: This function will return the flag values of specific registry. line 1360
   - Get File Sandbox: This function will return the assigned flag values of the required files. line 1368
   - Set Registry Sandbox: It is one of the set sand box function used to assign the properties to specific registry in the virtual package. line 1377
   - Set File Sandbox: This is a function used to assign the value to the specific file in a registry.  line 1383

3. Running App functions: Here we will define a set of private functions to carry out the execution operations of the programs by using running app class and libraries or implementation file being imported. The functions like runningappenum and runningappAlive are used to track the status of executing program. line 1431

   Some of the running app functions discussed in source code are:

   - DBG: It is a debugger function used to carry out local and remote debugging operations. line 1432
   - RunningappenumCallback: It is call back function to the runningappenum which will enumerate the running apps in the current user session.
   - GetRunningApps: This function is used to get the list of running applications in a user session. As we can the list running app attribute is used to present the list of running applications. line 1457
   - FindRunningapp: This function is used to find the specific running app from the list given by the GetRunningApps function. line 1474

4. Deployed App Functions: In this we defined a private and public functions to make the source code available to use. It means are the libraries and source files need for the compiling the program are imported into the sourced and are defined using these deployed

functions. The functions like enumdeployedcallback, list deployed apps are used which are useful to define and execute the imported files. line 1488

Some of the deployed app functions are discussed below:

- DeployedEnumCallback: This is call back function to the deployedappenum which will enumerate the deployed virtual applications in the user profile. line 1489
- DeplyedAppId: This function is used to assign the unique id to the deployed applications in the current session. line 1497
- List DeployedApps: This function will have list attributes defined which will be providing the list of deployed applications. line 1508
- DeployedAppDir: This function is used to define the set of directories in the currently deployed applications. line 1521

5. Dot Net wrapper functions: These functions are used to set the name and description got from the APIRET statuses. Here we add two additional cases for the status like APIRET.out of file descriptors and APIRET.no processors which will be helpful to assess the number of file descriptors being used and if we ran out of file descriptors, we need to close the already used file descriptors. line 941

   In this wrapper function we will have the switch case for various types of error handling conditions and APIRET (Application programming Interface Return Values) statuses.

   - APIRET_Success: This case will give the message as success showing the success status. line 969
   - APIRET_Failure: This case is to notify that the status of function is failure. line 970
   - APIRET_VirtualFile_DBError: This is case is to show that data base error in the virtual file system. line 971
   - APIRET_NotFound: This case to get the status as the resource or required file is not found. line 973
   - APIRET_File_create_error: This case is to notify there was error in file creation. line 976
   - APIRET_Pe_resourceError: In this we can see the virtual package resource error has occurred. line 977
   - APIRET_Memory_Error: This case is to define the errors occurred while handling the memory. line 978
   - APIRET_VirtReg_deploy_error: This case describes about the error in deploying the virtual registry. line 980
   - APIRET.Insufficient_Privileges: This case describes that user does not have

certain permissions to access the files or registry. line 986

- APIRET.Cancelled: It describes about the cancelling or aborting the ongoing operation, where these aborting is defined in the VirtFsAddEx function. line 989
- APIRET_DotNet_Required: It shows an error message if the host system does not have the dot net which is required for compiling this source code. line 988

6. Helper Functions: Some of the helper functions are defined to provide easy readability of source code and to perform some complex computations of deployed app functions. These functions are used to carry out if there are any excess of computations in any other functions defined. line 1533

   The helper functions used in this source code are:

   - FriendlyshortCutName: This function is used set more descriptive name for the application which will sending its return value to friendly name function in running app class. line 1533

The changes made in following source code are highlighted they are,

- some libraries needed for 64-bit processors is missing, so some libraries like Microsoft. win64 been added to the source code.
- In some functions in the source code there are set properties to assign value, but there is no get properties accessor to return the values to the functions, so the get properties have been added to functions which are not having get properties accessors.
- The missing 64-bit file attributes are added.

The ample changes that are made were explained in detail here,

1. Microsoft windows 64-bit libraries are added to import the required Microsoft libraries for running in 64-bit environment. line9
2. 64-bit file attributes are being added to source where necessary to provide access to the file from the 64-bit system. line26
3. Calls the function Virtfsopen32 if the app is 32-bit otherwise calls the function virtfsopen 64 which debugs the 64-bit app. line147
4. File flags for the 64-bit system environment is added to check flag status or file status. line289
5. Importing the file which has implementation for the function debuggingapp64 which debugs the 32-bit app. line834
6. Importing the file which has implementation for the function debuggingapp64 which debugs the 64-bit app. line839
7. This, functions check the app for whether it is 32 bit or 64 bit and call appropriate

debugging functions. line844

8. These functions are used to set the name and description got from the APIRET statuses. Here we add two additional cases for the status like APIRET.out of file descriptors and APIRET.no processors which will be helpful to assess the number of file descriptors
9. Setting name and description for the case APIRET.out of file descriptors. line974
10. setting name and description for the case APIRET.no processors. line995
11. This function returns whether the format is 64 bit or not (true. false) line1003.
12. This function create package and returns whether package is created successfully or not (true/false). line1080
13. Creating the package by calling packageCreate function which creates package and return the status of package creation. line1082
14. Mark the package is opened and return true if the status is success otherwise return false. line1084
15. This function returns the APPSIZE. line1699
16. This function takes name of appExe, packagerExe and hash table as input and return whether properties are set and executed correctly or not (true/false). line2012
17. Setting args by formatting string in the order properties and name of appExe. line2017
18. Executing the program by calling EXECProg function which takes packager name and reference of exit Code and returns whether program is executed successfully or not (true/false). line2023
19. Returns true only if the exec OK is true and exit Code is SUCCESS. line2024
20. This function returns the path for the packager.exe file. line2028

### 6.1.2 Problems During Code Execution:

The major problems faced while working with this source code the process of execution. For the first time when this code was executed it was creating .exe file in the folder path specified. But when trying to open that .exe file there was no action. So, I tried of adding the 64-bit file attributes, function where they were missing as specified in the changes stated above.

### 6.1.3 Draw backs of code:

This code still has some draw backs like graphical user interface. When we execute and open .exe file created by the source code there was no proper user-friendly graphical options. So, to make the use of Cameyo.exe some graphical implementation files and libraries can be imported.

.

# APPENDIX

```
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using System.IO;
using System.Collections;
using Microsoft.Win32;
using Microsoft.Win32.SafeHandles;
using console.IO;
using Microsoft.win64;
using Microsoft.win64.safeHandles;
Public static void Main
{

namespace VirtPackageAPI
{
public struct VirtFsNode
{
public String FileName;
public VIRT_FILE_FLAGS FileFlags;
public UInt64 CreationTime;
public UInt64 LastAccessTime;
public UInt64 LastWriteTime;
public UInt64 ChangeTime;
public UInt64 EndOfFile;
public UInt32 FileAttributes;
public UInt64 FileAttributes;
public Object ClientData;     // Arbitrary client data; you can use this
however you like
};

[Flags]
public enum VIRT_FILE_FLAGS
```

```
{
NO_FLAGS = 0x0,
ISFILE = 0x0001,                    // File or directory?
DELETED = 0x0002,                   // Deleted by virtual app (NOT_FOUND)
DEPLOY_UPON_PRELOAD = 0x0008,       // Force file deploy (Disk mode)
DISCONNECTED = 0x0010,              // Set when on-disk file is modified from
DB
PKG_FILE = 0x0020,                  // File/dir is part of the original
package (as opposed to files newly-added to sandbox during package use)
DEPLOY_RAM_MODE = 0x0200,           // Force file deploy (RAM mode)
ALL_FLAGS = ISFILE | DELETED | DEPLOY_UPON_PRELOAD | DISCONNECTED | PKG_FILE
| DEPLOY_RAM_MODE
}

[Flags]
public enum VIRT_PROCESS_FLAGS
{
VINTEGRATE_PROCESS_ONLY = 1,
VINTEGRATE_RECURSIVE = 2
}

public class VirtPackage
{
public enum APIRET
{
SUCCESS = 0,
FAILURE = 1,
VIRTFILES_DB_ERROR = 2,
VIRTFILES_ZIP_ERROR = 3,
NOT_FOUND = 5,
INVALID_PARAMETER = 6,
FILE_CREATE_ERROR = 7,
PE_RESOURCE_ERROR = 8,
MEMORY_ERROR = 9,
COMMIT_ERROR = 10,
VIRTREG_DEPLOY_ERROR = 11,
OUTPUT_ERROR = 12,
INSUFFICIENT_BUFFER = 13,
LOADLIBRARY_ERROR = 14,
VIRTFILES_INI_ERROR = 15,
APP_NOT_DEPLOYED = 16,
INSUFFICIENT_PRIVILEGES = 17,
_32_64_BIT_MISMATCH = 18,
DOTNET_REQUIRED = 19,
CANCELLED = 20,
INJECTION_FAILED = 21,
OLD_VERSION = 22,
PASSWORD_REQUIRED = 23,
PASSWORD_MISMATCH = 24,
INSUFFICIENT_LICENSE = 25,
}

public const int SANDBOXFLAGS_PASSTHROUGH = 1;
public const int SANDBOXFLAGS_COPY_ON_WRITE = 2;
public const int SANDBOXFLAGS_STRICTLY_ISOLATED = 3;

public const int ISOLATIONMODE_CUSTOM = 0;
public const int ISOLATIONMODE_ISOLATED = 1;
public const int ISOLATIONMODE_FULL_ACCESS = 2;
public const int ISOLATIONMODE_DATA = 3;

private const String DLL32 = "PackagerDll.dll";
private const String DLL64 = "PackagerDll64.dll";
public const int MAX_STRING = 64 * 1024;

private IntPtr hPkg;
public bool opened;
public String openedFile;
```

```
APIRET lastError;

private const int MAX_PATH = 260;
public const int MAX_APPID_LENGTH = 128;

public const int LICENSETYPE_PRO = 2;
public const int LICENSETYPE_DEV = 3;

[StructLayout(LayoutKind.Sequential)]
private struct SYSTEMTIME
{
[MarshalAs(UnmanagedType.U2)]
public short Year;
[MarshalAs(UnmanagedType.U2)]
public short Month;
[MarshalAs(UnmanagedType.U2)]
public short DayOfWeek;
[MarshalAs(UnmanagedType.U2)]
public short Day;
[MarshalAs(UnmanagedType.U2)]
public short Hour;
[MarshalAs(UnmanagedType.U2)]
public short Minute;
[MarshalAs(UnmanagedType.U2)]
public short Second;
[MarshalAs(UnmanagedType.U2)]
public short Milliseconds;
}

//
// DLL imports

// PackageOpen
```

```
[DllImport(DLL32, EntryPoint = "PackageOpen", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageOpen64(
String PackageExeFile,
UInt32 Reserved,
ref IntPtr hPkg);
[DllImport(DLL64, EntryPoint = "PackageOpen", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageOpen64(
String PackageExeFile,
UInt64 Reserved,
ref IntPtr hPkg);
private static int PackageOpen(
String PackageExeFile,
UInt32 Reserved,
UInt64 Reserved,
ref IntPtr hPkg)
{
return Is32Bit() ? PackageOpen32(PackageExeFile, Reserved, ref hPkg);
return Is64Bit() ? PackageOpen64(PackageExeFile, Reserved, ref hPkg);
}

// PackageCreate
[DllImport(DLL32, EntryPoint = "PackageCreate", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageCreate32(
String AppID,
String AppVirtDll,
String LoaderExe,
ref IntPtr hPkg);
[DllImport(DLL64, EntryPoint = "PackageCreate", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageCreate64(
String AppID,
String AppVirtDll,
String LoaderExe,
ref IntPtr hPkg);
private static int PackageCreate(
String AppID,
String AppVirtDll,
String LoaderExe,
ref IntPtr hPkg)
{
return Is32Bit() ? PackageCreate32(AppID, AppVirtDll, LoaderExe, ref
hPkg) : PackageCreate64(AppID, AppVirtDll, LoaderExe, ref hPkg);
}

// PackageClose
[DllImport(DLL32, EntryPoint = "PackageClose", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static void PackageClose32(
IntPtr hPkg);
[DllImport(DLL64, EntryPoint = "PackageClose", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static void PackageClose64(
IntPtr hPkg);
private static void PackageClose(
IntPtr hPkg)
{
if (Is32Bit())
PackageClose32(hPkg);
Else
PackageClose64(hPkg);
}

// PackageSave
[DllImport(DLL32, EntryPoint = "PackageSave", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
```

```
private extern static int PackageSave32(
IntPtr hPkg,
String OutFileName);
[DllImport(DLL64, EntryPoint = "PackageSave", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageSave64(
IntPtr hPkg,
String OutFileName);
private static int PackageSave(
IntPtr hPkg,
String OutFileName)
{
return Is32Bit() ? PackageSave32(hPkg, OutFileName) : PackageSave64(hPkg, OutFileName);
}

// PackageGetProperty
[DllImport(DLL32, EntryPoint = "PackageGetProperty", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageGetProperty32(
IntPtr hPkg,
String Name,
StringBuilder Value,
UInt32 ValueLen);
[DllImport(DLL64, EntryPoint = "PackageGetProperty", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageGetProperty64(
IntPtr hPkg,
String Name,
StringBuilder Value,
UInt32 ValueLen);
private static int PackageGetProperty(
IntPtr hPkg,
String Name,
StringBuilder Value,
UInt32 ValueLen)
{
return Is32Bit() ? PackageGetProperty32(hPkg, Name, Value, ValueLen) :
PackageGetProperty64(hPkg, Name, Value, ValueLen);
}

// PackageSetProperty
[DllImport(DLL32, EntryPoint = "PackageSetProperty", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageSetProperty32(
IntPtr hPkg,
String Name,
String Value);
[DllImport(DLL64, EntryPoint = "PackageSetProperty", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageSetProperty64(
IntPtr hPkg,
String Name,
String Value);
private static int PackageSetProperty(
IntPtr hPkg,
String Name,
String Value)
{
return Is32Bit() ? PackageSetProperty32(hPkg, Name, Value) :
PackageSetProperty64(hPkg, Name, Value);
}

// PackageSetIconFile
[DllImport(DLL32, EntryPoint = "PackageSetIconFile", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageSetIconFile32(
IntPtr hPkg,
String FileName);
```

```
[DllImport(DLL64, EntryPoint = "PackageSetIconFile", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageSetIconFile64(
IntPtr hPkg,
String FileName);
private static int PackageSetIconFile(
IntPtr hPkg,
String FileName)
{
return Is32Bit() ? PackageSetIconFile32(hPkg, FileName) :
PackageSetIconFile64(hPkg, FileName);
}

// PackageSetProtection
[DllImport(DLL32, EntryPoint = "PackageSetProtection", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageSetProtection32(
IntPtr hPkg,
[MarshalAs(UnmanagedType.LPStr)] String Password,
Int32 ProtectedActions,
String RequireCertificate);
[DllImport(DLL64, EntryPoint = "PackageSetProtection", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int PackageSetProtection64(
IntPtr hPkg,
[MarshalAs(UnmanagedType.LPStr)] String Password,
Int32 ProtectedActions,
String RequireCertificate);
private static int PackageSetProtection(
IntPtr hPkg,
[MarshalAs(UnmanagedType.LPStr)] String Password,
Int32 ProtectedActions,
String RequireCertificate)
{
        return Is32Bit() ? PackageSetProtection32(hPkg, Password,
    ProtectedActions, RequireCertificate) : PackageSetProtection64(hPkg,
    Password, ProtectedActions, RequireCertificate);
        }

        //
        // VirtFs imports
        private delegate bool VIRTFS_ENUM_CALLBACK(
        ref Object Data,
        [MarshalAs(UnmanagedType.LPWStr)] String FileName,
        UInt32 FileFlags,
        Uint64 Fileflags,
        UInt64 CreationTime,
        UInt64 LastAccessTime,
        UInt64 LastWriteTime,
        UInt64 ChangeTime,
        UInt64 EndOfFile,
        UInt32 FileAttributes
        UInt64 FileAttributes);

        [DllImport(DLL32, EntryPoint = "VirtFsEnum", CharSet =
    CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
        private extern static int VirtFsEnum32(
        IntPtr hPkg,
        VIRTFS_ENUM_CALLBACK Callback,
        ref Object Data);
        [DllImport(DLL64, EntryPoint = "VirtFsEnum", CharSet =
    CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
        private extern static int VirtFsEnum64(
        IntPtr hPkg,
        VIRTFS_ENUM_CALLBACK Callback,
        ref Object Data);
        private static int VirtFsEnum(
        IntPtr hPkg,
        VIRTFS_ENUM_CALLBACK Callback,
        ref Object Data)
        {
        return Is32Bit() ? VirtFsEnum32(hPkg, Callback, ref Data) :
    VirtFsEnum64(hPkg, Callback, ref Data);
        }

        // VirtFsAdd
        [DllImport(DLL32, EntryPoint = "VirtFsAdd", CharSet = CharSet.Unicode,
    CallingConvention = CallingConvention.StdCall)]
        private extern static int VirtFsAdd32(
        IntPtr hPkg,
        String SrcFileName,
        String DestFileName,
        bool bVariablizeName);
        [DllImport(DLL64, EntryPoint = "VirtFsAdd", CharSet = CharSet.Unicode,
    CallingConvention = CallingConvention.StdCall)]
        private extern static int VirtFsAdd64(
        IntPtr hPkg,
        String SrcFileName,
        String DestFileName,
        bool bVariablizeName);
        private static int VirtFsAdd(
        IntPtr hPkg,
        String SrcFileName,
        String DestFileName,
        bool bVariablizeName)
        {
        return Is32Bit() ? VirtFsAdd32(hPkg, SrcFileName, DestFileName,
    bVariablizeName) : VirtFsAdd64(hPkg, SrcFileName, DestFileName,
    bVariablizeName);
        }

        // VirtFsAddEx
        [DllImport(DLL32, EntryPoint = "VirtFsAddEx", CharSet =
    CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
```

```
        private extern static int VirtFsAddEx32(
        IntPtr hPkg,
        String SrcFileName,
        String DestFileName,
        bool bVariablizeName,
        UInt32 FileFlags);
        [DllImport(DLL64, EntryPoint = "VirtFsAddEx", CharSet =
    CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
        private extern static int VirtFsAddEx64(
        IntPtr hPkg,
        String SrcFileName,
        String DestFileName,
        bool bVariablizeName,
        UInt64 FileFlags);
        private static int VirtFsAddEx(
        IntPtr hPkg,
        String SrcFileName,
        String DestFileName,
        bool bVariablizeName,
        UInt64 FileFlags)
        {
        return Is32Bit() ? VirtFsAddEx32(hPkg, SrcFileName, DestFileName,
    bVariablizeName, FileFlags) : VirtFsAddEx64(hPkg, SrcFileName, DestFileName,
    bVariablizeName, FileFlags);
        }

        // VirtFsAddEmptyDir
        [DllImport(DLL32, EntryPoint = "VirtFsAddEmptyDir", CharSet =
    CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
        private extern static int VirtFsAddEmptyDir32(
        IntPtr hPkg,
        String DirName,
        bool bVariablizeName);
        [DllImport(DLL64, EntryPoint = "VirtFsAddEmptyDir", CharSet =
    CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
```

```
        private extern static int VirtFsAddEmptyDir64(
        IntPtr hPkg,
        String DirName,
        bool bVariablizeName);
        private static int VirtFsAddEmptyDir(
        IntPtr hPkg,
        String DirName,
        bool bVariablizeName)
        {
        return Is32Bit() ? VirtFsAddEmptyDir32(hPkg, DirName, bVariablizeName)
   : VirtFsAddEmptyDir64(hPkg, DirName, bVariablizeName);
        }

        // VirtFsExtract
        [DllImport(DLL32, EntryPoint = "VirtFsExtract", CharSet =
   CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
        private extern static int VirtFsExtract32(
        IntPtr hPkg,
        String FileName,
        String TargetDir);
        [DllImport(DLL64, EntryPoint = "VirtFsExtract", CharSet =
   CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
        private extern static int VirtFsExtract64(
        IntPtr hPkg,
        String FileName,
        String TargetDir);
        private static int VirtFsExtract(
        IntPtr hPkg,
        String FileName,
        String TargetDir)
        {
        return Is32Bit() ? VirtFsExtract32(hPkg, FileName, TargetDir) :
   VirtFsExtract64(hPkg, FileName, TargetDir);
        }
```

```

    // VirtFsDelete
    [DllImport(DLL32, EntryPoint = "VirtFsDelete", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtFsDelete32(
    IntPtr hPkg,
    String FileName);
    [DllImport(DLL64, EntryPoint = "VirtFsDelete", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtFsDelete64(
    IntPtr hPkg,
    String FileName);
    private static int VirtFsDelete(
    IntPtr hPkg,
    String FileName)
    {
    return Is32Bit() ? VirtFsDelete32(hPkg, FileName) :
VirtFsDelete64(hPkg, FileName);
    }

    // VirtFsSetFileStreaming
    [DllImport(DLL32, EntryPoint = "VirtFsSetFileStreaming", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtFsSetFileStreaming32(
    IntPtr hPkg,
    String FileName);
    [DllImport(DLL64, EntryPoint = "VirtFsSetFileStreaming", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtFsSetFileStreaming64(
    IntPtr hPkg,
    String FileName);
    public static int VirtFsSetFileStreaming(
    IntPtr hPkg,
    String FileName)
    {
    return Is32Bit() ? VirtFsSetFileStreaming32(hPkg, FileName) :
 VirtFsSetFileStreaming64(hPkg, FileName);
    }


    //
    // VirtReg imports

    // VirtRegGetWorkKey
    [DllImport(DLL32, EntryPoint = "VirtRegGetWorkKey", CharSet =
 CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtRegGetWorkKey32(
    IntPtr hPkg,
    StringBuilder WorkKey,
    UInt32 WorkKeyLen);
    [DllImport(DLL32, EntryPoint = "VirtRegGetWorkKey", CharSet =
 CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtRegGetWorkKey64(
    IntPtr hPkg,
    StringBuilder WorkKey,
    UInt32 WorkKeyLen);
    private static int VirtRegGetWorkKey(
    IntPtr hPkg,
    StringBuilder WorkKey,
    UInt32 WorkKeyLen)
    {
    return Is32Bit() ? VirtRegGetWorkKey32(hPkg, WorkKey, WorkKeyLen) :
 VirtRegGetWorkKey64(hPkg, WorkKey, WorkKeyLen);
    }

    // VirtRegGetWorkKeyEx
    [DllImport(DLL32, EntryPoint = "VirtRegGetWorkKeyEx", CharSet =
 CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtRegGetWorkKeyEx32(
    IntPtr hPkg,
    StringBuilder WorkKey,
    UInt32 WorkKeyLen,
    SafeWaitHandle hAbortEvent);
    [DllImport(DLL64, EntryPoint = "VirtRegGetWorkKeyEx", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtRegGetWorkKeyEx64(
    IntPtr hPkg,
    StringBuilder WorkKey,
    UInt32 WorkKeyLen,
    SafeWaitHandle hAbortEvent);
    private static int VirtRegGetWorkKeyEx(
    IntPtr hPkg,
    StringBuilder WorkKey,
    UInt32 WorkKeyLen,
    SafeWaitHandle hAbortEvent)
    {
    return Is32Bit() ? VirtRegGetWorkKeyEx32(hPkg, WorkKey, WorkKeyLen, hAbortEvent) : VirtRegGetWorkKeyEx64(hPkg, WorkKey, WorkKeyLen, hAbortEvent);
    }

    // VirtRegSaveWorkKey
    [DllImport(DLL32, EntryPoint = "VirtRegSaveWorkKey", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtRegSaveWorkKey32(
    IntPtr hPkg);
    [DllImport(DLL64, EntryPoint = "VirtRegSaveWorkKey", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtRegSaveWorkKey64(
    IntPtr hPkg);
    private static int VirtRegSaveWorkKey(
    IntPtr hPkg)
    {
    return Is32Bit() ? VirtRegSaveWorkKey32(hPkg) : VirtRegSaveWorkKey64(hPkg);
```

```
}


//
// Sandbox imports

// SandboxGetRegistryFlags
[DllImport(DLL32, EntryPoint = "SandboxGetRegistryFlags", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int SandboxGetRegistryFlags32(
IntPtr hPkg,
String Path,
bool bVariablizeName,
ref UInt32 SandboxFlags);
[DllImport(DLL64, EntryPoint = "SandboxGetRegistryFlags", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int SandboxGetRegistryFlags64(
IntPtr hPkg,
String Path,
bool bVariablizeName,
ref UInt32 SandboxFlags);
private static int SandboxGetRegistryFlags(
IntPtr hPkg,
String Path,
bool bVariablizeName,
ref UInt64 SandboxFlags)
{
return Is32Bit() ? SandboxGetRegistryFlags32(hPkg, Path,
bVariablizeName, ref SandboxFlags) : SandboxGetRegistryFlags64(hPkg, Path,
bVariablizeName, ref SandboxFlags);
}

// SandboxGetFileFlags
[DllImport(DLL32, EntryPoint = "SandboxGetFileFlags", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
```

```
private extern static int SandboxGetFileFlags32(
IntPtr hPkg,
String Path,
bool bVariablizeName,
ref UInt32 SandboxFlags);
[DllImport(DLL64, EntryPoint = "SandboxGetFileFlags", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int SandboxGetFileFlags64(
IntPtr hPkg,
String Path,
bool bVariablizeName,
ref UInt64 SandboxFlags);
private static int SandboxGetFileFlags(
IntPtr hPkg,
String Path,
bool bVariablizeName,
ref UInt32 SandboxFlags)
{
return Is32Bit() ? SandboxGetFileFlags32(hPkg, Path, bVariablizeName,
ref SandboxFlags) : SandboxGetFileFlags64(hPkg, Path, bVariablizeName, ref
SandboxFlags);
}

// SandboxSetRegistryFlags
[DllImport(DLL32, EntryPoint = "SandboxSetRegistryFlags", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int SandboxSetRegistryFlags32(
IntPtr hPkg,
String Path,
bool bVariablizeName,
UInt64 SandboxFlags);
[DllImport(DLL64, EntryPoint = "SandboxSetRegistryFlags", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int SandboxSetRegistryFlags64(
IntPtr hPkg,
String Path,
bool bVariablizeName,
UInt64 SandboxFlags);
private static int SandboxSetRegistryFlags(
IntPtr hPkg,
String Path,
bool bVariablizeName,
UInt32 SandboxFlags)
{
return Is32Bit() ? SandboxSetRegistryFlags32(hPkg, Path,
bVariablizeName, SandboxFlags) : SandboxSetRegistryFlags64(hPkg, Path,
bVariablizeName, SandboxFlags);
}

// SandboxSetFileFlags
[DllImport(DLL32, EntryPoint = "SandboxSetFileFlags", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int SandboxSetFileFlags32(
IntPtr hPkg,
String Path,
bool bVariablizeName,
UInt32 SandboxFlags);
[DllImport(DLL64, EntryPoint = "SandboxSetFileFlags", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int SandboxSetFileFlags64(
IntPtr hPkg,
String Path,
bool bVariablizeName,
UInt32 SandboxFlags);
private static int SandboxSetFileFlags(
IntPtr hPkg,
String Path,
bool bVariablizeName,
UInt64 SandboxFlags)
```

```
    {
    return Is32Bit() ? SandboxSetFileFlags32(hPkg, Path, bVariablizeName, SandboxFlags) : SandboxSetFileFlags64(hPkg, Path, bVariablizeName, SandboxFlags);
    }

    // VirtFsGetFileFlags
    [DllImport(DLL32, EntryPoint = "VirtFsGetFileFlags", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtFsGetFileFlags32(
    IntPtr hPkg,
    String Path,
    ref UInt32 FileFlags);
    [DllImport(DLL64, EntryPoint = "VirtFsGetFileFlags", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtFsGetFileFlags64(
    IntPtr hPkg,
    String Path,
    ref UInt64 FileFlags);
    private static int VirtFsGetFileFlags(
    IntPtr hPkg,
    String Path,
    ref UInt32 FileFlags)
    {
    return Is32Bit() ? VirtFsGetFileFlags32(hPkg, Path, ref FileFlags) : VirtFsGetFileFlags64(hPkg, Path, ref FileFlags);
    }

    // VirtFsSetFileFlags
    [DllImport(DLL32, EntryPoint = "VirtFsSetFileFlags", CharSet = CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int VirtFsSetFileFlags32(
    IntPtr hPkg,
    String Path,
    UInt32 FileFlags);
```

```
[DllImport(DLL64, EntryPoint = "VirtFsSetFileFlags", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int VirtFsSetFileFlags64(
IntPtr hPkg,
String Path,
UInt64 FileFlags);
private static int VirtFsSetFileFlags(
IntPtr hPkg,
String Path,
UInt32 FileFlags)
{
return Is32Bit() ? VirtFsSetFileFlags32(hPkg, Path, FileFlags) :
VirtFsSetFileFlags64(hPkg, Path, FileFlags);
}


//
// 'Quick' functions (do not require package to be opened, can be
called on closed package files)

// QuickReadIni
[DllImport(DLL32, EntryPoint = "QuickReadIni", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int QuickReadIni32(
String PackageExeFile,
StringBuilder IniBuf,
UInt32 IniBufLen);
[DllImport(DLL64, EntryPoint = "QuickReadIni", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int QuickReadIni64(
String PackageExeFile,
StringBuilder IniBuf,
UInt64 IniBufLen);
public static int QuickReadIni(
String PackageExeFile,
StringBuilder IniBuf,
UInt32 IniBufLen)
{
return Is32Bit() ? QuickReadIni32(PackageExeFile, IniBuf, IniBufLen) :
QuickReadIni64(PackageExeFile, IniBuf, IniBufLen);
}

// QuickReadIniValues (wrapper)
public static Hashtable QuickReadIniValues(string PacakgeExeFile)
{
StringBuilder sb = new StringBuilder(16384);
VirtPackage.QuickReadIni(PacakgeExeFile, sb, 16384);
return VirtPackage.ReadIniSettingsBuf(sb.ToString());
}

// QuickBuildAppendiceIndex (reserved for internal use)
[DllImport(DLL32, EntryPoint = "QuickBuildAppendiceIndex", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int QuickBuildAppendiceIndex32(
byte[] pLastFileBytes,
UInt32 cbLastFileBytes,
byte[] pNewIndex,
ref UInt32 pcbNewIndex,
byte[] pSandboxCfg,
UInt32 cbSandboxCfg,
byte[] pIniBuf,
UInt32 cbIniBuf);
[DllImport(DLL64, EntryPoint = "QuickBuildAppendiceIndex", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int QuickBuildAppendiceIndex64(
byte[] pLastFileBytes,
UInt32 cbLastFileBytes,
byte[] pNewIndex,
ref UInt32 pcbNewIndex,
byte[] pSandboxCfg,
UInt32 cbSandboxCfg,
byte[] pIniBuf,
UInt32 cbIniBuf);
public static int QuickBuildAppendiceIndex(
byte[] pLastFileBytes,
UInt32 cbLastFileBytes,
byte[] pNewIndex,
ref UInt32 pcbNewIndex,
byte[] pSandboxCfg,
UInt32 cbSandboxCfg,
byte[] pIniBuf,
UInt32 cbIniBuf)
{
int ret = Is32Bit()
? QuickBuildAppendiceIndex32(pLastFileBytes, cbLastFileBytes,
pNewIndex, ref pcbNewIndex, pSandboxCfg, cbSandboxCfg, pIniBuf, cbIniBuf)
: QuickBuildAppendiceIndex64(pLastFileBytes, cbLastFileBytes,
pNewIndex, ref pcbNewIndex, pSandboxCfg, cbSandboxCfg, pIniBuf, cbIniBuf);
return ret;
}

// QuickExtractAppendiceIndex (reserved for internal use)
[DllImport(DLL32, EntryPoint = "QuickExtractAppendiceIndex", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int QuickExtractAppendiceIndex32(
String FileName,
String OutputFile);
[DllImport(DLL64, EntryPoint = "QuickExtractAppendiceIndex", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int QuickExtractAppendiceIndex64(
String FileName,
String OutputFile);
public static int QuickExtractAppendiceIndex(
String FileName,
String OutputFile)
{
return Is32Bit() ? QuickExtractAppendiceIndex32(FileName, OutputFile)
 : QuickExtractAppendiceIndex64(FileName, OutputFile);
}
static public byte[] QuickExtractAppendiceIndexBytes(
String FileName)
{
string outputFile = Path.GetTempFileName();
try { File.Delete(outputFile); }
catch { }
if (VirtPackage.QuickExtractAppendiceIndex(FileName, outputFile) !=
 (int)VirtPackage.APIRET.SUCCESS)
return null;
byte[] ret = File.ReadAllBytes(outputFile);
try { File.Delete(outputFile); }
catch { }
return ret;
}

// LicDataLoadFromFile (reserved for internal use)
[DllImport(DLL32, EntryPoint = "LicDataLoadFromFile", CharSet =
 CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int LicDataLoadFromFile32(
String FileName);
[DllImport(DLL64, EntryPoint = "LicDataLoadFromFile", CharSet =
 CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int LicDataLoadFromFile64(
String FileName);
public static int LicDataLoadFromFile(
String FileName)
{
int ret = Is32Bit()
```

```
    ? LicDataLoadFromFile32(FileName)
    : LicDataLoadFromFile64(FileName);
    return ret;
    }


    //
    // DeployedApp imports

    private delegate bool DEPLOYEDAPP_ENUM_CALLBACK(
    ref Object Data,
    [MarshalAs(UnmanagedType.LPWStr)] String AppID);

    // DeployedAppEnum
    [DllImport(DLL32, EntryPoint = "DeployedAppEnum", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int DeployedAppEnum32(
    DEPLOYEDAPP_ENUM_CALLBACK Callback,
    ref Object Data);
    [DllImport(DLL64, EntryPoint = "DeployedAppEnum", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int DeployedAppEnum64(
    DEPLOYEDAPP_ENUM_CALLBACK Callback,
    ref Object Data);
    private static int DeployedAppEnum(
    DEPLOYEDAPP_ENUM_CALLBACK Callback,
    ref Object Data)
    {
    return Is32Bit() ? DeployedAppEnum32(Callback, ref Data) :
DeployedAppEnum64(Callback, ref Data);
    }

    // DeployedAppGetDir
```

```
[DllImport(DLL32, EntryPoint = "DeployedAppGetDir", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int DeployedAppGetDir32(
String AppID,
StringBuilder BaseDirName,
UInt32 BaseDirNameLen);
[DllImport(DLL64, EntryPoint = "DeployedAppGetDir", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int DeployedAppGetDir64(
String AppID,
StringBuilder BaseDirName,
UInt32 BaseDirNameLen);
private static int DeployedAppGetDir(
String AppID,
StringBuilder BaseDirName,
UInt32 BaseDirNameLen)
{
return Is32Bit() ? DeployedAppGetDir32(AppID, BaseDirName,
BaseDirNameLen) : DeployedAppGetDir64(AppID, BaseDirName, BaseDirNameLen);
}

[StructLayout(LayoutKind.Sequential)]
public struct VIRT_PROCESS
{
public UInt32 PID;
public UInt32 Flags;
}


//
// RunningApp imports

// RunningAppEnum
```

```
[DllImport(DLL32, EntryPoint = "RunningAppEnum", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int RunningAppEnum32(
RUNNINGAPP_ENUM_CALLBACK Callback,
ref Object Data);
[DllImport(DLL64, EntryPoint = "RunningAppEnum", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int RunningAppEnum64(
RUNNINGAPP_ENUM_CALLBACK Callback,
ref Object Data);
private static int RunningAppEnum(
RUNNINGAPP_ENUM_CALLBACK Callback,
ref Object Data)
{
return Is32Bit() ? RunningAppEnum32(Callback, ref Data) :
RunningAppEnum64(Callback, ref Data);
}

// RunningAppEnumKeepAlive
[DllImport(DLL32, EntryPoint = "RunningAppEnumKeepAlive", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int RunningAppEnumKeepAlive32(
ref IntPtr Context,
RUNNINGAPP_ENUM_CALLBACK Callback,
ref Object Data);
[DllImport(DLL64, EntryPoint = "RunningAppEnumKeepAlive", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
private extern static int RunningAppEnumKeepAlive64(
ref IntPtr Context,
RUNNINGAPP_ENUM_CALLBACK Callback,
ref Object Data);
private static int RunningAppEnumKeepAlive(
ref IntPtr Context,
RUNNINGAPP_ENUM_CALLBACK Callback,
ref Object Data)
```

```
    {
    return Is32Bit() ? RunningAppEnumKeepAlive32(ref Context, Callback,
ref Data) : RunningAppEnumKeepAlive64(ref Context, Callback, ref Data);
    }

    // RunningAppEnumKeepAliveFree
    [DllImport(DLL32, EntryPoint = "RunningAppEnumKeepAliveFree", CharSet
= CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int RunningAppEnumKeepAliveFree32(
    IntPtr Context);
    [DllImport(DLL64, EntryPoint = "RunningAppEnumKeepAliveFree", CharSet
= CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int RunningAppEnumKeepAliveFree64(
    IntPtr Context);
    private static int RunningAppEnumKeepAliveFree(
    IntPtr Context)
    {
    return Is32Bit() ? RunningAppEnumKeepAliveFree32(Context) :
RunningAppEnumKeepAliveFree64(Context);
    #debugging app
    [DllImport(DLL32, EntryPoint = "debuggingapp", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int debuggingapp32(
    ref IntPtr Context,
    debuggingapp_ENUM_CALLBACK Callback,
    ref Object Data);
    [DllImport(DLL64, EntryPoint = "debuggingapp", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int debuggingapp64(
    ref IntPtr Context,
    debuggingapp _ENUM_CALLBACK Callback,
    ref Object Data);
    private static int debuggingapp(
    ref IntPtr Context,
    debuggingapp _ENUM_CALLBACK Callback,
    ref Object Data)
    {
    return Is32Bit() ? debuggingapp32(ref Context, Callback, ref Data) :
debuggingapp64(ref Context, Callback, ref Data);
    }

    [StructLayout(LayoutKind.Sequential)]
    private struct RUNNING_APP
    {
    public UInt32 Version;
    public UInt32 SerialId;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = MAX_PATH * 2 * 2)]
    public char[] CarrierExeName;
    public UInt32 CarrierPID;
    public UInt32 StartTickTime;
    public UInt32 SyncStreamingDuration;
    public UInt32 TotalPIDs;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 128)]
    public VIRT_PROCESS[] Processes;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = MAX_APPID_LENGTH *
2)]
    public char[] AppID;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = MAX_APPID_LENGTH *
2)]
    public char[] FriendlyName;
    }
    public class RunningApp
    {
    public String AppID;
    public List<VIRT_PROCESS> Processes;
    public String CarrierExeName;
    public UInt32 SerialId;
    public UInt32 CarrierPID;
    public UInt32 StartTickTime;
```

```
public String FriendlyName;

public int GetVintegrationMode(out bool anyVirtProcessRunning)
{
int vintegration = 0;
anyVirtProcessRunning = false;
foreach (VirtPackage.VIRT_PROCESS process in this.Processes)
{
if (((int)process.Flags &
((int)VIRT_PROCESS_FLAGS.VINTEGRATE_PROCESS_ONLY |
(int)VIRT_PROCESS_FLAGS.VINTEGRATE_RECURSIVE)) != 0)
vintegration |= (int)process.Flags;
else
anyVirtProcessRunning = true;
}
return vintegration;
}

public static RunningApp FromAppID(String AppID)
{
List<RunningApp> runningApps = GetRunningApps();
foreach (RunningApp runningApp in runningApps)
{
if (runningApp.AppID.Equals(AppID,
StringComparison.InvariantCultureIgnoreCase))
return runningApp;
}
return null;
}
}
private delegate bool RUNNINGAPP_ENUM_CALLBACK(
ref Object Data,
ref RUNNING_APP RunningApp);

```

```
    //
    // Utils imports

    // UtilsGenericToLocalPath
    [DllImport(DLL32, EntryPoint = "UtilsGenericToLocalPath", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int UtilsGenericToLocalPath32(
    String GenericPath,
    StringBuilder LocalPath,
    UInt32 LocalPathLen);
    [DllImport(DLL64, EntryPoint = "UtilsGenericToLocalPath", CharSet =
CharSet.Unicode, CallingConvention = CallingConvention.StdCall)]
    private extern static int UtilsGenericToLocalPath64(
    String GenericPath,
    StringBuilder LocalPath,
    UInt32 LocalPathLen);
    public static int UtilsGenericToLocalPath(
    String GenericPath,
    StringBuilder LocalPath,
    UInt32 LocalPathLen)
    {
    return Is32Bit() ? UtilsGenericToLocalPath32(GenericPath, LocalPath,
LocalPathLen) : UtilsGenericToLocalPath64(GenericPath, LocalPath,
LocalPathLen);
    }
    public static string GenericToLocalDir(string GenericPath)
    {
    //if (string.IsNullOrEmpty(genericPath))
    StringBuilder sbValue = new StringBuilder(VirtPackage.MAX_STRING);
    VirtPackage.APIRET Ret =
(VirtPackage.APIRET)VirtPackage.UtilsGenericToLocalPath(GenericPath,
sbValue, VirtPackage.MAX_STRING);
    return sbValue.ToString();
    }
```

```
//
// .NET wrapper
public VirtPackage()
{
opened = false;
openedFile = "";
lastError = APIRET.SUCCESS;
//private const String DLLNAME = "PackagerDll.dll";
}

~VirtPackage()
{
Close();
}

public APIRET GetLastError()
{
return lastError;
}

public void ResetLastError()
{
lastError = APIRET.SUCCESS;
}

static public void ApiRetStr(APIRET apiRet, out string name, out
string description)
{
switch (apiRet)
{
case APIRET.SUCCESS:                              name =
"APIRET_SUCCESS"; description = "Success"; break;
```

```
    case APIRET.FAILURE:                          name = "APIRET_FAILURE"; description = "Failure"; break;
    case APIRET.VIRTFILES_DB_ERROR:        name = "APIRET_VIRTFILES_DB_ERROR"; description = "Virtual filesystem database error"; break;
    case APIRET.VIRTFILES_ZIP_ERROR: name = "APIRET_VIRTFILES_ZIP_ERROR"; description = "ZIP error"; break;
    case APIRET.NOT_FOUND:                        name = "APIRET_NOT_FOUND"; description = "File / resource not found"; break;
    case APIRET.out of file descriptors:                    name = "ran out of file descriptors"; description = "all the file descriports are used"; break;
    case APIRET.INVALID_PARAMETER:         name = "APIRET_INVALID_PARAMETER"; description = "Invalid parameter"; break;
    case APIRET.FILE_CREATE_ERROR:         name = "APIRET_FILE_CREATE_ERROR"; description = "File creation error"; break;
    case APIRET.PE_RESOURCE_ERROR:         name = "APIRET_PE_RESOURCE_ERROR"; description = "Virtual package resource error"; break;
    case APIRET.MEMORY_ERROR:              name = "APIRET_MEMORY_ERROR"; description = "Memory error"; break;
    case APIRET.COMMIT_ERROR:              name = "APIRET_COMMIT_ERROR"; description = "Error committing virtual package changes"; break;
    case APIRET.VIRTREG_DEPLOY_ERROR:name = "APIRET_VIRTREG_DEPLOY_ERROR"; description = "Error deploying virtual registry"; break;
    case APIRET.OUTPUT_ERROR:              name = "APIRET_OUTPUT_ERROR"; description = "Output error"; break;
    case APIRET.INSUFFICIENT_BUFFER: name = "APIRET_INSUFFICIENT_BUFFER"; description = "Insufficient buffer size"; break;
    case APIRET.LOADLIBRARY_ERROR:         name = "APIRET_LOADLIBRARY_ERROR"; description = "LoadLibrary error"; break;
    case APIRET.VIRTFILES_INI_ERROR: name = "APIRET_VIRTFILES_INI_ERROR"; description = "Virtual package INI error"; break;
    case APIRET.APP_NOT_DEPLOYED:          name = "APIRET_APP_NOT_DEPLOYED"; description = "Application not depliyed"; break;
    case APIRET.INSUFFICIENT_PRIVILEGES:name = "APIRET_INSUFFICIENT_PRIVILEGES"; description = "Insufficient user permissions"; break;
```

```
case APIRET._32_64_BIT_MISMATCH: name = "APIRET_32_64_BIT_MISMATCH"; description = "32/64 bit mismatch"; break;
case APIRET.DOTNET_REQUIRED:            name = "APIRET_DOTNET_REQUIRED"; description = ".NET framework required"; break;
case APIRET.CANCELLED:                            name = "APIRET_CANCELLED"; description = "Operation cancelled"; break;
case APIRET.INJECTION_FAILED:           name = "APIRET_INJECTION_FAILED"; description = "Injection failed"; break;
case APIRET.OLD_VERSION:                name = "APIRET_OLD_VERSION"; description = "Old package version"; break;
case APIRET.PASSWORD_REQUIRED:          name = "APIRET_PASSWORD_REQUIRED"; description = "Password required"; break;
case APIRET.PASSWORD_MISMATCH:          name = "APIRET_PASSWORD_MISMATCH"; description = "Password mismatch"; break;
case APIRET.INSUFFICIENT_LICENSE:name = "APIRET_INSUFFICIENT_LICENSE"; description = "Insufficient license"; break;
case APIRET.no processors:                        name = "all processors are busy "; description = "cpu processors are busy"; break;
default:                                                    name = String.Format("{0}", (int)apiRet); description = "Unknown error " + ((int)apiRet).ToString(); break;
}
}

static public bool Is32Bit()
{
return (IntPtr.Size == 4);
static public bool Is64Bit()
{
return (IntPtr.Size == 4);
}
if (opened)
{
PackageClose(hPkg);
opened = false;
openedFile = "";
```

```
}
return true;
}

public bool Save(String FileName)
{
APIRET apiRet;
return SaveEx(FileName, out apiRet);
}

public bool SaveEx(String FileName, out APIRET apiRet)
{
int Ret = PackageSave(hPkg, FileName);
apiRet = (APIRET)Ret;
lastError = (apiRet != APIRET.SUCCESS ? apiRet : lastError);
if (apiRet == APIRET.SUCCESS)
return true;
else
return false;
}

public bool Open(string PackageExeFile)
{
APIRET apiRet;
return Open(PackageExeFile, out apiRet);
}

public bool Open(String PackageExeFile, out APIRET apiRet)
{
apiRet = (APIRET)PackageOpen(PackageExeFile, 0, ref hPkg);
lastError = (apiRet != APIRET.SUCCESS ? apiRet : lastError);
if (apiRet == APIRET.SUCCESS)
{
opened = true;
```

```
int passwordPos = PackageExeFile.IndexOf('|');
if (passwordPos == -1)
openedFile = PackageExeFile;
else
openedFile = PackageExeFile.Substring(0, passwordPos);
return true;
}
Else
return false;
}

public bool Create(String AppID, String AppVirtDll, String LoaderExe)
{
APIRET Ret;
return Create(AppID, AppVirtDll, LoaderExe, out Ret);
}

public bool Create(String AppID, String AppVirtDll, String LoaderExe,
out APIRET Ret)
{
Ret = (APIRET)PackageCreate(AppID, AppVirtDll, LoaderExe, ref hPkg);
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
if (Ret == APIRET.SUCCESS)
{
opened = true;
// Note: openedFile remains empty
return true;
}
Else
public bool set(String AppID, String AppVirtDll, String LoaderExe)
{
APIRET Ret;
return set(AppID, AppVirtDll, LoaderExe, out Ret);
}
```

```
public bool set(String AppID, String AppVirtDll, String LoaderExe, out
APIRET Ret)
{
Ret = (APIRET)PackageCreate(AppID, AppVirtDll, LoaderExe, ref hPkg);
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
if (Ret == APIRET.SUCCESS)
{
opened = true;
// Note: openedFile remains empty
return true;
}
Else
return false;
}
else if (Ret == APIRET.NOT_FOUND)
return false;
else
return false;
}

public String GetProperty(String Name)
{
String Value = "";
if (GetProperty(Name, ref Value))
return (Value);
else
return ("");
}

public bool SetProperty(String Name, String Value)
{
APIRET Ret;
return SetProperty(Name, Value, out Ret);
```

```
}

public bool SetProperty(String Name, String Value, out APIRET Ret)
{
Ret = (APIRET)PackageSetProperty(hPkg, Name, Value);
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
if (Ret == APIRET.SUCCESS)
return true;
else if (Ret == APIRET.FILE_CREATE_ERROR)
return false;
else
return false;
}

public bool SetProtection(String Password, int ProtectedActions,
String RequireCertificate)
{
APIRET Ret;
return SetProtection(Password, ProtectedActions, RequireCertificate,
out Ret);
}

public bool SetProtection(String Password, int ProtectedActions,
String RequireCertificate, out APIRET Ret)
{
Ret = (APIRET)PackageSetProtection(hPkg, Password, ProtectedActions,
RequireCertificate);
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
if (Ret == APIRET.SUCCESS)
return true;
else if (Ret == APIRET.FILE_CREATE_ERROR)
return false;
else
return false;
```

```
}

public bool SetIcon(String FileName)
{
APIRET Ret;
return SetIcon(FileName, out Ret);
}

public bool SetIcon(String FileName, out APIRET Ret)
{
Ret = (APIRET)PackageSetIconFile(hPkg, FileName);
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
if (Ret == APIRET.SUCCESS)
return true;
else if (Ret == APIRET.NOT_FOUND)
return false;
else
return false;
}

//
// VirtFs functions
private bool EnumFilesCallback(
ref Object Data,
[MarshalAs(UnmanagedType.LPWStr)] String FileName,
UInt32 FileFlags,
UInt64 Fileflags,
UInt64 CreationTime,
UInt64 LastAccessTime,
UInt64 LastWriteTime,
UInt64 ChangeTime,
UInt64 EndOfFile,
UInt32 FileAttributes)
{
```

```
VirtFsNode virtFsNode = new VirtFsNode();
virtFsNode.FileName = FileName;
virtFsNode.FileFlags = (VIRT_FILE_FLAGS)FileFlags;
virtFsNode.CreationTime = CreationTime;
virtFsNode.LastAccessTime = LastAccessTime;
virtFsNode.LastWriteTime = LastWriteTime;
virtFsNode.ChangeTime = ChangeTime;
virtFsNode.EndOfFile = EndOfFile;
virtFsNode.FileAttributes = FileAttributes;
((List<VirtFsNode>)Data).Add(virtFsNode);
return true;
}

public bool EnumFiles(
ref List<VirtFsNode> VirtFsNodes)
{
VIRTFS_ENUM_CALLBACK Callback = new
VIRTFS_ENUM_CALLBACK(EnumFilesCallback);
Object Data = VirtFsNodes;
VirtFsEnum(hPkg, Callback, ref Data);
return true;
}

public bool AddFile(
String SrcFileName,
String DestFileName,
bool bVariablizeName)
{
VIRT_FILE_FLAGS fileFlags = VIRT_FILE_FLAGS.ISFILE |
VIRT_FILE_FLAGS.DEPLOY_UPON_PRELOAD | VIRT_FILE_FLAGS.PKG_FILE;
return AddFileEx(SrcFileName, DestFileName, bVariablizeName,
fileFlags);
}

```

```
public bool AddFileEx(
String SrcFileName,
String DestFileName,
bool bVariablizeName,
VIRT_FILE_FLAGS fileFlags)
{
APIRET Ret = (APIRET)VirtFsAddEx(hPkg, SrcFileName, DestFileName,
  bVariablizeName, (uint)fileFlags);
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
if (Ret == APIRET.SUCCESS)
return true;
else if (Ret == APIRET.VIRTFILES_DB_ERROR)
return true;
else if (Ret == APIRET.NOT_FOUND)
return false;
else
return false;
}

public bool AddEmptyDir(
String DirName,
bool bVariablizeName)
{
APIRET Ret = (APIRET)VirtFsAddEmptyDir(hPkg, DirName,
  bVariablizeName);
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
if (Ret == APIRET.SUCCESS)
return true;
else if (Ret == APIRET.VIRTFILES_DB_ERROR)
return true;
else
return false;
}

```

```
public bool AddDir(
String SrcFolderName,
String DestFolderName,
bool bVariablizeName)
{
if (!System.IO.Directory.Exists(SrcFolderName))
return false;

AddEmptyDir(DestFolderName, bVariablizeName);

string[] files = System.IO.Directory.GetFiles(SrcFolderName);
foreach (string file in files)
{
if (!AddFile(file, DestFolderName + "\\" +
System.IO.Path.GetFileName(file), bVariablizeName))
return false;
}

string[] subDirs = System.IO.Directory.GetDirectories(SrcFolderName);
foreach (string dir in subDirs)
{
if (!AddDir(dir, DestFolderName + "\\" +
System.IO.Path.GetFileName(dir), bVariablizeName))
return false;
}
return true;
}

public bool ExtractFile(
String FileName,
String TargetDir)
{
APIRET Ret = (APIRET)VirtFsExtract(hPkg, FileName, TargetDir);
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
```

```
if (Ret == APIRET.SUCCESS)
return true;
else if (Ret == APIRET.NOT_FOUND)
return false;
else if (Ret == APIRET.FILE_CREATE_ERROR)
return false;
else
return false;
}

public bool DeleteFile(
String FileName)
{
APIRET Ret = (APIRET)VirtFsDelete(hPkg, FileName);
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
if (Ret == APIRET.SUCCESS)
return true;
else if (Ret == APIRET.NOT_FOUND)
return false;
else
return false;
}

public bool SetFileStreaming(
String FileName)
{
APIRET Ret = (APIRET)VirtFsSetFileStreaming(hPkg, FileName);
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
if (Ret == APIRET.SUCCESS)
return true;
else if (Ret == APIRET.NOT_FOUND)
return false;
else
return false;
```

```
}

//
// VirtReg functions
public RegistryKey GetRegWorkKeyEx(System.Threading.AutoResetEvent
abortEvent)
{
StringBuilder sbWorkKey = new StringBuilder(MAX_STRING);

SafeWaitHandle waitHandle;
if (abortEvent != null)
waitHandle = abortEvent.SafeWaitHandle;
else
waitHandle = new SafeWaitHandle(IntPtr.Zero, true); ;
APIRET Ret = (APIRET)VirtRegGetWorkKeyEx(hPkg, sbWorkKey, MAX_STRING,
waitHandle);
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
if (Ret == APIRET.SUCCESS)
{
RegistryKey key =
Registry.CurrentUser.OpenSubKey(sbWorkKey.ToString(), true);
if (key == null)
{
key = Registry.CurrentUser.CreateSubKey(sbWorkKey.ToString());
}
return (key);
}
else if (Ret == APIRET.INSUFFICIENT_BUFFER)
return null;
else
return null;
}

public RegistryKey GetRegWorkKey()
```

```
{
return GetRegWorkKeyEx(null);
}

[DllImport("kernel32.dll")]
static extern void OutputDebugString(string lpOutputString);

public bool SaveRegWorkKey()
{
APIRET Ret = (APIRET)VirtRegSaveWorkKey(hPkg);
OutputDebugString("SaveRegWorkKey() ret=" + (int)Ret + " LE=" +
Marshal.GetLastWin32Error() + "\n");
lastError = (Ret != APIRET.SUCCESS ? Ret : lastError);
if (Ret == APIRET.SUCCESS)
return true;
else if (Ret == APIRET.INVALID_PARAMETER)
return false;
else
return false;
}

//
// Sandbox functions

// Sandbox Get functions
public UInt32 GetRegistrySandbox(String Path, bool bVariablizeName)
{
UInt32 SandboxFlags = 0;
SandboxGetRegistryFlags(hPkg, Path, bVariablizeName, ref
SandboxFlags);
return SandboxFlags;
}
public UInt32 GetRegistrySandbox(String Path) { return
GetRegistrySandbox(Path, false); }    // Overload
```

```

public UInt32 GetFileSandbox(String Path, bool bVariablizeName)
{
UInt32 SandboxFlags = 0;
SandboxGetFileFlags(hPkg, Path, bVariablizeName, ref SandboxFlags);
return SandboxFlags;
}
public UInt32 GetFileSandbox(String Path) { return
GetFileSandbox(Path, false); }   // Overload

// Sandbox Set functions
public void SetRegistrySandbox(String Path, UInt32 SandboxFlags, bool
bVariablizeName)
{
SandboxSetRegistryFlags(hPkg, Path, bVariablizeName, SandboxFlags);
}
public void SetRegistrySandbox(String Path, UInt32 SandboxFlags) {
SetRegistrySandbox(Path, SandboxFlags, false); }   // Overload

public void SetFileSandbox(String Path, UInt32 SandboxFlags, bool
bVariablizeName)
{
SandboxSetFileFlags(hPkg, Path, bVariablizeName, SandboxFlags);
}
public void SetFileSandbox(String Path, UInt32 SandboxFlags) {
SetFileSandbox(Path, SandboxFlags, false); }   // Overload

public void SetFileFlags(String Path, VIRT_FILE_FLAGS FileFlags)
{
APIRET apiRet = (APIRET)VirtFsSetFileFlags(hPkg, Path,
(UInt32)FileFlags);
lastError = (apiRet != APIRET.SUCCESS ? apiRet : lastError);
}
public VIRT_FILE_FLAGS GetFileFlags(String Path)
{
```

```
UInt32 FileFlags = 0;
APIRET apiRet = (APIRET)VirtFsGetFileFlags(hPkg, Path, ref FileFlags);
lastError = (apiRet != APIRET.SUCCESS ? apiRet : lastError);
return (VIRT_FILE_FLAGS)FileFlags;
}

static String LPWStrToString(char[] lpwstr)
{
String res = "";
for (int i = 0; i < lpwstr.Length; i += 2)
{
if (lpwstr[i] == 0)
break;
res += lpwstr[i];
}
return res;
}

static List<UInt32> ArrayToList(UInt32[] array, UInt32 count)
{
List<UInt32> res = new List<UInt32>();
for (int i = 0; i < count; i++)
res.Add(array[i]);
return res;
}

static List<VIRT_PROCESS> ArrayToList(VIRT_PROCESS[] array, UInt32 count)
{
List<VIRT_PROCESS> res = new List<VIRT_PROCESS>();
for (int i = 0; i < count; i++)
res.Add(array[i]);
return res;
}
```

```
//
// RunningApp functions
static private void Dbg(string msg)
{
OutputDebugString("CameyoMenu: " + msg + "\r\n");
}

static private bool EnumRunningAppsCallback(
ref Object Data,
ref RUNNING_APP RunningAppRaw)
{
Dbg("EnumRunningAppsCallback: in, sizeof(RunningApp)=" +
Marshal.SizeOf(RunningAppRaw));
RunningApp runningApp = new RunningApp()
{
AppID = LPWStrToString(RunningAppRaw.AppID),
CarrierExeName = LPWStrToString(RunningAppRaw.CarrierExeName),
FriendlyName = LPWStrToString(RunningAppRaw.FriendlyName),
CarrierPID = RunningAppRaw.CarrierPID,
StartTickTime = RunningAppRaw.StartTickTime,
SerialId = RunningAppRaw.SerialId,
Processes = ArrayToList(RunningAppRaw.Processes,
RunningAppRaw.TotalPIDs)
};
((List<RunningApp>)Data).Add(runningApp);
Dbg("EnumRunningAppsCallback: out");
return true;
}

static public List<RunningApp> GetRunningApps()
{
RUNNINGAPP_ENUM_CALLBACK Callback = new
RUNNINGAPP_ENUM_CALLBACK(EnumRunningAppsCallback);
```

```
List<RunningApp> list = new List<RunningApp>();
Object data = list;
if ((APIRET)RunningAppEnum(Callback, ref data) == APIRET.SUCCESS)
{
Dbg("GetRunningApps: out");
return list;
}
Else
{
Dbg("GetRunningApps: out (null)");
return null;
}
}

static public RunningApp FindRunningApp(string appID)
{
List<RunningApp> list = GetRunningApps();
if (list == null)
return null;
foreach (RunningApp app in list)
{
if (app.AppID == appID)
return app;
}
return null;
}

//
// DeployedApp functions
static private bool EnumDeployedAppsCallback(
ref Object Data,
[MarshalAs(UnmanagedType.LPWStr)] String AppID)
{
((List<String>)Data).Add(AppID);
```

```
return true;
}

static public List<String> DeployedAppIDs()
{
DEPLOYEDAPP_ENUM_CALLBACK Callback = new
DEPLOYEDAPP_ENUM_CALLBACK(EnumDeployedAppsCallback);
List<String> list = new List<String>();
Object data = list;
if ((APIRET)DeployedAppEnum(Callback, ref data) == APIRET.SUCCESS)
return list;
else
return null;
}

static public List<DeployedApp> DeployedApps()
{
List<String> appIDs = DeployedAppIDs();
List<DeployedApp> deployedApps = new List<DeployedApp>();
foreach (String appID in appIDs)
{
DeployedApp deployedApp = DeployedApp.FromAppID(appID);
if (deployedApp != null)
deployedApps.Add(deployedApp);
}
return deployedApps;
}

static public String DeployedAppDir(String AppID)
{
StringBuilder sbValue = new StringBuilder(MAX_STRING);
APIRET Ret = (APIRET)DeployedAppGetDir(AppID, sbValue, MAX_STRING);
if (Ret == APIRET.SUCCESS)
return sbValue.ToString();
```

```
else
return null;  // Error
}

//
// Helper functions
static public String FriendlyShortcutName(String rawShortcutName)
{
String friendly = System.IO.Path.GetFileName(rawShortcutName);
if (friendly.EndsWith(".lnk",
  StringComparison.InvariantCultureIgnoreCase))
friendly = friendly.Substring(0, friendly.Length - 4);
return (friendly);
}

public int GetIsolationMode()
{
// Isolation. Note: it is allowed to have no checkbox selected at all.
string isolationModeStr = GetProperty("IsolationMode");
if (!string.IsNullOrEmpty(isolationModeStr))
{
if (isolationModeStr.Equals("Data",
  StringComparison.InvariantCultureIgnoreCase))
return ISOLATIONMODE_DATA;
else if (isolationModeStr.Equals("FullAccess",
  StringComparison.InvariantCultureIgnoreCase))
return ISOLATIONMODE_FULL_ACCESS;
else if (isolationModeStr.Equals("Isolated",
  StringComparison.InvariantCultureIgnoreCase))
return ISOLATIONMODE_ISOLATED;
else
return ISOLATIONMODE_CUSTOM;
}
Else
```

```
{
// Legacy (before "IsolationMode" property)
if (GetFileSandbox("") == VirtPackage.SANDBOXFLAGS_COPY_ON_WRITE &&
GetRegistrySandbox("") == VirtPackage.SANDBOXFLAGS_COPY_ON_WRITE &&
GetFileSandbox("%Personal%") == VirtPackage.SANDBOXFLAGS_PASSTHROUGH
&&
GetFileSandbox("%Desktop%") == VirtPackage.SANDBOXFLAGS_PASSTHROUGH &&
GetFileSandbox("UNC") == VirtPackage.SANDBOXFLAGS_PASSTHROUGH)
return ISOLATIONMODE_DATA;
else if (GetFileSandbox("") == VirtPackage.SANDBOXFLAGS_COPY_ON_WRITE
&&
GetRegistrySandbox("") == VirtPackage.SANDBOXFLAGS_COPY_ON_WRITE)
return ISOLATIONMODE_ISOLATED;
else if (GetFileSandbox("") == VirtPackage.SANDBOXFLAGS_PASSTHROUGH &&
GetRegistrySandbox("") == VirtPackage.SANDBOXFLAGS_PASSTHROUGH)
return ISOLATIONMODE_FULL_ACCESS;
else
return ISOLATIONMODE_CUSTOM;
}
}

public void SetIsolationMode(int IsolationMode)
{
switch (IsolationMode)
{
case ISOLATIONMODE_DATA:
SetProperty("IsolationMode", "Data");
break;
case ISOLATIONMODE_ISOLATED:
SetProperty("IsolationMode", "Isolated");
break;
case ISOLATIONMODE_FULL_ACCESS:
SetProperty("IsolationMode", "FullAccess");
break;
case ISOLATIONMODE_CUSTOM:
default:
SetProperty("IsolationMode", "Custom");
break;
}
}

static public Hashtable ReadIniSettingsBuf(String iniBuf)
{
Try
{
String[] lines = iniBuf.Split('\r', '\n');
var values = new Hashtable(StringComparer.InvariantCultureIgnoreCase);
for (int i = 0; i < lines.Length; i++)
{
if (String.IsNullOrEmpty(lines[i]))
continue;
try
{
int equal = lines[i].IndexOf('=');
if (equal != -1)
values.Add(lines[i].Substring(0, equal), lines[i].Substring(equal + 1));
else
values.Add(lines[i], "");
}
catch { }
}
return values;
}
Catch
{
return null;
}
```

```
}

static public Hashtable ReadIniSettings(String IniFile)
{
if (!File.Exists(IniFile))
return null;
try
{
String iniBuf = File.ReadAllText(IniFile, Encoding.Unicode);
if (iniBuf.IndexOf("AppID") == -1)
iniBuf = File.ReadAllText(IniFile, Encoding.ASCII);  // Happens when
modified with notepad
return ReadIniSettingsBuf(iniBuf);
}
Catch
{
return null;
}
}

public class RunningAppsEnumerator
{
IntPtr Context = IntPtr.Zero;

public List<RunningApp> GetRunningApps()
{
RUNNINGAPP_ENUM_CALLBACK Callback = new
RUNNINGAPP_ENUM_CALLBACK(EnumRunningAppsCallback);
List<RunningApp> list = new List<RunningApp>();
Object data = list;
Dbg("GetRunningApps");
if ((APIRET)RunningAppEnumKeepAlive(ref Context, Callback, ref data)
== APIRET.SUCCESS)
return list;
else
return null;
}

~RunningAppsEnumerator()
{
RunningAppEnumKeepAliveFree(Context);
}
}
}

public class DeployedApp
{
public String AppID { get { return m_AppID; } }
internal String m_AppID;
public String Appsize { get { return m_Appsize; } }
internal String m_Appsize;
public String BaseDirName  { get { return m_BaseDirName; } }
internal String m_BaseDirName;
public String CarrierExeName { get { return m_CarrierExeName; } }
internal String m_CarrierExeName;
public long OccupiedSize { get { return GetOccupiedSize(); } }
public long m_OccupiedSize = -1;
public long ExeSize { get { return GetExeSize(); } }
public long m_ExeSize = -1;
public String EngineVersion { get { return GetEngineVersion(); } }
public String m_EngineVersion;

// Basic ini settings
public String Version { get { return GetVersion(); } }
public String m_Version;
public String Publisher { get { return GetPublisher(); } }
public String m_Publisher;
public String BuildUid { get { return GetBuildUid(); } }
public String m_BuildUid;
public String CloudPkgId { get { return GetCloudPkgId(); } }
public String m_CloudPkgId;
public String Streamer { get { return GetStreamer(); } }
public String m_Streamer;
public String FriendlyName { get { return GetFriendlyName(); } }
public String m_FriendlyName;
public String AutoLaunch { get { return GetAutoLaunch(); } }
public String m_AutoLaunch;
public String Shortcuts { get { return GetShortcuts(); } }
public String m_Shortcuts;
public String StopInheritance { get { return GetStopInheritance(); } }
public String m_StopInheritance;
public List<String> IntegratedComponents { get { return GetIntegratedComponents(); } }
public List<String> m_IntegratedComponents;

public Hashtable IniProperties { get { return m_IniProperties; } }
internal Hashtable m_IniProperties;

public DeployedApp(String appID, String baseDirName, String carrierExeName)
{
m_AppID = appID;
m_BaseDirName = baseDirName;
m_CarrierExeName = carrierExeName;
m_IniProperties = VirtPackage.QuickReadIniValues(carrierExeName); //ReadIniSettings(Path.Combine(baseDirName, "VirtApp.ini"));
}

public static long RecursiveDirSize(DirectoryInfo d)
{
long Size = 0;

```

```
// Add file sizes.
try
{
FileInfo[] fis = d.GetFiles();
foreach (FileInfo fi in fis)
{
try
{
Size += fi.Length;
}
catch { }
}
}
catch { }

// Add subdirectory sizes.
try
{
DirectoryInfo[] dis = d.GetDirectories();
foreach (DirectoryInfo di in dis)
{
Size += RecursiveDirSize(di);
}
}
catch { }
return (Size);
}

private String GetVersion()
{
if (!String.IsNullOrEmpty(m_Version) || m_IniProperties == null ||
  m_IniProperties["Version"] == null)
return m_Version;
m_Version = (String)IniProperties["Version"];
```

```
return m_Version;
}

private String GetPublisher()
{
if (!String.IsNullOrEmpty(m_Publisher) || m_IniProperties == null ||
    m_IniProperties["Publisher"] == null)
return m_Publisher;
m_Publisher = (String)IniProperties["Publisher"];
return m_Publisher;
}

private String GetBuildUid()
{
if (!String.IsNullOrEmpty(m_BuildUid) || m_IniProperties == null ||
    m_IniProperties["BuildUID"] == null)
return m_BuildUid;
m_BuildUid = (String)IniProperties["BuildUID"];
return m_BuildUid;
}

private String GetCloudPkgId()
{
if (!String.IsNullOrEmpty(m_CloudPkgId) || m_IniProperties == null ||
    m_IniProperties["CloudPkgId"] == null)
return m_CloudPkgId;
m_CloudPkgId = (String)IniProperties["CloudPkgId"];
return m_CloudPkgId;
}

private String GetStreamer()
{
if (!String.IsNullOrEmpty(m_Streamer) || m_IniProperties == null ||
    m_IniProperties["Streamer"] == null)
```

```
return m_Streamer;
m_Streamer = (String)IniProperties["Streamer"];
return m_Streamer;
}

private String GetFriendlyName()
{
if (!String.IsNullOrEmpty(m_FriendlyName) || m_IniProperties == null
|| m_IniProperties["FriendlyName"] == null)
return m_FriendlyName;
m_FriendlyName = (String)IniProperties["FriendlyName"];
return m_FriendlyName;
}

private String GetAutoLaunch()
{
if (!String.IsNullOrEmpty(m_AutoLaunch) || m_IniProperties == null ||
m_IniProperties["AutoLaunch"] == null)
return m_AutoLaunch;
m_AutoLaunch = (String)IniProperties["AutoLaunch"];
return m_AutoLaunch;
}

private String GetShortcuts()
{
if (!String.IsNullOrEmpty(m_Shortcuts) || m_IniProperties == null ||
m_IniProperties["Shortcuts"] == null)
return m_Shortcuts;
m_Shortcuts = (String)IniProperties["Shortcuts"];
return m_Shortcuts;
}

private String GetStopInheritance()
{
```

```
if (!String.IsNullOrEmpty(m_StopInheritance) || m_IniProperties ==
null || m_IniProperties["StopInheritance"] == null)
return m_StopInheritance;
m_StopInheritance = (String)IniProperties["StopInheritance"];
return m_StopInheritance;
}

// Returns a list of integrated components, or null if not integrated
private List<String> GetIntegratedComponents()
{
if (m_IntegratedComponents != null)   // Result already cached?
return (m_IntegratedComponents.Count == 0 ? null :
m_IntegratedComponents);

// Not cached yet (first time). Perform.
m_IntegratedComponents = new List<string>();   // Leaving it empty if
no integration, so as to indicate: 'cached'
try
{
RegistryKey key = Registry.CurrentUser.OpenSubKey("Software\\VOS\\" +
this.AppID + "\\Integrated", false);
if (key != null)
{
foreach (string keyName in key.GetSubKeyNames())
m_IntegratedComponents.Add(keyName);
}
key.Close();
}
catch { }
return (m_IntegratedComponents.Count == 0 ? null :
m_IntegratedComponents);
}

public static long GetOccuppiedAppSize(string BaseDirName)
{
```

```
if (!Directory.Exists(BaseDirName))
{
return 0;
}
DirectoryInfo d = new DirectoryInfo(BaseDirName);
return RecursiveDirSize(d);
}

private long GetOccupiedSize()
{
if (m_OccupiedSize != -1)
return m_OccupiedSize;
m_OccupiedSize = GetOccuppiedAppSize(m_BaseDirName);
return m_OccupiedSize;
}

private long GetExeSize()
{
if (m_ExeSize != -1)
return m_ExeSize;
if (!File.Exists(m_CarrierExeName))
{
m_ExeSize = 0;
return m_ExeSize;
}
FileInfo f = new FileInfo(m_CarrierExeName);
m_ExeSize = f.Length;
return m_ExeSize;
}

private String GetEngineVersion()
{
if (!String.IsNullOrEmpty(m_EngineVersion))
return m_EngineVersion;
```

```

    // EngineVersion property (available only on 2.0.713 and higher)
    m_EngineVersion = (String)IniProperties["EngineVersion"];
    if (!String.IsNullOrEmpty(m_EngineVersion))
    return m_EngineVersion;

    // No property (backward compatibility mode); try to find AppVirtDll
    String appVirtDll = m_BaseDirName + "\\AppVirtDll_" + m_AppID +
".dll";
    if (!File.Exists(appVirtDll))
    return "";
    System.Diagnostics.FileVersionInfo fileVersionInfo =
System.Diagnostics.FileVersionInfo.GetVersionInfo(appVirtDll);
    if (fileVersionInfo == null)
    return "";
    m_EngineVersion = fileVersionInfo.FileVersion.Replace(" ",
"").Replace(",", ".");  // virtPkg.EngineVer is in the form: "1, 7, 534, 0"
    return m_EngineVersion;
    }

    delegate bool DirectoryExistsDelegate(string folder);
    static bool DirectoryExistsTimeout(string path, int
millisecondsTimeout)
    {
    try
    {
    DirectoryExistsDelegate callback = new
DirectoryExistsDelegate(Directory.Exists);
    IAsyncResult result = callback.BeginInvoke(path, null, null);
    if (result.AsyncWaitHandle.WaitOne(millisecondsTimeout, false))
    {
    return callback.EndInvoke(result);
    }
    else
    {
```

```
//callback.EndInvoke(result);   // Problem: this seems to make the current thread block for response!
return false;
}
}
catch (Exception)
{
return false;
}
}

delegate bool FileExistsDelegate(string file);
static bool FileExistsTimeout(string path, int millisecondsTimeout)
{
try
{
FileExistsDelegate callback = new FileExistsDelegate(File.Exists);
IAsyncResult result = callback.BeginInvoke(path, null, null);
if (result.AsyncWaitHandle.WaitOne(millisecondsTimeout, false))
{
return callback.EndInvoke(result);
}
else
{
//callback.EndInvoke(result);   // Problem: this seems to make the current thread block for response!
return false;
}
}
catch (Exception)
{
return false;
}
}
```

```
static public DeployedApp FromAppID(String appID)
{
try
{
RegistryKey key = Registry.CurrentUser.OpenSubKey("Software\\VOS\\" +
  appID, false);
if (key == null)
return null;
String baseDirName = (String)key.GetValue("BaseDirName");
String carrierExeName = (String)key.GetValue("CarrierExeName");
System.Diagnostics.Debug.WriteLine(appID + ": " + carrierExeName);

// Detect & avoid disconnected shares / mapped drives, as their I/O
  can take a long time before failing..
if (!FileExistsTimeout(carrierExeName, 1000)) //||
  !DirectoryExistsTimeout(baseDirName, 1000))
return null;

return new DeployedApp(appID, baseDirName, carrierExeName);
}
catch
{
return null;
}
}
}

//
// Packager command line functions
public class PackagerCmdLine
{
static public Hashtable ReadIni(string appExe, string packagerExe)
{
```

```
string tempFile = Path.GetTempFileName();
try { System.IO.File.Delete(tempFile); }
catch { }

int exitCode = -1;
string args = string.Format("-Quiet -ExtractIni \"{0}\" \"{1}\"",
appExe, tempFile);
bool execOk = ExecProg(packagerExe, args, true, ref exitCode);
if (!execOk)
return null;
if (!File.Exists(tempFile))
return null;
try   // finally
{
if (exitCode != (int)VirtPackageAPI.VirtPackage.APIRET.SUCCESS)
return null;
return VirtPackage.ReadIniSettings(tempFile);
}
Finally
{
try { System.IO.File.Delete(tempFile); }
catch { }
}
}

static public bool SetProperties(string appExe, Hashtable values,
string packagerExe)
{
// -SetProperties
int exitCode = -1;
string properties = "";
foreach (DictionaryEntry value in values)
{
if (!string.IsNullOrEmpty(properties))
properties += ",,";
properties += value.Key + "=" + value.Value;
}
string args = string.Format("-Quiet \"-SetProperties:{0}\" {1}",
properties, appExe);
bool execOk = ExecProg(packagerExe, args, true, ref exitCode);
return (execOk && exitCode ==
(int)VirtPackageAPI.VirtPackage.APIRET.SUCCESS);
}

static public string PackagerExe()
{
return
Path.Combine(Path.GetDirectoryName(System.Diagnostics.Process.GetCurrentProc
ess().MainModule.FileName), "Packager.exe");
static public bool GetProperties(string appExe, Hashtable values,
string packagerExe)
{
// -GetProperties
int exitCode = -1;
string properties = "";
foreach (DictionaryEntry value in values)
{
if (!string.IsNullOrEmpty(properties))
properties += ",,";
properties += value.Key + "=" + value.Value;
}
string args = string.Format("-Quiet \"-GetProperties:{0}\" {1}",
properties, appExe);
bool execOk = ExecProg(packagerExe, args, true, ref exitCode);
return (execOk && exitCode ==
(int)VirtPackageAPI.VirtPackage.APIRET.SUCCESS);
}

static public string PackagerExe()
```

```
{
return
Path.Combine(Path.GetDirectoryName(System.Diagnostics.Process.GetCurrentProcess().MainModule.FileName), "Packager.exe");
}
}
}
catch { }
return false;
}
```

# REFERENCES

1. https://www.prosyscom.tech/tips-tricks/what-is-a-portable-app-and-why-does-it-matter/
2. https://www.howtogeek.com/290358/what-is-a-portable-app-and-why-does-it-matter/
3. http://usefultipssaring.blogspot.com/2014/08/5-portable-app-creators-to-make-any.html
4. https://promotionaldrives.com/blog/portable-apps-pros-cons/
5. https://en.wikipedia.org/wiki/Cameyo
6. https://searchvmware.techtarget.com/definition/VMware-ThinApp 2.4.2
7. https://en.wikipedia.org/wiki/VMware_ThinApp
8. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware-ThinApp-Reviewers-Guide.pdf
9. http://www.appvshop.com/spoon-studio.html
10. https://enigmaprotector.com/en/aboutvb.html
11. http://carlcheo.com/portable-app-creators
12. https://www.rorymon.com/blog/spoon-net-virtualization/
13. https://www.rorymon.com/blog/evalaze-application-virtualization-2-1/
14. https://www.techcheers.com/portable-app-creator-windows/
15. https://www.hongkiat.com/blog/create-portable-apps-cameyo/
16. https://en.wikipedia.org/wiki/PortableApps.com#History
17. https://reviews.financesonline.com/p/cameyo/
18. https://www.thewindowsclub.com/list-of-portable-apps-suites-for-windows
19. https://files.cameyo.com/sdk/index.html?runningappenum.htm

BIBLIOGRAPHY:

https://github.com/cameyo/cameyo

https://github.com/INRIA/spoon

https://github.com/josephspurrier/goappmation

https://portableapps.com/apps

http://carlcheo.com/portable-app-creators