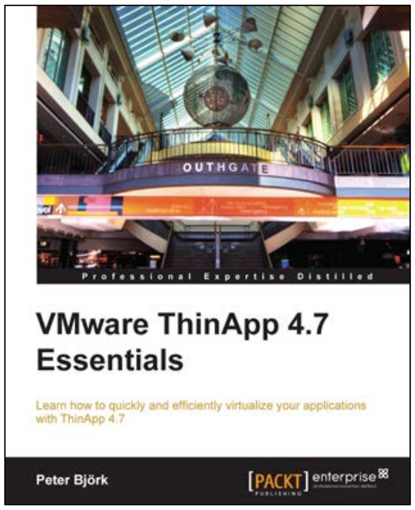
VMware ThinApp 4.7 Essentials

Peter Björk


Professional Expertise Distilled
VMware ThinApp 4.7 Essentials
Learn how to quickly and efficiently virtualize your applications with ThinApp 4.7
Peter Björk

Chapter No. 1
"Application Virtualization"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.1 "Application Virtualization"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Peter Björk has many years of ThinApp experience. He started out working with Thinstall, and continued after VMware acquired the product in 2008, renaming it ThinApp. Peter supports ThinApp in the EMEA region. As a teacher, Peter has educated many ThinApp packagers around the world. Peter lives in Sweden with his wife and two kids, a boy and a girl.

I would like to thank the people who have supported me throughout the writing of this book. First and foremost, my thanks go out to my wonderful wife, Lena. Without her help and support, this book would never have been written. I know ThinApp to a depth not many others do, but my writing skills are clearly insufficient. Luckily, Lena has the gift of words, so with her support I managed to write this book. To my two wonderful kids, Albin and Filippa, who constantly remind me of what's important in my life. I would also like to thank my reviewers, Aaron Black and Adam Eckerle. Their valuable input was important for this book. I also thank PACKT Publishing for trusting in me to write this book. It's my first book and the team: Andrew Duckworth, Abhishek Kori, and Arun Nadar really helped me through the process. I must thank Jonathan Clark for coming up with the great idea of Thinstall, and with that created what became the better part of the my career. Last but not least, my thoughts go to the family of late Ge van Geldorp. Ge was an amazing developer and without his genius coding, ThinApp would not be what it is today. Ge, you are missed every day.

VMware ThinApp 4.7 Essentials

VMware ThinApp 4.7 is an application virtualization solution which allows its admins to package Windows applications so that they are portable.

"VMware ThinApp 4.7 Essentials" shows you how to create and deploy ThinApp packages in order to improve the portability, manageability, and compatibility of applications by encapsulating them from the underlying operating system on which they are executed.

ThinApp eliminates application conflicts, reducing the need and cost of recoding and regression testing.

No matter if you are completely new to VMware ThinApp or an experienced ThinApp packager, this is the book for you. I've made an effort to make sure that everyone can learn something in each chapter. This book will cover everything needed to become a successful ThinApp packager. This book does not talk about the competition. I wanted this book to be technically oriented and so very little, if any, is of a non-technical nature.

What This Book Covers

Chapter 1, Application Virtualization, covers basic application virtualization concepts. It also covers important concepts like isolation modes, the sandbox, and much more.

Chapter 2, Application Packaging, explains the whole packaging process. It takes you through a simple packaging example, which you can easily perform yourself. Entry points and the data container are explained as well as how your packaging environment affects your packages..

Chapter 3, Deployment of ThinApp Packages, walks you through the different methods for deployment as it's now time to deploy the package to your end users. We cover ThinApp native methods of deployment as well as using VMware View and VMware Horizon Application Manager.

Chapter 4, Updating and Tweaking Your ThinApp Project, covers how to maintain your packages using different methods and helps you choose the appropriate method for different types of updates as after a while, all applications must be updated one way or another.

Chapter 5, How to Distribute updates, covers how to deploy your newly created updated package. ThinApp offers many different methods, so a good portion is spent on helping you identify which methods to use for which update.

Chapter 6, Design and Implementation Considerations using ThinApp, outlines general implementation guidelines. The chapter goes through things you need to be aware of in order to successfully implement ThinApp in your environment.

Chapter 7, Troubleshooting, teaches you how to conduct efficient troubleshooting of ThinApp packages, since sometimes you may face an issue while trying to package a certain application. I have shared some tips and tricks that I've picked up from my many years of ThinApp packaging.

Appendix, References, provides you with a complete Package.ini parameter reference as well as all folder macros, and environment variables supported by ThinApp.

1
Application Virtualization

In this chapter we will cover a general overview of application virtualization and ThinApp. We will start by exploring what application virtualization is and why it is superior to local installations. We will then cover the architecture behind ThinApp and how we can manipulate and customize ThinApp packages to suit our specific requirements.

By the end of this chapter, you will have learned about:

- Application virtualization
- Why you should use application virtualization
- ThinApp architecture
- Common ThinApp vocabulary
- The sandbox
- Isolation modes
- Application linking with the help of AppLink

Application virtualization

Application virtualization encapsulates an application and all of its components into a package that is easy to deploy and manage. Using virtualization allows you to execute the application as if it was locally installed when it is not. Normally when you install an application it will register DLL files, create registry keys, and copy files into your operating system. This modifies your operating system and you will always run the risk of overwriting something already installed and breaking an existing application. By virtualizing the application, you will never install anything on the client, you will simply execute the application. There is also a virtualization layer hooking into the APIs of the application. When hooking the API for, let's say Open File, it is possible for the virtualization layer to present a virtual environment for the application, thus fooling the application into thinking it is already locally installed and therefore allowing it to execute.

The benefits of using application virtualization are many. Your operating system stays clean. By having clean machines, your clients will be more stable. A virtualized application is much easier to deploy, maintain, and retire than a natively installed one. With application virtualization, it is often possible to run two otherwise conflicting applications simultaneously on the same machine. Not using application virtualization makes it pretty much impossible to have Microsoft Office 2003 and Microsoft Office 2010 installed on the same client and run both at the same time.

ThinApp overview

VMware ThinApp is a packaging format. Like MSI and other packaging formats, ThinApp simplifies application deployment. ThinApp uses virtualization to package your application, which lets you execute the packaged application without having to install it. When using ThinApp, you simply need to have access to your package in order to use the application, as compared to the legacy MSI format in which you need to install and register your application on the local machine. As a side effect of using virtualization, you can isolate the filesystem and registry components from the locally installed applications as well as from other virtualized applications. This allows you to run conflicting applications on the same machine. Since you will never install anything locally, the use of an application will not alter your operating system. Your client will be much cleaner, more stable, and will operate faster for a longer time. ThinApp minimizes the constant reinstallation of the operating system due to repetitive application installs, which leave residue and often create conflicts that eventually leave the operating system in a degraded state necessitating a complete system rebuild.

ThinApp has one very obvious advantage over other solutions out there. It is agentless, meaning you need nothing locally installed in order to execute an application packaged with the help of ThinApp. Being agentless greatly reduces the administration overhead. When a new ThinApp version is released, you don't have to touch any existing packages already deployed. Start using the new version to capture new applications. You can happily deploy these next to an old ThinApp package since there is no conflict between ThinApp versions running side-by-side. Being agentless also lets you offer an application to a user bringing his or her own device without the need to ever touch the device. You don't run the risk of being accused of altering the user's machine.

Out of the box, ThinApp is capable of virtualizing 60 – 80 percent of your applications. Having more ThinApp knowledge and experience might allow you to virtualize up to 85 – 90 percent. You will, most of the time, never achieve 100 percent virtualization. This means you will, most of the time, have two packaging formats in place – native installation (often MSI) and ThinApp. ThinApp supports virtualizing Services, COM, and DCOM objects. ThinApp does not support virtualizing device drivers, Network visible DCOM, Global Hook DLLs, and COM+. There might be workarounds to these limitations. One of these could be to load what is not supported outside the virtual environment. One of the main reasons to virtualize an application is to keep your operating system clean. This is why ThinApp does not make many changes to the operating system when registering a ThinApp package. Registering a package will give you a certain level of shell integration, such as shortcuts, file type registrations, and a few more, but not all. Context menus are a typical example. This changed user experience might be a reason not to virtualize an application, even though ThinApp can package it. For instance, 7-Zip adds a context menu item so that when you right-click on a ZIP file in Windows Explorer, you can perform zip/unzip operations without having to open the application directly. A 7-Zip ThinApp package will happily perform zip/unzip operations when launched directly, but the users will not have access to the right-click context menu. Most of the times you can create context menus pointing to a virtualized application but it is not something ThinApp creates automatically for you when registering the package.

Even though you will probably not be able to reach 100 percent application virtualization, ThinApp adds significant value to your application's deployment and management infrastructure. Every application you manage to virtualize will be easier to maintain and cheaper to support.

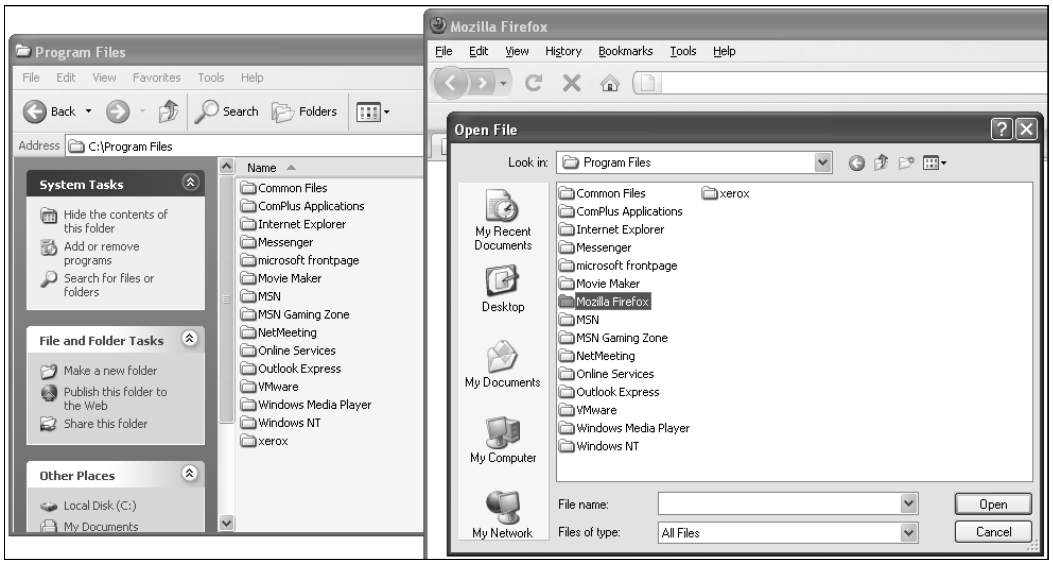
ThinApp architecture

Since it cannot be mentioned too many times, ThinApp is agentless. Nothing needs to be installed on the client in order to run and use a ThinApped application. The ThinApp runtime is built into each one of the ThinApp packages you create. ThinApp does not create conflicts between different versions of ThinApp runtimes, so you can run packages built using different ThinApp versions on one single machine.

The ThinApp runtime manages file and registry access within the virtual environment. With the help of isolation modes you can decide what may or may not be modified on the native operating system. The ThinApp runtime loads processes and manages memory. Because it is the ThinApp runtime that launches a process, the runtime now monitors all API calls made by the process. The runtime is also able to intercept the API calls and manipulate both the request and reply. This is referred to as **hooking the API calls**. The ThinApp runtime hooks hundreds of Win32 APIs in order to create the virtual environment. Let's say an application tries to open a file. The ThinApp runtime sees this request, hooks it, and is now capable of passing a virtualized file to the application, instead of serving the native file to the application. The ThinApp runtime does not hook all possible Windows APIs, only the ones needed to present a virtual environment to the application package. API calls to hardware such as graphical drivers are not hooked.

A ThinApp package contains not only the ThinApp runtime, but also includes a virtualized registry and filesystem. You as a packager decide the content of the virtual environment during packaging. The virtual environment built into the package is called the read-only version of the virtual environment. The end user cannot modify the content within the package. Only you as a packager can change the content.

Changes made by either the user or the application itself are often stored in the sandbox. The sandbox content is a part of the whole virtual environment known to the application.



The view of the environment of a package is a merge between the physical and the virtualized. In the previous screenshot, Mozilla Firefox sees the content of native `C:\Program Files` as well as the virtualized folder called **Mozilla Firefox**. The **Mozilla Firefox** folder is not available to the operating system (Explorer window).

When the virtualized application is launched, the virtual environment is initiated by the ThinApp runtime and presented to the executing process. The application believes it is locally installed on the machine. The packaging process of ThinApp does not alter the application's files in any way. The ThinApp runtime loads the processes and by launching it, the ThinApp runtime can hook into the API calls made by the processes and present the virtual environment.

Common ThinApp vocabulary

In order to have a meaningful discussion about ThinApp, we need to agree on some common vocabulary. I prefer to give you this vocabulary earlier in the book rather than later. If you have already used ThinApp, most of this will already be known. If you are new to ThinApp, don't worry, as we will cover all of it in more detail as the book progresses.

The capturing process

This is the whole process of capturing an application. You can run **Setup Capture**, install your application, and save the capture into a project folder. The capture process analyzes all changes made to your capture machine and stores those into a project folder. These changes are what will become the virtual environment and make the captured application believe it is locally installed on the target machine.

The capture machine

This is the machine on which you run the capture process. Most of the time it's a virtual machine since that allows easy reversion to different machine states (snapshots). After successfully capturing an application, you will revert to a clean state before you capture a new application.

The project folder

This is the outcome of your capturing process. Now the real work as a packager begins. It's the project folder that contains the virtualized environment such as files and registry keys recorded during your capturing process.

The package

When you compile your project folder, the outcome will be the package. The package is what your users consume in order to execute the captured application. The package will normally be found in the `bin` folder within your project folder. A package can be one single file or multiple files, one being the data container and others being entry points.

The data container

The data container is the file containing your compiled project folder. It's the container for the whole virtual environment and the ThinApp runtime.

The entry point

Entry points are the doorways for the user to access the virtualized application. An entry point specifies what will be executed in the virtualized environment of your data container. The target of your entry point may or may not be virtualized. It is possible to have an entry point for a Java Runtime package launching your locally installed Internet Explorer. Internet Explorer would see the virtualized environment and therefore use the version of Java packaged. An entry point can also be a data container. Otherwise, if it's only an entry point, the data container must be located in the same folder as the entry point. An entry point can be used to any data container. The entry point simply searches for the specified data container's name and will happily use any data container. An entry point contains registration information such as icon, file types, object types, protocols and where to create shortcuts.

Compiling or building your ThinApp package

The building process is the process of taking the content of your project folder and compiling it into a virtual environment. This process can be issued from within ThinApp's capturing tool, **Setup Capture**, or from within your project folder by launching the `build.bat` batch file. Every time you change the content of your project folder, you'll have to recompile it in order for the changes to be applied to the package.

The build machine

This is any machine you can use to compile your project. It may or may not be your capture machine. You do not have to use a certain operating system or even a clean machine in order to compile your package. Any machine should do the trick. The build machine must have access to the ThinApp utilities folder and your project folder in order to successfully compile your project.

The ThinApp utilities folder

This is the folder created during the installation of VMware ThinApp. Most of the time it's found in `C:\Program Files\VMware\VMware ThinApp`. Since ThinApp utilities are virtualized, you can move this folder to any location. I personally store the folder on a network share for easy access from all my different capture machines.

The ThinApp runtime

This package embedded runtime allows the virtual environment to be created. The ThinApp runtime loads the virtualized application's processes and DLLs. It hooks Windows APIs in order to present a virtualized environment to the virtualized application.

Read-Only data

This is the virtual environment, filesystem, and registry, compiled into the ThinApp package. Since the package is in a compiled format, no regular end user can open this file and modify its content.

Read and write data

This is what we call the data stored in the sandbox. The sandbox is where ThinApp stores changes made to the environment by the virtualized application or the end user. Deleting the sandbox will revert the package to its read-only data state.

Folder macros

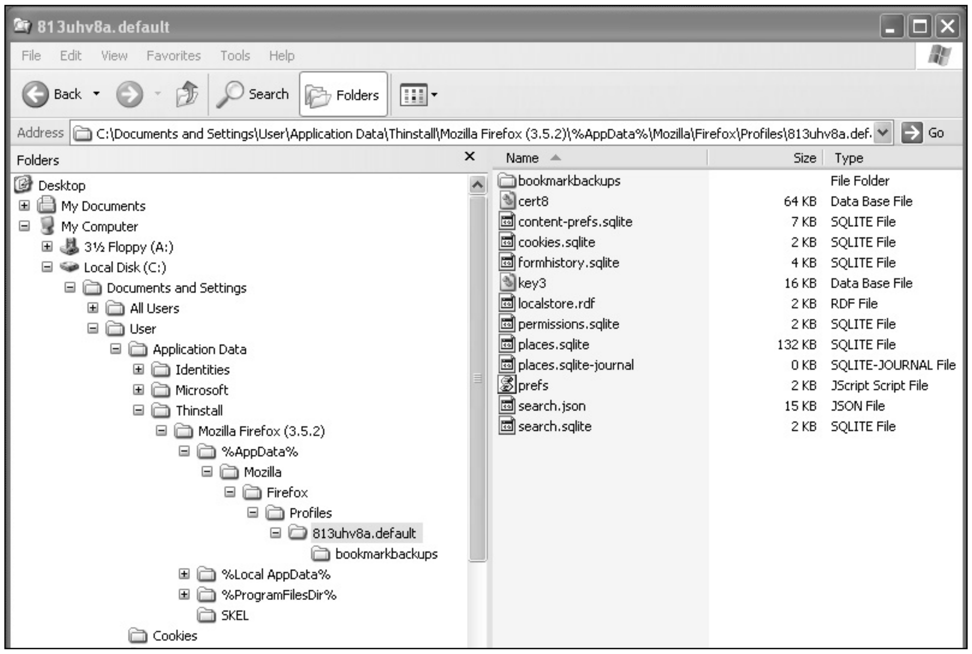
These are much like system variables in a Windows operating system, but these are ThinApp-specific variables. Some folder macros share the same name as Windows variables such as `%AppData%` pointing to the users' roaming profile. But others are different, for example `%ProgramFilesDir%` represents the system variable `%ProgramFiles%`. When you use VBScripts within your packages, you must understand that there is a difference between folder macros and system variables. The use of folder macros allows package portability. When you launch a package on an English OS, your `%ProgramFilesDir%` will be `C:\Program Files`, while on a German OS it is the same folder macro pointing to `C:\Programme`. This way, the application you virtualized will find its installation folder where it expects to find it, no matter what language of OS it's running on. You can find a list of all folder macros in *References*.

The sandbox

Many applications require the ability to write or modify data on the computer's filesystem and registry. When this need arises, ThinApp writes this data to the sandbox. This process is configurable and can be controlled through isolation modes.

The sandbox will store user settings so that these are preserved between application launches. If you delete the sandbox, the package will revert to its vanilla state. How big the sandbox will become depends on two factors: isolation modes and the behavior of the application.

The sandbox is a normal folder storing complete, fully functional versions of modified files. Let's say you run a virtualized application using `.ini` configuration files. Changing the application's configuration would alter the `.ini` file, and in your sandbox you would find the new version. It's fully functional and possible to open, for example, in native Notepad. The files are stored in folder macros, representing the path to the file. Since files are stored as native files and not in a binary blob, it's easy to perform backups of your sandbox. You can do single file restores and your antivirus software can scan its content without any problems.



Address C:\Documents and Settings\User\Application Data\Thinstall\Mozilla Firefox (3.5.2)\%AppData%\Mozilla\Firefox\Profiles\813uhv8a.def

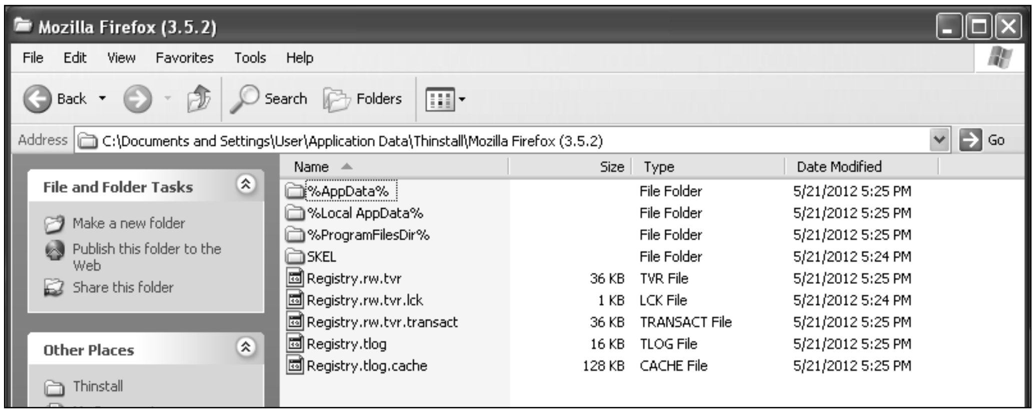
```
Desktop
  My Documents
  My Computer
    3½ Floppy (A:)
    Local Disk (C:)
      Documents and Settings
        All Users
        User
          Application Data
            Identities
            Microsoft
            Thinstall
              Mozilla Firefox (3.5.2)
                %AppData%
                  Mozilla
                    Firefox
                      Profiles
                        813uhv8a.default
                          bookmarkbackups
                %Local AppData%
                %ProgramFilesDir%
                SKEL
          Cookies
```

Name	Size	Type
bookmarkbackups		File Folder
cert8	64 KB	Data Base File
content-prefs.sqlite	7 KB	SQLITE File
cookies.sqlite	2 KB	SQLITE File
formhistory.sqlite	4 KB	SQLITE File
key3	16 KB	Data Base File
localstore.rdf	2 KB	RDF File
permissions.sqlite	2 KB	SQLITE File
places.sqlite	132 KB	SQLITE File
places.sqlite-journal	0 KB	SQLITE-JOURNAL File
prefs	2 KB	JScript Script File
search.json	15 KB	JSON File
search.sqlite	2 KB	SQLITE File

The previous screenshot shows the sandbox contents stored as plain files in a path represented by a folder macro.

Modifications to the registry are also kept in the sandbox. In order to guarantee integrity, the registry is stored in a transactional database format. This makes it a little harder to investigate the contents of the registry changes stored in the sandbox, but with the tool called `vregtool.exe` found in the ThinApp utilities folder, it's still possible. It's important to maintain the integrity of the registry since the registry in the sandbox also includes a file database, telling the ThinApp runtime where to find each file.

The registry files are found in the root of the sandbox and are all called **Registry**.

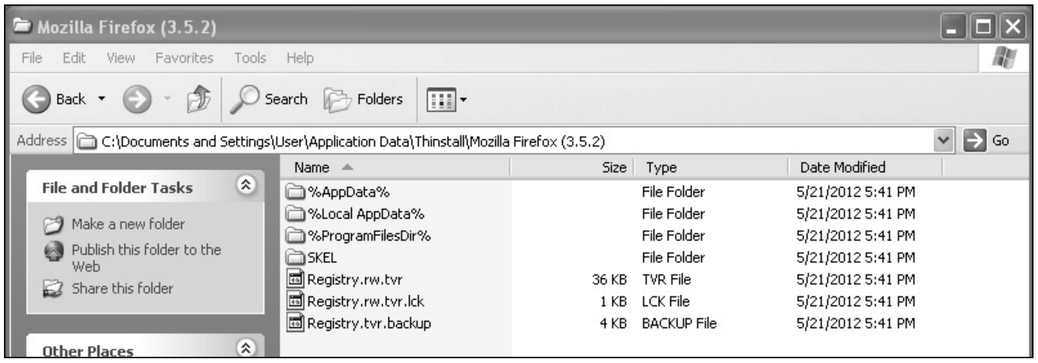


Address C:\Documents and Settings\User\Application Data\Thinstall\Mozilla Firefox (3.5.2)

Name	Size	Type	Date Modified
%AppData%		File Folder	5/21/2012 5:25 PM
%Local AppData%		File Folder	5/21/2012 5:25 PM
%ProgramFilesDir%		File Folder	5/21/2012 5:25 PM
SKEL		File Folder	5/21/2012 5:24 PM
Registry.rw.tvr	36 KB	TVR File	5/21/2012 5:25 PM
Registry.rw.tvr.lck	1 KB	LCK File	5/21/2012 5:24 PM
Registry.rw.tvr.transact	36 KB	TRANSACT File	5/21/2012 5:25 PM
Registry.tlog	16 KB	TLOG File	5/21/2012 5:25 PM
Registry.tlog.cache	128 KB	CACHE File	5/21/2012 5:25 PM

The previous screenshot is an example of sandbox contents.

The database format for storing the registry was introduced in ThinApp Version 4.0.4. With `DisableTransactionRegistry=1` in your `Package.ini` you can still use the legacy format, which uses a flat file with a backup of the last known good state. It's not very likely that you will want to use the legacy format, but in some rare implementations it has proven to speed up execution of the package, especially if the user's sandbox is stored on a network share.



Address C:\Documents and Settings\User\Application Data\Thinstall\Mozilla Firefox (3.5.2)

Name	Size	Type	Date Modified
%AppData%		File Folder	5/21/2012 5:41 PM
%Local AppData%		File Folder	5/21/2012 5:41 PM
%ProgramFilesDir%		File Folder	5/21/2012 5:41 PM
SKEL		File Folder	5/21/2012 5:41 PM
Registry.rw.tvr	36 KB	TVR File	5/21/2012 5:41 PM
Registry.rw.tvr.lck	1 KB	LCK File	5/21/2012 5:41 PM
Registry.tvr.backup	4 KB	BACKUP File	5/21/2012 5:41 PM

The previous screenshot is a sandbox using the legacy method of storing the registry.

The sandbox can be located anywhere as long as the end user has permission to modify the location. The sandbox will be created and updated in the context of the user.

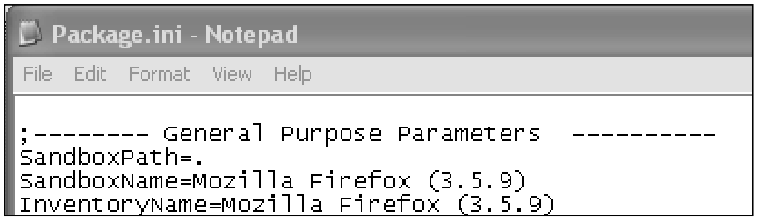
You can specify the location of the sandbox using the parameter `SandboxPath=` in `Package.ini` (more information about `Package.ini` can be found in the next chapter). If you do not specify `SandboxPath=`, the default location will be the user's roaming profile, in a folder called `Thinstall`. You can override the sandbox location using environment variables or by creating a folder called either the project's sandbox name or simply `Thinstall` in the same folder as the package.

You can use `SandboxPath=` in `Package.ini` in different ways.

The following is how you store the sandbox in a location next to the package:

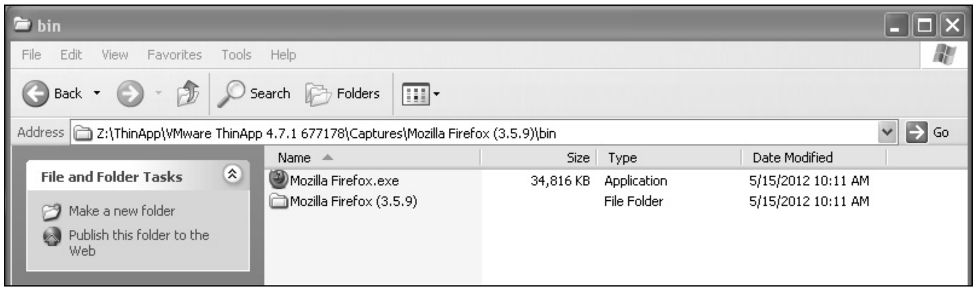
```
[BuildOptions]
SandboxPath=.
```

This is shown in the following screenshot:



```
;-------- General Purpose Parameters  ----------
SandboxPath=.
SandboxName=Mozilla Firefox (3.5.9)
InventoryName=Mozilla Firefox (3.5.9)
```

The following screenshot shows the result:



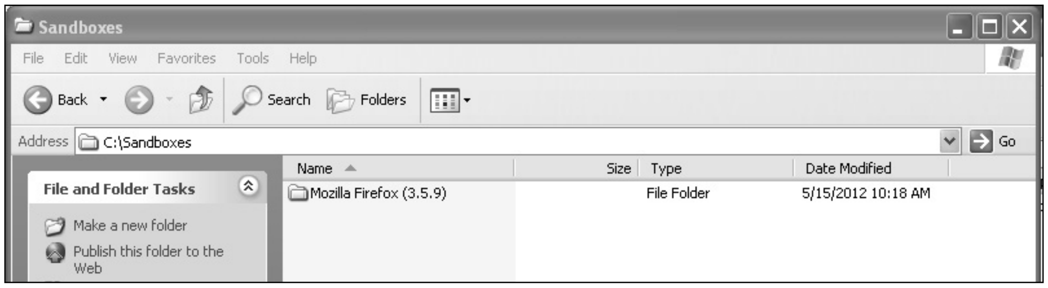
Address: Z:\ThinApp\VMware ThinApp 4.7.1 677178\Captures\Mozilla Firefox (3.5.9)\bin

Name	Size	Type	Date Modified
Mozilla Firefox.exe	34,816 KB	Application	5/15/2012 10:11 AM
Mozilla Firefox (3.5.9)		File Folder	5/15/2012 10:11 AM

More examples are given as follows:

```
[BuildOptions]
SandboxPath=C:\Sandboxes
```

The following screenshot shows the result:

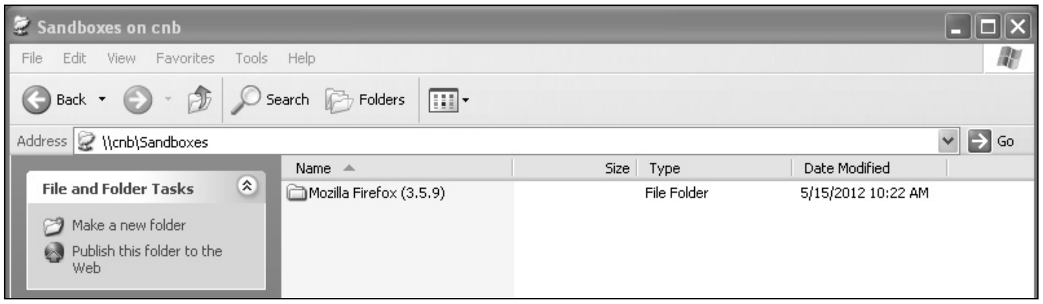


C:\Sandboxes

Name	Size	Type	Date Modified
Mozilla Firefox (3.5.9)		File Folder	5/15/2012 10:18 AM

```
[BuildOptions]
SandboxPath=\\cnb\Sandboxes
```

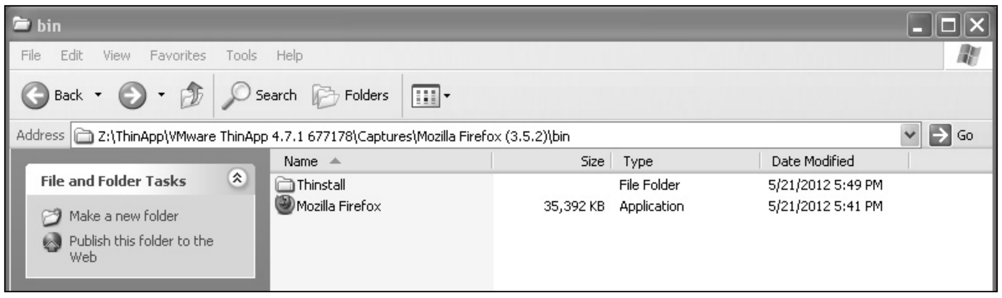
The following screenshot shows the result:



\\cnb\Sandboxes

Name	Size	Type	Date Modified
Mozilla Firefox (3.5.9)		File Folder	5/15/2012 10:22 AM

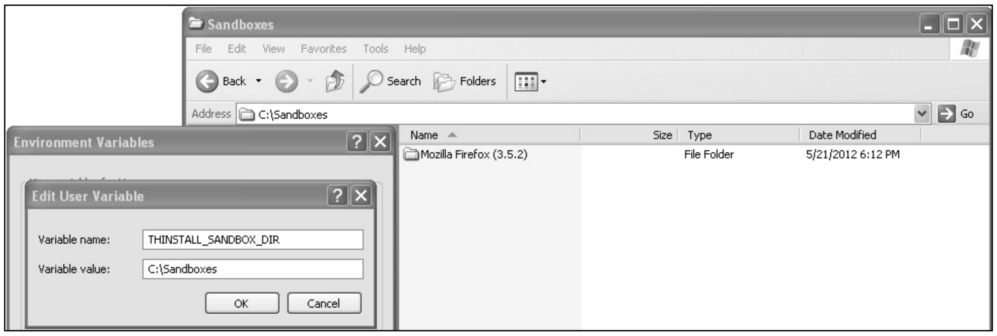
Creating a folder called **Thinstall** next to the package will change the sandbox location. This comes in handy especially during troubleshooting. By using a **Thinstall** folder, you can override the content in your existing sandbox. The **Thinstall** folder is shown in the following screenshot:



Z:\ThinApp\VMware ThinApp 4.7.1 677178\Captures\Mozilla Firefox (3.5.2)\bin

Name	Size	Type	Date Modified
Thinstall		File Folder	5/21/2012 5:49 PM
Mozilla Firefox	35,392 KB	Application	5/21/2012 5:41 PM

Using environment variables to override a package sandbox location allows you to use the same package in many different environments. Let's say you want to store the sandbox in the default location on laptops, while you want to store them on a network share on your Terminal servers. Using an environment variable on your Terminal servers allows you to re-use the package without rebuilding it.



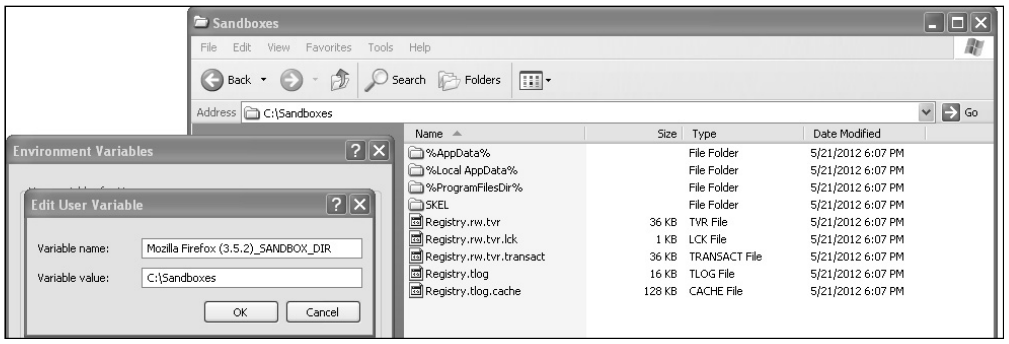
Address: C:\Sandboxes

Variable name: THINSTALL_SANDBOX_DIR

Variable value: C:\Sandboxes

Name	Size	Type	Date Modified
Mozilla Firefox (3.5.2)		File Folder	5/21/2012 6:12 PM

`THINSTALL_SANDBOX_DIR` overrides the sandbox location for all of your packages.



Address: C:\Sandboxes

Variable name: Mozilla Firefox (3.5.2)_SANDBOX_DIR

Variable value: C:\Sandboxes

Name	Size	Type	Date Modified
%AppData%		File Folder	5/21/2012 6:07 PM
%Local AppData%		File Folder	5/21/2012 6:07 PM
%ProgramFilesDir%		File Folder	5/21/2012 6:07 PM
SKEL		File Folder	5/21/2012 6:07 PM
Registry.rw.tvr	36 KB	TVR File	5/21/2012 6:07 PM
Registry.rw.tvr.lck	1 KB	LCK File	5/21/2012 6:07 PM
Registry.rw.tvr.transact	36 KB	TRANSACT File	5/21/2012 6:07 PM
Registry.tlog	16 KB	TLOG File	5/21/2012 6:07 PM
Registry.tlog.cache	128 KB	CACHE File	5/21/2012 6:07 PM

The environment variable `%SandboxName_SANDBOX_DIR%` redirects a specific package's sandbox location. Please note the variable value specifies the root of your sandbox folder.

ThinApp searches for the sandbox in a specific order. ThinApp starts by looking for the environment variable, `%SandboxName_SANDBOX_DIR%` followed by `%THINSTALL_SANDBOX_DIR%`. If no environment variable is found, ThinApp will look for the following folders and store the sandbox in the following locations:

- `LOCATION_OF_PACKAGE\SandboxName.ComputerName`

  For example, `C:\Program Files\Firefox\Mozilla Firefox 3.5.2.My_Computer`

- `LOCATION_OF_PACKAGE\SandboxName`

  For example, `C:\Program Files\Firefox\Mozilla Firefox 3.5`

- `LOCATION_OF_PACKAGE\Thinstall\SandboxName.ComputerName`

  For example, `C:\Program Files\Firefox\Thinstall\Mozilla Firefox 3.5.2.My_Computer`

- `LOCATION_OF_PACKAGE\Thinstall\SandboxName`

  For example, `C:\Program Files\Firefox\Thinstall\Mozilla Firefox 3.5.2`

- `SandboxPath_In_Package.ini\SandboxName.ComputerName`

  For example, `H:\Sandboxes\Mozilla Firefox 3.5.2.My_Computer`

- `SandboxPath_In_Package.ini\SandboxName`

  For example, `H:\Sandboxes\Mozilla Firefox 3.5.2`

If ThinApp fails to find `%SandboxName_SANDBOX_DIR%`, `%THINSTALL_SANDBOX_DIR%`, a `Thinstall` folder next to itself, or `SandboxPath=` in `Package.ini`, then ThinApp will create the sandbox in the default location, that is, in the user's roaming profile (`%AppData%`).

The search order for the sandbox in `%AppData%` is:

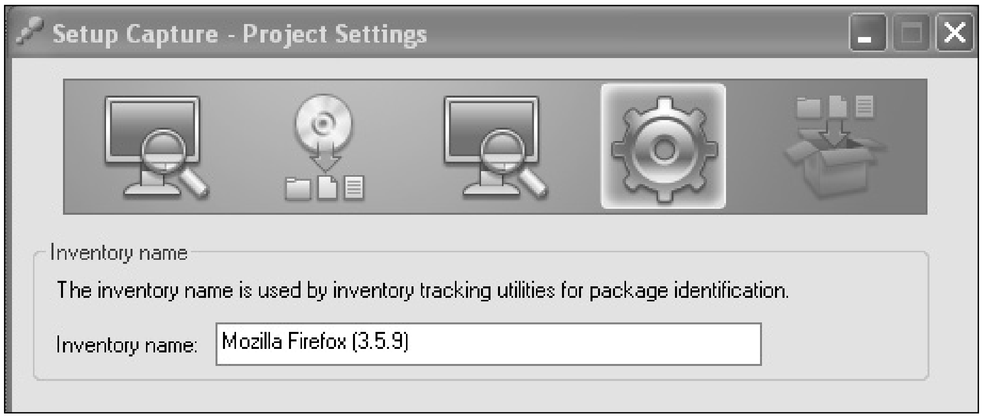
- `%AppData%\Thinstall\SandboxName.ComputerName`

  For example, `C:\Documents and Settings\User\Application Data\Thinstall\Mozilla Firefox 3.5.2.My_Computer`

- `%AppData%\Thinstall\SandboxName`

  For example, `C:\Documents and Settings\User\Application Data\Thinstall\Mozilla Firefox 3.5.2`

You can change the name of the sandbox. The default name will be taken from **Inventory name** specified during the capturing process, as shown in the following screenshot:



Inventory name

The inventory name is used by inventory tracking utilities for package identification.

Inventory name: Mozilla Firefox (3.5.9)

Using the parameter `SandboxName=` in `Package.ini` enables you to set the sandbox name.

Isolation modes

Isolation modes are by far the most important thing to fully understand when it comes to ThinApp. Most of the troubleshooting you will face is related to isolation modes in one way or another. Isolation modes are the packager's method of specifying what level of interaction the package is allowed to have with the underlying operating system.

You can specify different isolation modes on a per directory or registry sub-tree basis. Any child will inherit its parent isolation mode if not overridden.

ThinApp offers three different isolation modes.

Merged

Merged allows the virtualized application to interact with local files, folders, and registry keys. The package can read local elements and is able to modify local elements. Any new element will be created on the local system. If any of the virtualized elements are modified, the modifications will be stored in the sandbox.

Merged mostly mimics the behavior of a natively installed application. The actions of the package are still subject to the privileges of the user running the application. If the user is not allowed to modify a location, the standard operating system dialog box will be displayed saying so.

WriteCopy

WriteCopy will allow the package to read any local elements, but if modified, the modification will end up in the sandbox and not the local system. If you create a new file or registry key in a WriteCopy location, it will be sandboxed. Modifications made to virtualized elements will be sandboxed.

WriteCopy will protect your local system from being modified by a virtualized application. WriteCopy is often used to allow applications demanding higher privileges to be able to executed by a standard user. The application thinks it is capable of modifying `C:\Windows` but all those operations end up in the sandbox.

Full

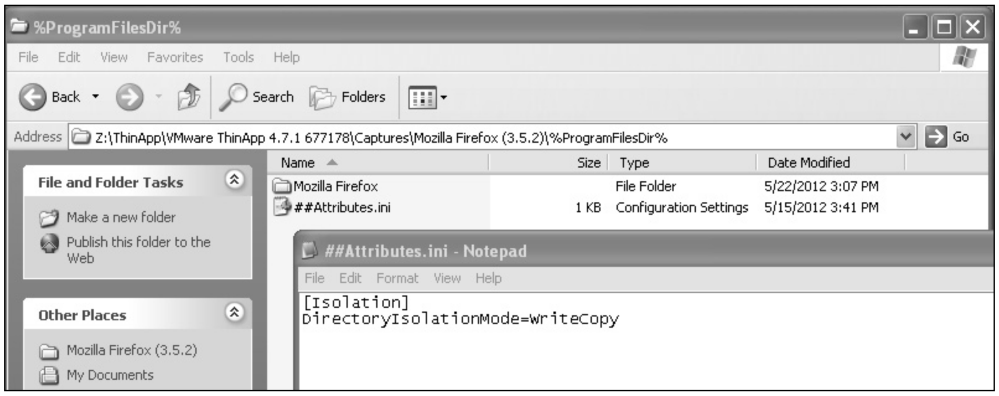
Full isolation mode will keep the virtualized application from accessing anything locally on the underlying operating system. Physical elements are hidden from the virtualized application. If you fully isolate a folder, only the folder's virtualized content will be available. New elements or modifications of a virtualized element will end up in the sandbox.

Full is mostly used to protect the virtualized application from seeing conflicting elements present on the local machine. Take for example, your virtualized Microsoft Office 2010 having Microsoft Office 2003 locally installed. If you don't protect the virtualized Office from seeing the old local installation of Office, the virtualized Office 2010 will start to self-repair.

To summarize the differences between the isolation modes, refer to the table given as follows:

Isolation mode	System elements	Virtual elements
Merged mode	Application can read and modify content.	Modifications will be sandboxed.
WriteCopy mode	Application can read content. Modifications will be sandboxed.	Modifications will be sandboxed.
Full mode	Application cannot read content.	Modifications will be sandboxed.

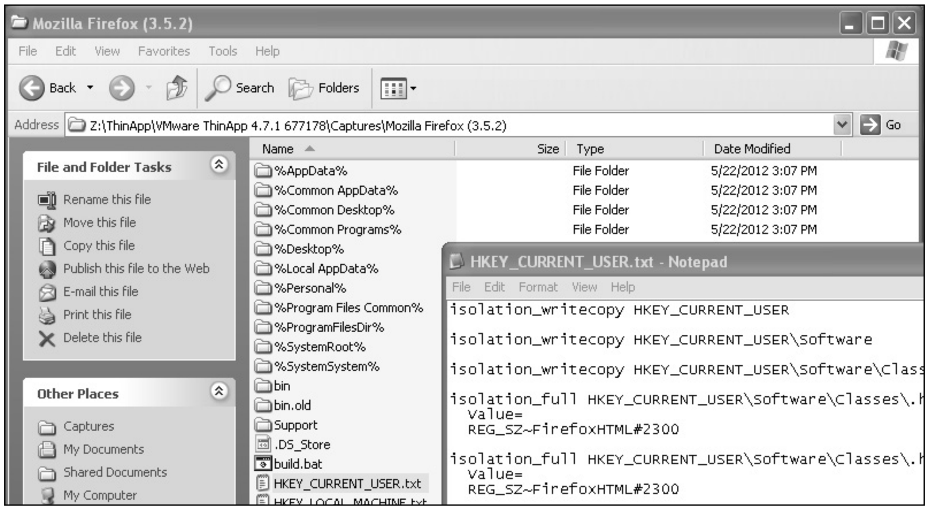
You specify different isolation modes for folders using a configuration file named `##Attributes.ini` located in each folder, as shown in the following screenshot:



```
[Isolation]
DirectoryIsolationMode=WriteCopy
```

The previous screenshot is an example of WriteCopy specified in the **%ProgramFilesDir%** folder.

In the virtual registry you specify isolation modes in front of the registry sub-tree.



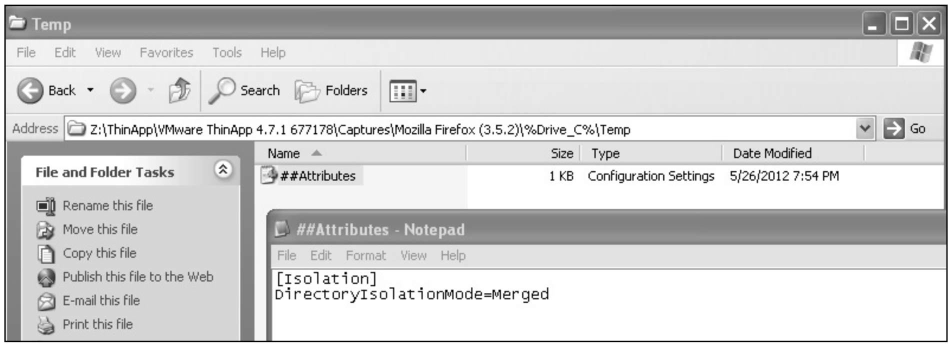
```
isolation_writecopy HKEY_CURRENT_USER

isolation_writecopy HKEY_CURRENT_USER\Software

isolation_writecopy HKEY_CURRENT_USER\Software\Class

isolation_full HKEY_CURRENT_USER\Software\Classes\.h
  Value=
  REG_SZ~FirefoxHTML#2300

isolation_full HKEY_CURRENT_USER\Software\Classes\.h
  Value=
  REG_SZ~FirefoxHTML#2300
```

Let's have a look at some isolation mode examples to help you fully understand isolation modes.

Example 1

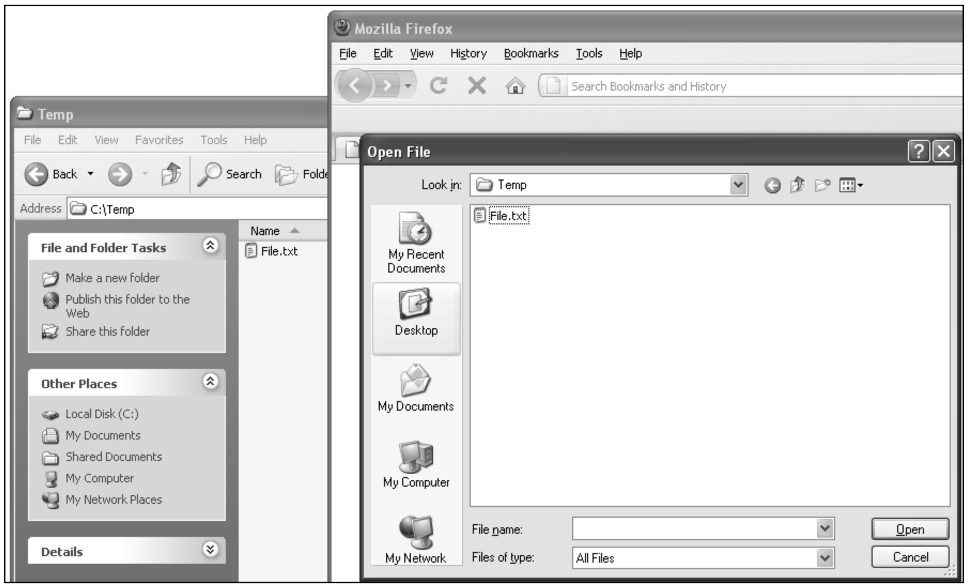
On your physical machine you have a file called `File.txt` within `C:\Temp folder`.

You have the representation of `C:\Temp` within your project folder where you specify either Merged or WriteCopy as an isolation mode, as shown in the following screenshot:


Temp
File Edit View Favorites Tools Help
Back Search Folders
Address Z:\ThinApp\VMware ThinApp 4.7.1 677178\Captures\Mozilla Firefox (3.5.2)\%Drive_C%\Temp
File and Folder Tasks
Rename this file
Move this file
Copy this file
Publish this file to the Web
E-mail this file
Print this file
Name Size Type Date Modified
##Attributes 1 KB Configuration Settings 5/26/2012 7:54 PM
##Attributes - Notepad
File Edit Format View Help

```
[Isolation]
DirectoryIsolationMode=Merged
```

Run your virtualized application (in this example, Mozilla Firefox) and browse to `C:\ Temp`. The application can see the local `File.txt` file, it can open it and read its content.

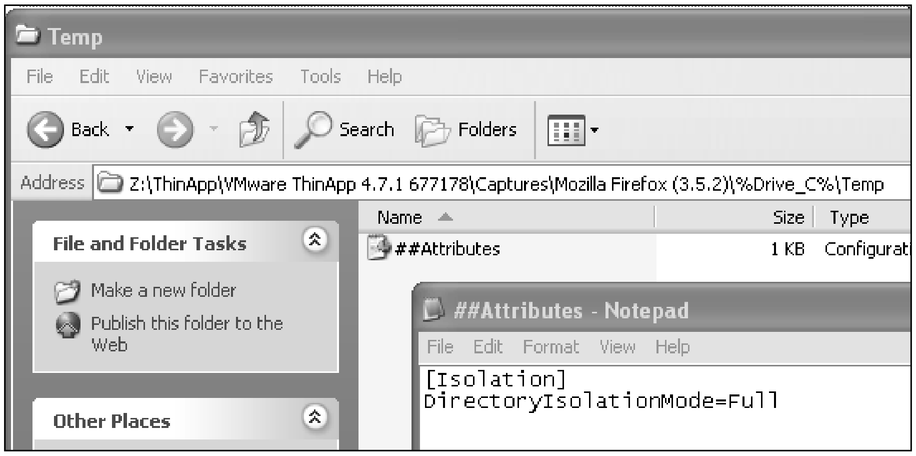


Merged and WriteCopy allows for the virtual environment to read and access native files and registry keys.

Example 2

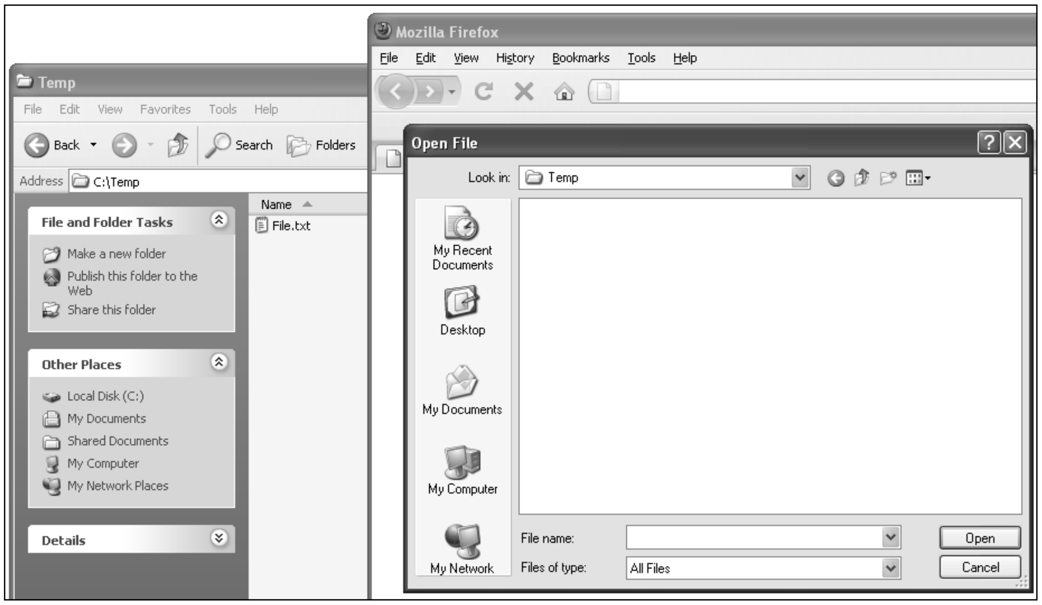
In the same scenario as the previous example, on your native machine you have `C:\Temp\File.txt`.

Within your project folder you've specified Full as the isolation mode in the `C:\Temp` folder.


Temp
File Edit View Favorites Tools Help
Back Search Folders
Address Z:\ThinApp\VMware ThinApp 4.7.1 677178\Captures\Mozilla Firefox (3.5.2)\%Drive_C%\Temp
File and Folder Tasks
Make a new folder
Publish this folder to the Web
Other Places
Name Size Type
##Attributes 1 KB Configurat
##Attributes - Notepad
File Edit Format View Help

```
[Isolation]
DirectoryIsolationMode=Full
```

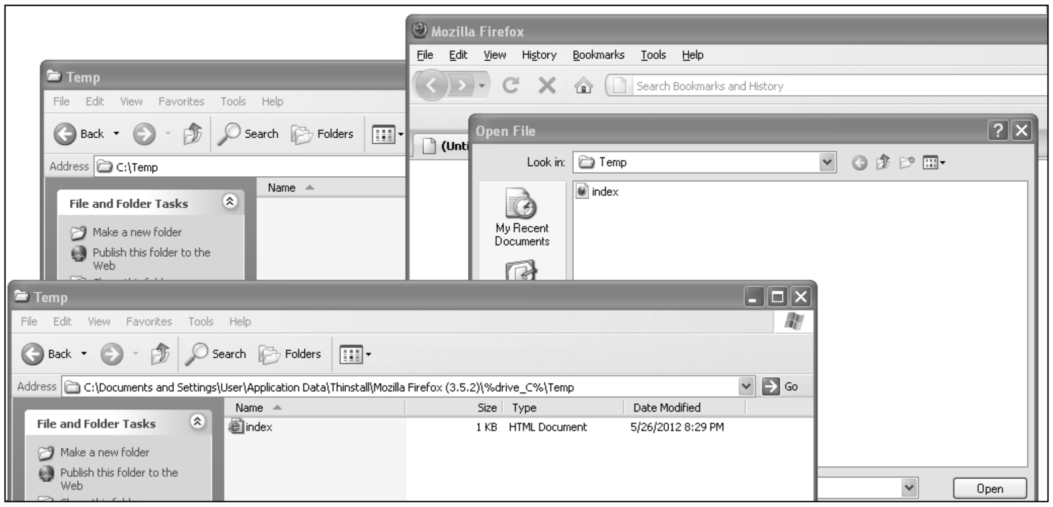
When you run your virtualized Mozilla Firefox and browse to `C:\Temp`, it looks empty, as shown in the following screenshot:



The Full isolation mode hides any native files or registry keys.

Example 3

You are using WriteCopy or Full as your isolation mode on `C:\Temp`. From within your virtualized application you save a file into `C:\Temp`. The file will be sandboxed and your native machine is kept clean. Your virtualized application sees the file as being located in `C:\Temp`.



Address: C:\Documents and Settings\User\Application Data\Thinstall\Mozilla Firefox (3.5.2)\%drive_C%\Temp

Name	Size	Type	Date Modified
index	1 KB	HTML Document	5/26/2012 8:29 PM

WriteCopy or Full will place new files in the sandbox and keep your physical machine clean. Note that there are different associations for the `.html` file between the native environment and the virtualized one. We will discuss the file type registrations later in *Chapter 3*, *Deployment of ThinApp Packages*.

No matter which one of the isolation modes you use, if a virtual file or registry key is modified, the modification will be stored in the sandbox.

When does a file end up in the sandbox? An application can access a file using one of two methods. It can be read-only, which means no modifications can be made to the file and the ThinApp runtime simply passes the file to the application. But if the application opens a file for writing operations, depending on the isolation mode, the ThinApp runtime will first copy the file into the sandbox and then pass the file to the application. This way ThinApp can guarantee that any writing operation needed can be done immediately to the file. This also means you might end up getting files in your sandbox that have not been modified by either the application or the user. It's not very common but nevertheless something you need to be aware of.

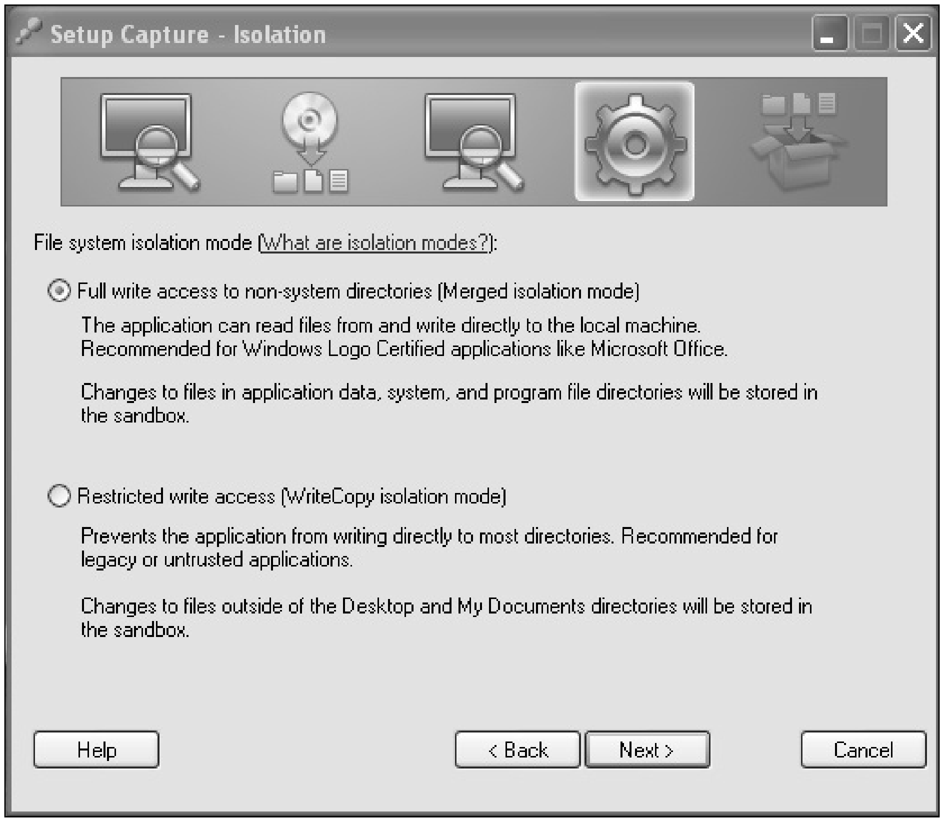
During the capture process, you're asked what default directory isolation mode you want to use. This is of very little technical importance and is mostly a policy decision. I tend to use WriteCopy as my default isolation mode during packaging and tweaking of the project. This way I know that all I do will be sandboxed. Later, when I compile my production version, I change to Merged as the default directory isolation mode. This way, users are less likely to run into the problem of storing a file somewhere without being able to find the file later on since it has been sandboxed. The default isolation mode is specified within your `Package.ini` file using the following parameter:

```
[Isolation]
DirectoryIsolationMode=
```

It's important to point out that you are only asked about the default directory isolation mode and not your default registry isolation mode. The default isolation mode for your registry is always WriteCopy but you can change it within `Package.ini`.

```
[Isolation]
RegistryIsolationMode=Merged
```

As a result of the above, the default isolation mode for the registry would be Merged instead of WriteCopy.



File system isolation mode (What are isolation modes?):

[✓ Full write access to non-system directories (Merged isolation mode)]

The application can read files from and write directly to the local machine. Recommended for Windows Logo Certified applications like Microsoft Office.

Changes to files in application data, system, and program file directories will be stored in the sandbox.

Restricted write access (WriteCopy isolation mode)

Prevents the application from writing directly to most directories. Recommended for legacy or untrusted applications.

Changes to files outside of the Desktop and My Documents directories will be stored in the sandbox.

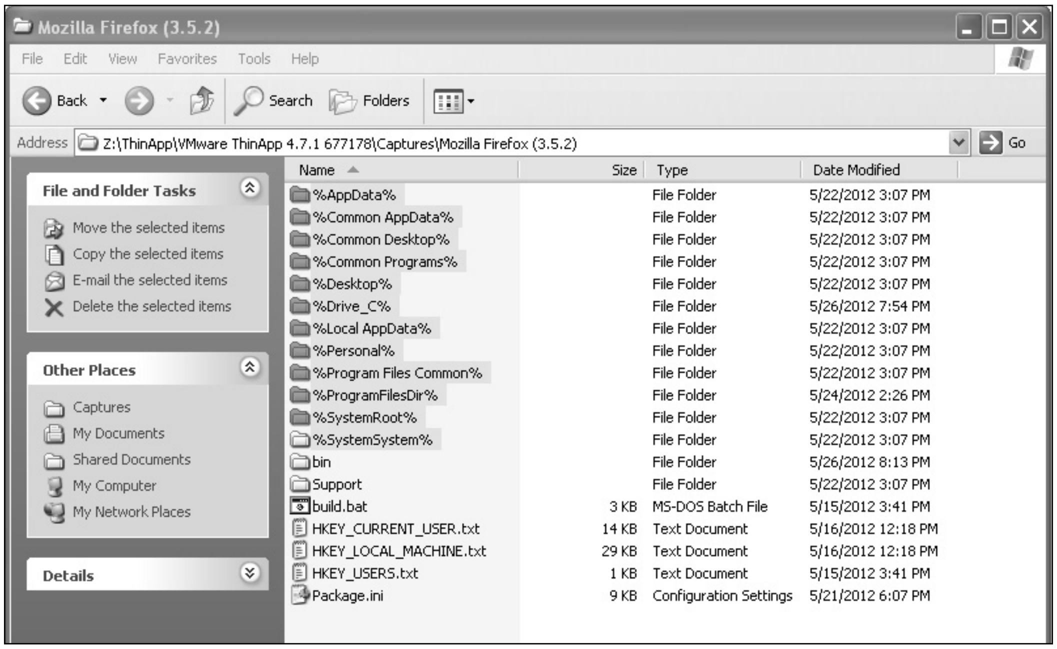
During the capture process you are asked about which default directory isolation mode to use, as shown in the previous screenshot.

The virtual filesystem

ThinApp packagers are working with three different virtual filesystems. The first one is the project folder content. Here, a packager can change the `.ini` files, replace the old `.dll` files with new updated ones, and delete or add any files and folders needed. The second virtual filesystem is created when compiling the project; an exact copy of the filesystem found in the project folder will be compiled into the package as a read-only version of the virtual filesystem. There is no way an end user can modify the content of the package. When using the application, a third version of the filesystem is created in the sandbox: the read and write version of the filesystem.

The complete filesystem known to the virtualized application is a combination of the native (physical) filesystem on the machine, the read-only virtual filesystem stored in the package, and the read and write version stored in the sandbox. If there is a conflict between the native filesystem and the virtual one, the virtual environment will win and the virtual file will be the one presented to the application. If there is a conflict between the sandbox content and the read-only filesystem, then the sandbox content will win.

All folders in the root of the project folder (excluding `Support` and `bin` folders) are in a variable format, for example, `%AppData%`. These variables are called **folder macros** and are similar to variables used in the operating system. Folder macros point to predefined locations. These locations may vary depending on the language of the operating system or which version of the Windows operating system you're running the package on. Some folder macros may use the same names as the ones in the operating system but they are different from one another. Especially when using VBScripts built into the packages, it is important to understand that there is a difference. It's the folder macros that allow a package to be portable between different operating systems.



Address Z:\ThinApp\VMware ThinApp 4.7.1 677178\Captures\Mozilla Firefox (3.5.2)

Name	Size	Type	Date Modified
%AppData%		File Folder	5/22/2012 3:07 PM
%Common AppData%		File Folder	5/22/2012 3:07 PM
%Common Desktop%		File Folder	5/22/2012 3:07 PM
%Common Programs%		File Folder	5/22/2012 3:07 PM
%Desktop%		File Folder	5/22/2012 3:07 PM
%Drive_C%		File Folder	5/26/2012 7:54 PM
%Local AppData%		File Folder	5/22/2012 3:07 PM
%Personal%		File Folder	5/22/2012 3:07 PM
%Program Files Common%		File Folder	5/22/2012 3:07 PM
%ProgramFilesDir%		File Folder	5/24/2012 2:26 PM
%SystemRoot%		File Folder	5/22/2012 3:07 PM
%SystemSystem%		File Folder	5/22/2012 3:07 PM
bin		File Folder	5/26/2012 8:13 PM
Support		File Folder	5/22/2012 3:07 PM
build.bat	3 KB	MS-DOS Batch File	5/15/2012 3:41 PM
HKEY_CURRENT_USER.txt	14 KB	Text Document	5/16/2012 12:18 PM
HKEY_LOCAL_MACHINE.txt	29 KB	Text Document	5/16/2012 12:18 PM
HKEY_USERS.txt	1 KB	Text Document	5/15/2012 3:41 PM
Package.ini	9 KB	Configuration Settings	5/21/2012 6:07 PM

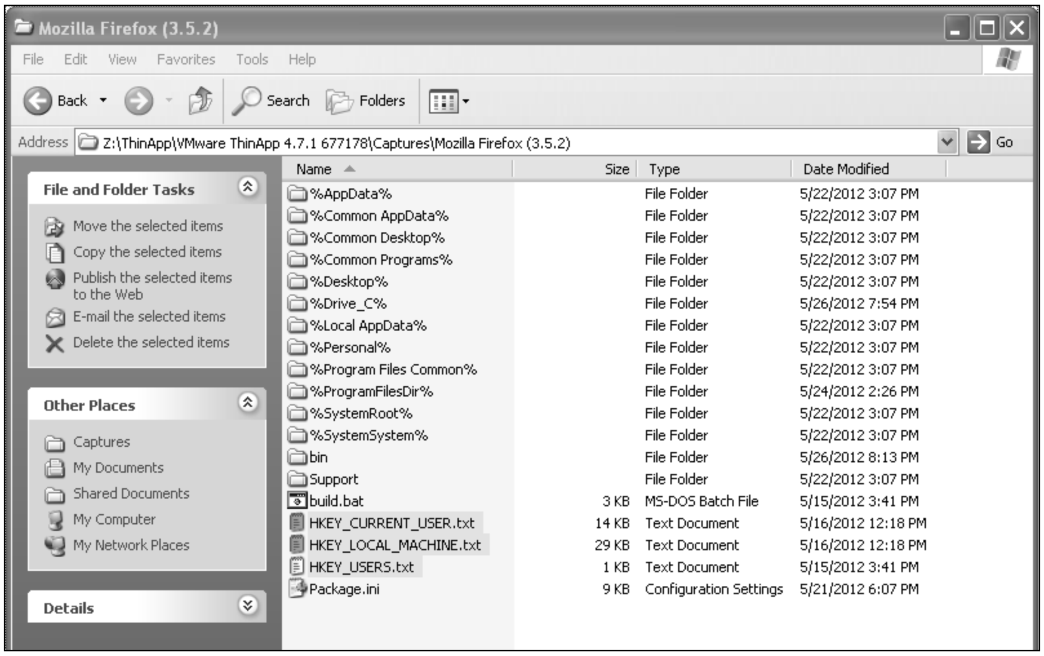
The previous screenshot shows a project folder showing some folder macros.

`%AppData%` refers to the user's roaming profile, which is mostly used to save user settings. Executing a ThinApp package on a Windows XP machine, the `%AppData%` will refer to `C:\Documents and Settings\UserName\Application Data`. Executing the same package on a Windows 7 machine, `%AppData%` will refer to `C:\Users\UserName\AppData\Roaming`. Since ThinApp uses `%AppData%`, the user settings will follow the user no matter which OS the package is executed on.

A list of all folder macros can be found in *References* at the end of this book.

The virtual registry

The virtual registry exists in three versions as well. Within the project folder you will find the virtual registry represented by three clear text files, **HKEY_CURRENT_USER.txt**, **HKEY_LOCAL_MACHINE.txt**, and **HKEY_USERS.txt**.



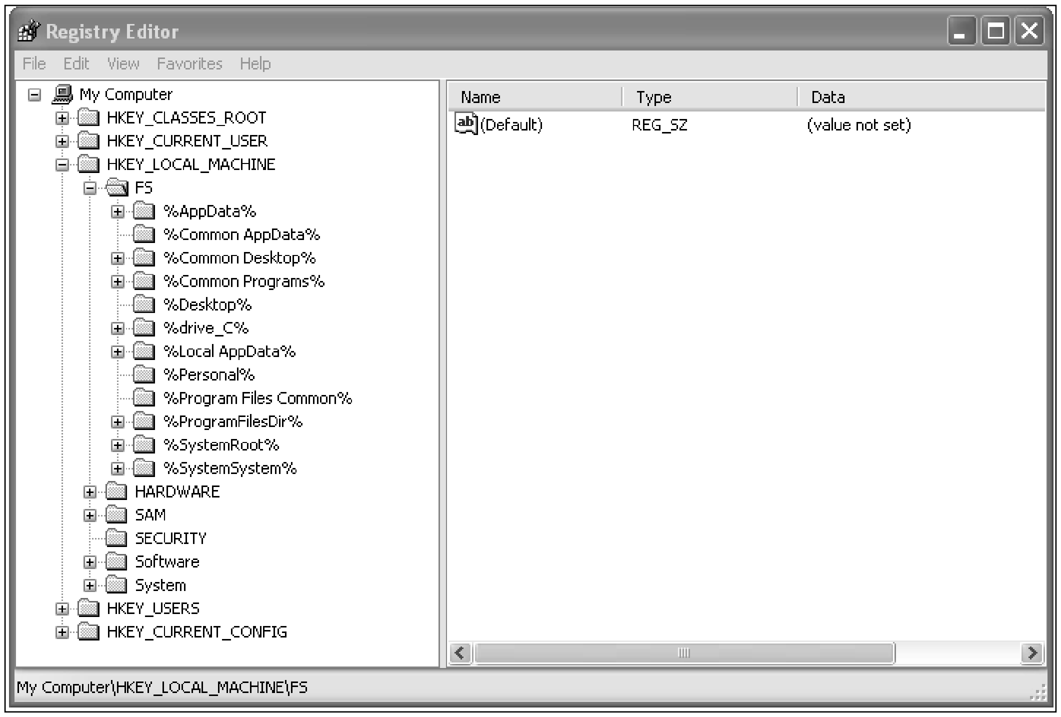
Address Z:\ThinApp\VMware ThinApp 4.7.1 677178\Captures\Mozilla Firefox (3.5.2)

Name	Size	Type	Date Modified
%AppData%		File Folder	5/22/2012 3:07 PM
%Common AppData%		File Folder	5/22/2012 3:07 PM
%Common Desktop%		File Folder	5/22/2012 3:07 PM
%Common Programs%		File Folder	5/22/2012 3:07 PM
%Desktop%		File Folder	5/22/2012 3:07 PM
%Drive_C%		File Folder	5/26/2012 7:54 PM
%Local AppData%		File Folder	5/22/2012 3:07 PM
%Personal%		File Folder	5/22/2012 3:07 PM
%Program Files Common%		File Folder	5/22/2012 3:07 PM
%ProgramFilesDir%		File Folder	5/24/2012 2:26 PM
%SystemRoot%		File Folder	5/22/2012 3:07 PM
%SystemSystem%		File Folder	5/22/2012 3:07 PM
bin		File Folder	5/26/2012 8:13 PM
Support		File Folder	5/22/2012 3:07 PM
build.bat	3 KB	MS-DOS Batch File	5/15/2012 3:41 PM
HKEY_CURRENT_USER.txt	14 KB	Text Document	5/16/2012 12:18 PM
HKEY_LOCAL_MACHINE.txt	29 KB	Text Document	5/16/2012 12:18 PM
HKEY_USERS.txt	1 KB	Text Document	5/15/2012 3:41 PM
Package.ini	9 KB	Configuration Settings	5/21/2012 6:07 PM

When you run **build.bat** the content of these registry files are compiled into the package as read-only versions. When you use the package, the read and write version is created in the sandbox.

You may ask yourself where **HKEY_CLASSES_ROOT** is. **HKEY_CLASSES_ROOT** is a merged view of `HKEY_LOCAL_MACHINE\Software\Classes` and `HKEY_CURRENT_USER\Software\Classes`. **HKEY_CLASSES_ROOT** will be created dynamically during the launch of your package, in a similar way to how the Windows OS generates **HKEY_CLASSES_ROOT** at boot time.

The file database is included in the virtual registry. You can see it while running `regedit.exe` within your virtual environment.



```
My Computer
  HKEY_CLASSES_ROOT
  HKEY_CURRENT_USER
  HKEY_LOCAL_MACHINE
    FS
      %AppData%
      %Common AppData%
      %Common Desktop%
      %Common Programs%
      %Desktop%
      %drive_C%
      %Local AppData%
      %Personal%
      %Program Files Common%
      %ProgramFilesDir%
      %SystemRoot%
      %SystemSystem%
    HARDWARE
    SAM
    SECURITY
    Software
    System
  HKEY_USERS
  HKEY_CURRENT_CONFIG
```

Name	Type	Data
(Default)	REG_SZ	(value not set)

My Computer\HKEY_LOCAL_MACHINE\FS

The ThinApp filesystem database can be viewed when running **Registry Editor** within the virtualized environment.

Application Linking (AppLink)

By default, two virtualized applications are isolated from each other. Application One cannot see files or registry entries virtualized in Application Two's package. The ThinApp feature AppLink lets packagers allow full integration between two or more packages. AppLinking packages will effectively merge the different virtualized environments into one big environment. ThinApp supports up to 250 packages linked together but in reality you will never AppLink that many packages together.

There will be a penalty in the startup time for each AppLink and pretty soon your implementation will become too complicated to maintain and manage. Try to limit the amount of AppLinks between five to ten.

AppLink will allow you to package your main application into one package and any dependencies as AppLink. This allows for a more modular design of your desktop environment. A typical use case is a packaged browser and Java, .NET, Active X, Flash as AppLink packages. AppLink is not limited to dependencies. A packaged Microsoft Office and an application tightly integrating with Office can be AppLinked together. This way it will look like both the applications are locally installed on the client, and full integration between applications is possible.

The package your end users launch first is called the parent package. Any AppLink packages are referred to as child packages. There is no difference between a parent and child package. Both are valid, normal ThinApp packages. A package being a child when Application A is launched can just as easily be a parent when you launch it separately. Adobe Acrobat Reader is an excellent example. It can be launched as a separate application but is often a child to your packaged Internet Explorer.

Let's say you packaged Internet Explorer and Adobe Acrobat Reader in two different packages. If you associate `.pdf` files to your Acrobat Reader Package you will be able to click on a link to a `.pdf` file from within your virtualized Internet Explorer and a separate Acrobat Reader window will be used to display the Acrobat document. If you want Internet Explorer (IE) to use the embedded Acrobat Reader within the IE window you must AppLink the two packages together. This way Internet Explorer will see Adobe Acrobat Reader as locally installed. The registry keys identifying the embedded functionality in IE will be present in the virtual environment.

When you launch a parent package, its virtual environment will load first, and then the child packages' environments will be merged into the active environment. This happens every time you launch the parent package. If you change the content of a child package the new updated environment will be merged upon the next launch of the parent package. This allows individual updates of your packages. The load order is either alphabetic or in the order specified within `Package.ini`.

The following is an example of an alphabetic load order:

```
[BuildOptions]
OptionalAppLinks=C:\Plugins\*.exe
```

The following is an example of a predefined load order:

```
[BuildOptions]
OptionalAppLinks=C:\Plugins\LoadMeFirst.exe; C:\Plugins\LoadMeLast.exe
```

When you configure AppLink, it is important to understand that you have to point to a data container. It is the virtual environment stored in the data container you want to merge into your running environment. So if your AppLink package uses a separate data container, make sure you refer to the correct file extension, that is, `FileName.dat`.

ThinApp supports one parent package being AppLinked to up to 250 child packages. ThinApp also supports many parent packages AppLinking to one child package. A child package can have its own AppLink, and nested loading of AppLinks is fully supported. This means you could end up launching one parent package, AppLinking to one child package that loads one or more child packages of its own. Pretty soon you risk losing the complete overview of where files and registries are located and which isolation mode is active. A file or a registry key may exist in more than one package in your AppLink chain. In order to resolve these conflicts, the ThinApp runtime will use "last loaded". This means if you have `C:\Temp\File.txt` in your parent package and in your child package, the version in your child package will be used. Parent environments are always loaded first and then child environments are loaded in either alphabetic order or in the load order specified within your `Package.ini` file. What about isolation modes? Here the ThinApp runtime uses a different method, wherein the most restrictive mode will win. This means if your parent package has Merged on the folder `C:\Temp`, then make sure not to use any other isolation modes in any of your child packages. Remember that nested packages will be part of the whole AppLink chain as well. Now it's getting clearer why I recommend using only a small number of AppLink packages in a desktop environment design.

An AppLink package is only loaded once per execution. If you have a complex AppLink chain referring to the same child package multiple times, the child package will be merged only once, the first time it is referred to.

AppLink conflict resolution for isolation modes

- *WriteCopy versus Merged, WriteCopy will win*
- *WriteCopy versus Full, Full will win*

AppLink will discard any sandboxes existing for the child package. Let's say you AppLink to Adobe Acrobat Reader. This package might have been used separately and therefore has the user settings stored within its sandbox. Now, when you execute the parent package, the parent sandbox is the only one in use and any settings stored within the Adobe Acrobat's sandbox will not be part of the running environment.

If your child packages have any virtualized services or VBScripts, they will be active when using AppLink. Bear in mind that, starting services may be a time consuming task. AppLinking to a package starting services might therefore add extra time to the launch time.

ThinApp supports the following two flavors of AppLink:

- Optional AppLink
- Required AppLink.

Optional AppLink

When using the dynamic AppLink called `OptionalAppLinks` in `Package.ini`, the package will AppLink to any package available. If no AppLink can be found, the package will happily launch anyway.

Using Optional AppLink offers a true dynamic design of your applications. You don't even have to know upon packaging if you need to AppLink or not. Simply activate `OptionalAppLinks` and you can always add functionality to your package later on.

The following are example configurations:

```
[BuildOptions]
OptionalAppLinks=plugins\*.exe
```

The result of this configuration is that, any package located in the folder called `plugins` relative to the parent package itself will be added.

```
[BuildOptions]
OptionalAppLinks=plugins\*.exe; plugins\*.dat
```

The result of this configuration is that, any package located in the `plugins` folder relative to the parent package will be AppLinked, including separate data containers.

```
[BuildOptions]
OptionalAppLinks=\\ServerName\ShareName\MyAppLinks\Java.dat
```

The result of this configuration is that the Java package located on a network share will be AppLinked.

```
[BuildOptions]
OptionalAppLinks=C:\Program Files\Java 1.6 (VMware ThinApp)\*.exe; C:\
Program Files\Flash (VMware ThinApp)\*.exe
```

The result of this configuration is that, if available, a virtualized Java and locally deployed Flash clients will be AppLinked. This is a very common AppLink configuration used when you deploy ThinApps with the help of MSI and existing deployment tools. We will get back to some different deployment scenarios later.

```
[BuildOptions]
OptionalAppLinks=%HOMEPATH%\*\*.exe
```

This configuration shows that, AppLink supports environment variables and wildcard searches. This example will search one folder deep in the users `%HOMEPATH%` for child packages called `*.exe`.

```
[BuildOptions]
OptionalAppLinks=\\ServerName\ShareName\*\*\*.dat
```

In this configuration, ThinApp will search two folders deep for child packages named `*.dat`. For example, both `\\ServerName\ShareName\AppLinks\Java\java.dat` and `\\ServerName\ShareName\AppLinks\Flash\Flash.dat` will be AppLinked.

Required AppLink

Required AppLink, called `RequiredAppLinks` in `Package.ini`, will deny execution of the package if the AppLink cannot be found. When you use a required AppLink, make sure you specify the whole filename of your child packages. You should not use the wildcard (`*`) since this will effectively disable the required rule set, that is, deny usage of the parent package if AppLink packages are not available.

```
[BuildOptions]
RequiredAppLinks=C:\Program Files\Java 1.6 (VMware ThinApp)\java.exe;
C:\Program Files\Flash (VMware ThinApp)\flash.exe
```

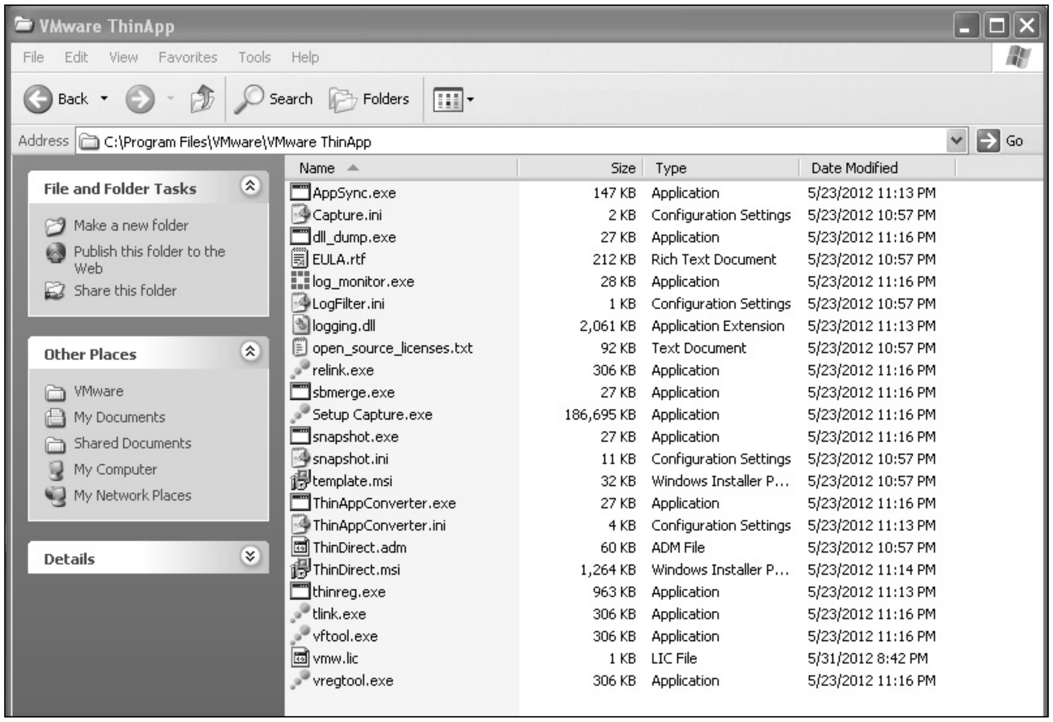
This configuration means that the Java and Flash packages will be AppLinked. If they are not accessible, the user will be denied the ability to run the parent package.

```
[BuildOptions]
RequiredAppLinks=\\ServerName\ShareName\java.exe
```

This configuration means that the Java package located on a network share will be AppLinked. If the Java package is not accessible, the user will be denied the ability to run the parent package.

The ThinApp utilities folder and its content

The ThinApp utilities folder is installed in `C:\Program Files\VMware\VMware ThinApp` by default. Only the ThinApp packagers need access to the ThinApp utilities folder - end users never need access to it. If desired, you can move this folder to a network share and run all the tools from there. The ThinApp utilities are virtualized using ThinApp so the folder is just as portable as any ThinApp package. Placing the folder on a network share makes it easier to access the tools from any machine. Often when packaging, you are using virtual machines and reverting the virtual machines to clean states between each capture. Having the ThinApp utilities folder on a network share will make them easier to maintain. Changing the version of ThinApp used does not require a new snapshot of your virtual machine. Another benefit is that all your packager colleagues can share one and the same ThinApp utilities folder and settings.

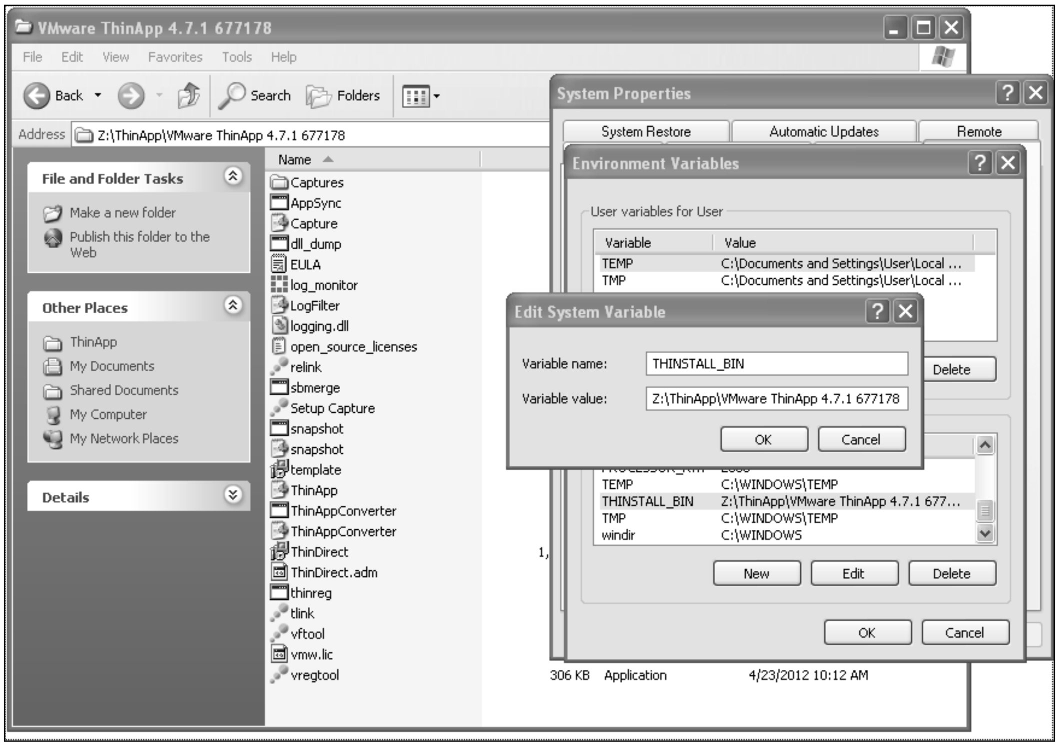


C:\Program Files\VMware\VMware ThinApp

Name	Size	Type	Date Modified
AppSync.exe	147 KB	Application	5/23/2012 11:13 PM
Capture.ini	2 KB	Configuration Settings	5/23/2012 10:57 PM
dll_dump.exe	27 KB	Application	5/23/2012 11:16 PM
EULA.rtf	212 KB	Rich Text Document	5/23/2012 10:57 PM
log_monitor.exe	28 KB	Application	5/23/2012 11:16 PM
LogFilter.ini	1 KB	Configuration Settings	5/23/2012 10:57 PM
logging.dll	2,061 KB	Application Extension	5/23/2012 11:13 PM
open_source_licenses.txt	92 KB	Text Document	5/23/2012 10:57 PM
relink.exe	306 KB	Application	5/23/2012 11:16 PM
sbmerge.exe	27 KB	Application	5/23/2012 11:16 PM
Setup Capture.exe	186,695 KB	Application	5/23/2012 11:16 PM
snapshot.exe	27 KB	Application	5/23/2012 11:16 PM
snapshot.ini	11 KB	Configuration Settings	5/23/2012 10:57 PM
template.msi	32 KB	Windows Installer P...	5/23/2012 10:57 PM
ThinAppConverter.exe	27 KB	Application	5/23/2012 11:16 PM
ThinAppConverter.ini	4 KB	Configuration Settings	5/23/2012 11:13 PM
ThinDirect.adm	60 KB	ADM File	5/23/2012 10:57 PM
ThinDirect.msi	1,264 KB	Windows Installer P...	5/23/2012 11:14 PM
thinreg.exe	963 KB	Application	5/23/2012 11:13 PM
tlink.exe	306 KB	Application	5/23/2012 11:16 PM
vftool.exe	306 KB	Application	5/23/2012 11:16 PM
vmw.lic	1 KB	LIC File	5/31/2012 8:42 PM
vregtool.exe	306 KB	Application	5/23/2012 11:16 PM

Default location of the ThinApp utilities Folder.

If you don't place the ThinApp utilities folder in the default location, you should make sure that you specify an environment variable called **THINSTALL_BIN** pointing to the ThinApp utilities folder. This way the `build.bat` file will find the location of the tools needed while building your project folder. Using the **THINSTALL_BIN** environment variable allows you to have multiple versions of the ThinApp utilities folder present. You can switch between active folders by simply changing the value of the environment variable.



Address Z:\ThinApp\VMware ThinApp 4.7.1 677178

Variable	Value
TEMP	C:\Documents and Settings\User\Local ...
TMP	C:\Documents and Settings\User\Local ...

Variable name: THINSTALL_BIN

Variable value: Z:\ThinApp\VMware ThinApp 4.7.1 677178

Variable	Value
TEMP	C:\WINDOWS\TEMP
THINSTALL_BIN	Z:\ThinApp\VMware ThinApp 4.7.1 677...
TMP	C:\WINDOWS\TEMP
windir	C:\WINDOWS

306 KB Application 4/23/2012 10:12 AM

Specify the location of your ThinApp utilities folder with the help of the **THINSTALL_BIN** environment variable, as shown in the previous screenshot.

Let's have a look at some of the files you'll find within the ThinApp utilities folder. Most of them will be discussed in much more detail later in this book. The following are the files present:

- `AppSync.exe`

  AppSync is one of the built-in update mechanisms within ThinApp. We will cover AppSync more in depth later in this book. Running **AppSync.exe** allows you to specify a package to AppSync and an AppSync URL, where the update is located, providing a more dynamic method of updating the deployed ThinApp packages than the AppSync you can configure using `package.ini`.

  `log_monitor.exe`.

  The **log monitor** is a trace tool used to troubleshoot ThinApp packages.

- `Capture.ini` and `LogFilter.ini`

  These are filter files used to filter what is captured while running the log monitor.

- `relink.exe`

  Relink is used to inject a new runtime, certain settings, and a license key into an existing package without the need to completely rebuild the whole project folder.

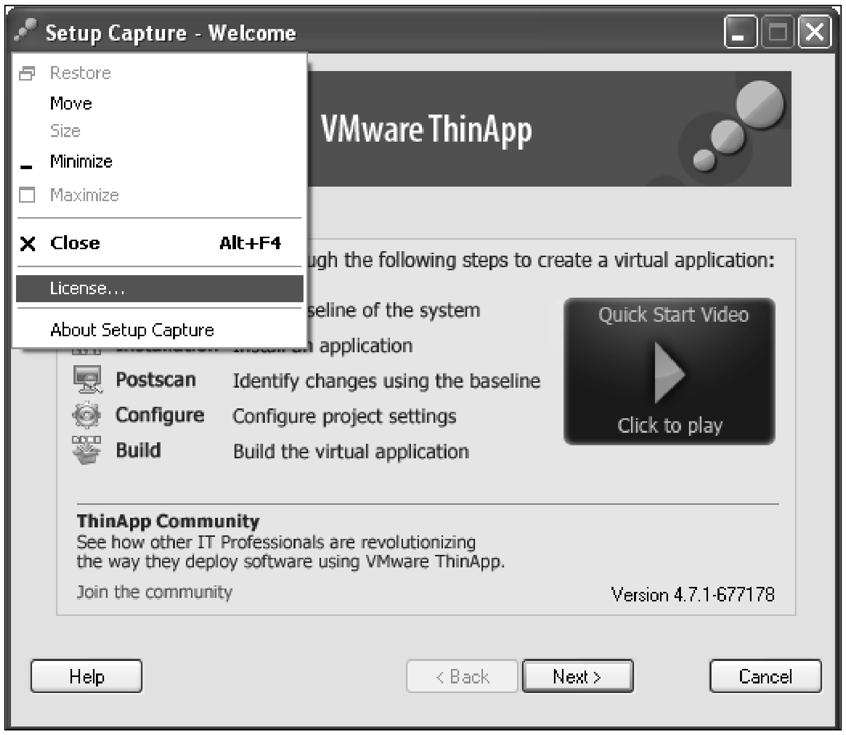
- `sbmerge.exe`

  Sbmerge stands for **sandbox merge**. It is a tool used by packagers to merge the content of a sandbox into an existing project folder. It is a great tool used to apply changes and updates to a project. Running `sbmerge.exe` without any switches will display the `help` file.

- `Setup Capture.exe`

  Setup Capture is the tool used to capture an application installation and create a project folder. Within **Setup Capture** you can specify your license key and if you want to change the license key or "licensed to" name, you must change these within **Setup Capture** and then rebuild your project or run relink on the packages.

You change the license key and "Licensed to" information by launching **Setup Capture**, clicking on the top-left corner and choosing **License**, as shown in the following screenshot:



...ugh the following steps to create a virtual application:

...seline of the system

...n application

Postscan Identify changes using the baseline

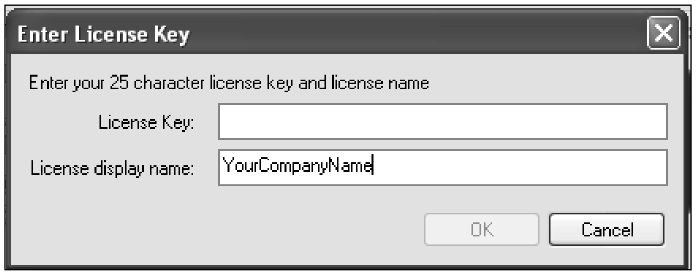
Configure Configure project settings

Build Build the virtual application

ThinApp Community
See how other IT Professionals are revolutionizing the way they deploy software using VMware ThinApp.
Join the community

Version 4.7.1-677178

You are now shown the **Enter License Key** dialog box, as in the following screenshot:



Enter your 25 character license key and license name

License Key:

License display name: YourCompanyName

Type in your license key and licensed display name. Click on **OK** and then **Cancel** in the **Setup Capture** main window. You have now successfully updated the license information. Simply rebuild your project to update your packages or run `relink.exe`.

- `snapshot.exe`

  This is the ThinApp snapshot tool. Running `snapshot.exe` from a command prompt allows you to capture an installation and create a project folder without running **Setup Capture**. Running `snapshot.exe` without any switches will give you the full `help` file. The following procedure will create a project folder using `snapshot.exe`:

  1. Run the command `snapshot.exe c:\PreScan.snapshot`.
  2. Install the application.
  3. Run `snapshot.exe c:\PostScan.snapshot`.
  4. Run `snapshot.exe c:\PreScan.snapshot -SuggestName c:\PostScan.snapshot`.
  5. Run the command `snapshot.exe c:\PreScan.snapshot -Diff c:\PostScan.snapshot c:\ProjectFolder`.
  6. Run `snapshot.exe c:\PreScan.snapshot -SuggestProject c:\PostScan.snapshot c:\OutputTemplate.ini`.
  7. Run `snapshot.exe c:\OutputTemplate.ini -GenerateProject c:\ProjectFolder`.

- `snapshot.ini`

  `Snapshot.ini` is the exclusion list used by **Setup Capture** and `snapshot.exe`. Here you can specify parts of the operating system that should not be scanned during the capturing process. The defaults are implemented to keep your project from capturing unnecessary content. It's not recommended to have an antivirus software installed on your capturing machine, but if policy dictates that you must, you can use the `snapshot.ini` file to exclude locations for the antivirus log files and such. This keeps the changes from being a part of the captured environment and thereby polluting your project folder.

- `template.msi`

  ThinApp can generate an MSI file to simplify the deployment of the ThinApp packages. Using an MSI file will allow the use of any existing deployment tool to distribute ThinApp packages. The MSI files that ThinApp generates are supported by any tool supporting MSI files but are not normal MSI files. ThinApp supports MSI files greater than 2 GB without the use of CAB files. This is accomplished with the help of virtualization within the MSI itself. You cannot use tools such as Orca to modify the MSI properties, since it will destroy the content of the MSI when saved. In order to change the MSI that ThinApp generates, you have to tweak the `template.msi` file instead. Changes applied to the `template.msi` (using Orca or any other tool) file will be incorporated into the MSI files that ThinApp generates.

- `ThinApp.ini`

  `ThinApp.ini` contains the **Setup Capture** user settings, for example, the options to build or skip build at the end of the capture process.

- `ThinAppConverter.exe` and `ThinAppConverter.ini`

  ThinApp Converter is a tool introduced in Version 4.6. With the help of ThinApp Converter and its configuration file `ThinAppConverter.ini`, you can automate the capturing process. This was an early version of automation and is more or less replaced with the tool called **ThinApp Factory**. ThinApp Converter drives the capture process by running virtual machines hosted on ESX or VMware Workstation. `ThinAppConverter.ini` is pretty much the only documentation available for this tool. There are third-party tools using ThinApp Converter to automatically convert installers into ThinApp project format. Quest ChangeBASE is one such tool using ThinApp Converter.

- `ThinDirect.msi` and `ThinDirect.adm`

  ThinDirect is a browser helper you install locally on your client. It will add itself as a browser helper to your local Internet Explorer and allows for automated redirection of URLs to specific packaged browsers. `ThinDirect.msi` is the standalone installer you use to deploy the browser helper. `ThinDirect.adm` is used to add Group Policy Management to your ThinDirect implementation. `ThinDirect.adm` includes five different browsers and 25 different URLs for each one. You can change the amount of supported browsers or URLs simply by editing the file in a text editor.

- `thinreg.exe`

  `Thinreg.exe` is a standalone tool that you can copy to any location. Most of the other tools in the ThinApp utilities must reside within the ThinApp utilities folder to function. `Thinreg.exe` is used to register a package on a client machine, offering the look and feel of a locally-installed application. By registering a package, you can add shortcuts onto your desktop or the **Start** menu and you can register file extensions, protocols, and object types to a package. Run `thinreg.exe` without any arguments for the `help` file.

- `tlink.exe`, `vftool.exe`, and `vregtool.exe`

  `tlink.exe`, `vftool.exe`, and `vregtool.exe` are all used to compile your project folder into a virtualized package. `Build.bat` calls these files. `Vregtool.exe` can also be used to investigate the registry changes located in the sandbox. Running `vregtool.exe` without any switches will show you the `help` file.

Summary

In this chapter we learned the basics of application virtualization, isolation modes, the sandbox, application linking, and we looked at the ThinApp utilities folder.

In the next chapter we will cover the packaging process in more detail.