VMware ThinApp User's Manual

VMware ThinApp 4.0

VMware ThinApp User's Manual
Item: EN-000003-01

You can find the most up-to-date technical documentation on our Web site at:

http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to:

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

About This Book

The *VMware ThinApp User's Manual* provides information about how to install and configure ThinApp (formerly Thinstall). The information includes how to upgrade your software, configure packages, configure isolation modes and sandboxes, use the utilities and tools, configure MSI files, use scripts, and use License Manager.

Intended Audience

This book is for anyone who wants to install, upgrade, or use ThinApp. Typical users are system administrators responsible for the distribution and maintenance of corporate software packages.

Document Feedback

VMware welcomes your suggestions for improving our documentation. If you have comments, send your feedback to:

docfeedback@vmware.com

Technical Support and Education Resources

The following sections describe the technical support resources available to you. You can access the current versions of this book and other books by going to:

http://www.vmware.com/support/pubs

Online and Telephone Support

Use online support to submit technical support requests, view your product and contract information, and register your products. Go to:

http://www.vmware.com/support

Customers with appropriate support contracts should use telephone support for the fastest response on priority 1 issues. Go to:

http://www.vmware.com/support/phone_support.html

Support Offerings

Find out how VMware support offerings can help meet your business needs. Go to:

http://www.vmware.com/support/services

VMware Professional Services

VMware Education Services courses offer extensive hands-on labs, case study examples, and course materials designed to be used as on-the-job reference tools. Courses are available onsite, in the classroom, and live online. For onsite pilot programs and implementation best practices, VMware Consulting Services provides offerings to help you assess, plan, build, and manage your virtual environment. To access information about education classes, certification programs, and consulting services, go to:

http://www.vmware.com/services/.

Introduction to VMware ThinApp

1

VMware ThinApp enables you to package, run, and manage your software applications. ThinApp captures the installation of an application in a single executable file and enables you to install your captured application with no dependencies on the host personal computer (PC).

This manual describes how to use ThinApp to capture and configure applications.

Chapter 2, "Capturing and Configuring Applications," on page 15, describes how to prepare a clean PC or virtual machine, how to use Setup Capture to capture applications, and how to configure your packages.

Chapter 3, "Configuring the Package.ini File," on page 29, describes the `Package.ini` file. This file provides settings you can use to configure your application captures.

Chapter 4, "Introduction to the Sandbox," on page 71, describes the sandbox. The sandbox holds runtime modifications that applications make as they are running.

Chapter 5, "Deploying and Working with Virtual Applications," on page 77, describes how to work with virtual applications that you have captured. The chapter describes how to stream captured applications, how to use captured applications with other system components, how to configure access control with Active Directory, and how to upgrade captured applications.

Chapter 6, "Configuring Isolation Modes," on page 89, describes how you can configure isolation modes to control read and write access for system directories and system registry subkeys. The chapter describes the Merged, WriteCopy, and Full isolation mode options in detail.

Chapter 7, "Using the Virtual File System," on page 95, describes the virtual file system, which receives requests that are directed to the host operating system.

Chapter 8, "Using the Virtual Registry," on page 99, describes how to configure the virtual registry, which stores the settings that are created when you install an application.

Chapter 9, "Using ThinApp Utilities and Tools," on page 109, describes how to use the Log Monitor, sandbox merge (sbmerge), ThinReg, snapshot, DLL dump, and ConfigDump utilities.

Chapter 10, "Generating MSI Files," on page 145, describes how to generate MSI files to deliver captured applications to remote locations and to create shortcuts and file type associations for your captured packages.

Chapter 11, "Using Scripts," on page 149, describes how to create scripts to run custom code before starting a captured application or after an application exits.

Chapter 12, "Troubleshooting," on page 167, describes what you need if you must contact customer support. The chapter also provides application-specific troubleshooting tips.

Capturing and Configuring Applications 2

To perform the procedures in this chapter, you must use the MSI installation file to install ThinApp on your computer.

This chapter includes the following topics:

Supported Operating Systems and Applications

ThinApp supports the following operating systems, and applications:

- 32-bit platforms: Windows NT, Windows 2000, Windows 2000 Server, Windows XP, Windows XPE, Windows 2003 Server, Windows Vista
- 64-bit platforms: Windows XP 64 bit, Windows 2003 64 bit, Windows Vista 64 bit
- 16-bit applications running on 32-bit Windows operating systems

- 32-bit applications running on 32-bit and 64-bit Windows operating systems
- Terminal Server and Citrix Metaframe

In addition, ThinApp supports the following items:

- Single-package support on all platforms
- Automatic migration for Windows NT or Windows 2000 to Windows XP and Windows XP to Windows Vista

ThinApp does not support the following operating systems and applications:

- 16-bit or non-x86 platforms such as Windows CE
- 64-bit applications running on 32-bit or 64-bit Windows operating systems
- 16-bit applications running on 64-bit Windows operating systems

Applications that ThinApp Cannot Capture

ThinApp does not support the capture of all applications and in some cases application functions might be degraded.

The following is a list of software that you typically must deploy using traditional installation technologies:

- Applications requiring installation of kernel-mode device drivers (ODBC drivers work because they are user-mode).
- Products such as anti-virus and personal firewalls.
- Scanner drivers and printer drivers.
- Some VPN clients.

The following sections list some known limitations.

Device Drivers

Applications that require device drivers do not work when they are packaged with ThinApp. You must natively install those device drivers on the host PC. Antivirus, VPN clients, personal firewalls, and disk and volume mounting-related utilities are usually not good candidates for ThinApp to capture. In addition, applications that make their features available by installing printer drivers cannot include a printer driver when ThinApp packages them.

Shell Integration

Some applications that ThinApp captures, such as applications that provide shell integration, might have reduced functions. When they are captured they are no longer integrated with other applications on the system.

Network-Visible DCOM Services

ThinApp isolates COM and DCOM. Applications that install network-accessible DCOM services are accessible on the local PC only by other captured applications running in the same sandbox. ThinApp supports virtual DCOM and COM on the same PC but does not support network DCOM.

Global Hook DLLs

Some applications use the `SetWindowsHookEx` API function to add a DLL to all processes on the host PC. The added DLL can intercept Windows messages. This method captures keyboard and mouse input from other applications. ThinApp ignores requests from applications that use `SetWindowsHookEx` to try to install Global Hook DLLs.

Using a Clean PC for Capturing

VMware recommends using a clean PC to capture application installations because installers skip files that already exist on the PC. If the installer skips files, the ThinApp package does not include them. If that happens, the application might fail to run on other PCs where the files do not exist. Setup Capture can scan the PC file system and registry quickly when it is clean.

A clean PC is a PC that has a Windows operating system installed and nothing else. In a corporate environment where you have a base desktop image, the base desktop image is your clean PC. In such cases, the desktop PC might already have some components and libraries installed (for example .NET 2.0).

When you capture a package on a PC that has .NET 2.0 already installed, .NET 2.0 is not included in the package. The application runs only on PCs that have .NET 2.0 already installed.

Capture the lowest platform you plan to support for your users. In most cases this is Windows 2000 or Windows XP. Most packages captured on Windows XP work on Windows 2000. In some cases, Windows XP includes some DLLs that Windows 2000 does not have. These DLLS are excluded from the package if the application normally installs these DLLs.

After you create the ThinApp package structure, you can overwrite files in the package with newer versions and rebuild the application without the capture process. You can use batch files to compile the application, copy updated files, and build a ThinApp executable file.

Installing VMware Server

When you create a ThinApp project using Setup Capture, VMware recommends using a clean installation of Windows.

The easiest way to set up a clean PC is to create a virtual machine. You can install Windows on the virtual machine and take a snapshot of the entire virtual machine in its clean state. After you use Setup Capture to capture an application's installation, you can restore the snapshot. You can then revert the virtual machine to a clean state, ready to capture a new application.

VMware Server is a free program that you can use to create virtual machines. You can download VMware Server from vmware.com at the following URL:

http://www.vmware.com/download/server/

For information about how to install and configure VMware Server, see the *VMware Server Virtual Machine Guide*. You can also use other VMware products that create virtual machines, such as Workstation.

Using Setup Capture

Use the Setup Capture wizard to create and build your packaged application.

To use Setup Capture

1 Start Setup Capture.

A full capture of an application requires a clean Windows system for the lowest platform you intend to support.

For example, use Windows 2000 if you plan to support Windows 2000, Windows XP, Windows 2003, and Windows Vista.

2 Start Setup Capture from the **Start** menu or by double-clicking `Setup Capture.exe` in your `Program Files\VMware\ThinApp` directory.

3 (Optional) Click **`Advanced Settings`** and select the drives and registry hives to scan and click **`OK`**.

This returns you to the Setup Capture wizard and the pre-scan instructions.

4 Click **Next** to begin the scan.

On a fast, clean PC, the scan process should take approximately 10 seconds for Windows XP.

5 Minimize the Setup Capture wizard and install the applications you want to capture. Then maximize the Setup Capture wizard.

6 When the scan finishes, select your user accessible entry points and click **Next**.

Your entry points are the executable files that start your virtual applications. The entry points you are able to choose from depend on the executables that your captured application creates during installation.

For example, if you install Microsoft Office, you can choose entry points for Word, Excel, and other applications that are installed during a Microsoft Office installation.

7 Select any Active Directory Groups that you want authorized to use the package you are creating. For more information on this setting, see "PermittedGroups" on page 46.

8 Select your sandbox location and click **Next**.

NOTE If you select Network drive / custom, be sure to enter the relative path to the location where you want the sandbox created. For example:

```
\\thinapp\sandbox\application1
```

The sandbox holds runtime modifications that your virtual applications make as they are running. For more information on the sandbox, see Chapter 4, "Configuring the Sandbox," on page 71.

9 Enter the directory where you want to save your package. Your package stores your captured software applications. A default directory is provided for you.

10 If you want to build an MSI package, select the **Build MSI Package** check box and enter your MSI file name. For more information about MSI package settings, see "MSI Generation" on page 62. For more information on setting up an MSI database, see "Building an MSI Database" on page 145.

11 Select your compression option and click **Next**. The compression settings you choose correspond with the compression parameters you set in the `Package.ini` file. For more information, see the compression parameters listed in "Isolation and Virtualization Granularity" on page 30.

12 Select **Merged** or **WriteCopy** isolation mode. The isolation mode determines which files and registry keys are visible and written by the virtual application you create. The default isolation mode is merged.

13 In the final dialog box of the Setup Capture wizard, select from the following three options:

- Click **Browse Project** to look at the project files in Windows Explorer.
- Click **Build Now** to create an executable or MSI file containing the files you installed during the Setup Capture process. The build output is shown in the display box below the Build Now button.
- Click **Finish** to complete your work on the project.

  This completes the Setup Capture process.

ThinApp selects one primary executable to host the ThinApp runtime, application registry data and files. The other executable files are shortcuts to the main executable and can be deleted prior to distribution to users if they are not needed.

Executables are selected based on which shortcuts were installed by the application during installation, so you can delete unimportant shortcuts before completing the post-install Setup Capture scan on the capture machine.

If you want a different primary executable after Setup Capture is completed, you can edit the `Package.ini` file. Shortcut executables cannot be run unless the primary executable is located in the same directory. Shortcut executables and the primary executables share a common virtual registry and file system, and can interact with each other.

Editing the Package.ini File

The Setup Capture wizard makes modifications to your `Package.ini` file. The `Package.ini` file can be found in your package directory. For example,

```
C:\Program Files\VMware\VMware ThinApp\Captures\my_virtual_app
```

You can edit the `Package.ini` file to customize the output of the Setup Capture process. For example, for Acrobat Reader you can make `AcroRd32.exe` the only executable.

In the `Package.ini` file, find:

```
[BuildOptions]
OutDir=bin
SandboxName=reader7
```

```
[AdobeDownloadManager.exe]
Source=%ProgramFilesDir%\Common
Files\Adobe\ESD\AdobeDownloadManager.exe
ReadOnlyData=bin\Package.ro.tvr

[AcroRd32.exe]
Source=%ProgramFilesDir%\Adobe\Acrobat 7.0\Reader\AcroRd32.exe
Shortcut=AdobeDownloadManager.exe

[reader_sl.exe]
Source=%ProgramFilesDir%\Adobe\Acrobat 7.0\Reader\reader_sl.exe
Shortcut=AdobeDownloadManager.exe
```

Replace it with:

```
[BuildOptions]
OutDir=bin
SandboxName=reader7

[AcroRd32.exe]
Source=%ProgramFilesDir%\Adobe\Acrobat 7.0\Reader\AcroRd32.exe
ReadOnlyData=bin\Package.ro.tvr
```

This change results in a virtualized application that can be run directly from a network share on restricted accounts without installation or modification to local PCs. The captured executable is read only and unmodifiable, so it can be placed in a central location where it is shared by multiple users.

For descriptions of the parameters contained in the `Package.ini` file, see Chapter 3, "Configuring the Package.ini File," on page 29.

A Real Windows Service

Real Windows services have the following features:

- They are packaged as a separate captured executable files.
- They run on boot-up.
- They run under the account specified by the service.
- They are directly visible to the control panel services manager.
- They can be started and stopped by any application.
- They require global system registry changes.

Running Captured Executables as Real Windows Services

You might want to run some applications with real Windows services at system boot up rather than when an application is started by a user.

To install a captured packaged executable as a real service

1 Remove virtual registry keys from your ThinApp project relating to services.

2 Add real system registry keys for the associated service that point to your ThinApp packaged executable.

To use Apache Web server

1 After capture, delete all service registry keys from `HKEY_LOCAL_MACHINE.txt HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services`.

2 On the target machine where you want to run the service, write real registry keys where ImagePath points to your captured executable, such as this:

```
c:\path_to_captured_apache\httpd.exe –k runservice
```

You can write the registry values using a `.reg` file or as part of an installation process.

3 You might need to reboot before the service shows up in the service control panel (**Start>Run>services.msc**). Once it appears, you can test starting and stopping the service directly by using the control panel.

NOTE Rebooting is not required to use the service. Rebooting causes the service control panel to refresh its list of available services.

Delete the following registry keys from `HKEY_LOCAL_MACHINE.txt` for the Apache Web server.

```
isolation_full HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Apache2.2
Value=Type
REG_DWORD=#10#00#00#00
Value=Start
REG_DWORD=#02#00#00#00
Value=ErrorControl
REG_DWORD=#01#00#00#00
Value=ImagePath
REG_EXPAND_SZ~"%ProgramFilesDir%\Apache Software
Foundation\Apache2.2\bin\httpd.exe" –k runservice#2300
Value=DisplayName
REG_SZ~Apache2.2#2300
Value=DependOnService
REG_MULTI_SZ~Tcpip#2300Afd#2300#2300
```

```
Value=DependOnGroup REG_MULTI_SZ=#00
Value=ObjectName
REG_SZ~LocalSystem#2300
Value=Description
REG_SZ~Apache/2.2.3 (Win32)#2300
isolation_full
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Apache2.2\Parameters
Value=ConfigArgs REG_MULTI_SZ~–f#2300%ProgramFilesDir%\Apache Software
Foundation\Apache2.2\conf\httpd.conf#2300–d#2300%ProgramFilesDir%\Apache
Software Foundation\Apache2.2\.#2300#2300

isolation_full
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Apache2.2\Security
Value=Security
REG_BINARY=#01#00#14#80#90#00#00#00#9c#00#00#00#14#00#00#00#30#00#00#00#02#00
#
1c#00#01#00#00#00#02#80#14#00#ff#01#0f#00#01#01#00#00#00#00#01#00#00#00#00
#02#
00#60#00#04#00#00#00#00#00#14#00#fd#01#02#00#01#01#00#00#00#00#05#12#00#00
#00
#00#00#18#00#ff#01#0f#00#01#02#00#00#00#00#00#05#20#00#00#00#20#02#00#00#00#0
0#14
#00#8d#01#02#00#01#01#00#00#00#00#00#05#0b#00#00#00#00#00#18#00#fd#01#02#00#0
1#0
2#00#00#00#00#00#05#20#00#00#00#23#02#00#00#01#01#00#00#00#00#05#12#00#00#
00# 01#01#00#00#00#00#05#12#00#00#00

isolation_full
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Apache2.2\Enum
Value=0
REG_SZ~Root\LEGACY_APACHE2.2\0000#2300
Value=Count
REG_DWORD=#01#00#00#00
Value=NextInstance
REG_DWORD=#01#00#00#00
```

Application PATH and Environment Variables

Setup Capture automatically captures system and user environment variables. These changes occur indirectly by capturing and comparing the registry. If an application needs a specific environment variable to run correctly, you can make environment variable changes prior to completing the Setup Capture process. Typically the application's installer does this automatically.

To manually add environment variables to ThinApp packages during the Setup Capture process

1 Select **`Control Panel>System>Advance>Environment Variables`** You can add new variables to either system or user.

Windows stores environment variables on a system-wide and user-specific basis. The files are stored in the following locations:

- System-wide environment variables:

  ```
  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session
  Manager\Environment
  ```

- User-specific environment variables:

  ```
  HKEY_CURRENT_USER\Environment
  ```

For packages running under ThinApp, any environment variables captured in these two locations are applied for all applications running in the same sandbox. Per-user values take precedence over per-machine values.

2 Manually edit `HKEY_LOCAL_MACHINE.txt` and `HKEY_CURRENT_USER.txt` to add environment variables after the capture has completed. For example, you can append the following text to the end of `HKEY_LOCAL_MACHINE.txt` to add a CLASSPATH environment variable that contains the string `c:\mypath`

```
isolation_writecopy
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session
Manager\Environment
Value=CLASSPATH
REG_SZ~c:\mypath#00
```

NOTE The `PATH` variable is ignored by ThinApp, instead a new variable `VirtualPath` is used.

PATH Variable

The `PATH` environment variable is treated differently because typically you do not want to completely replace the local system `PATH` variable, but instead prepend specific values. To accomplish this, Setup Capture uses a special value called `VirtualPath` to store the portions of the PATH variable that changed during setup capture. This `VirtualPath` is prepended to the local system `PATH` during program execution.

Packaging .NET Applications

The following sections describe packaging .NET applications.

Precaptured .NET

The following guidelines are targeted at software developers and IT administrators who want to deploy a .NET application without installing the .NET Framework.

You can do the following with precaptured versions of the .NET Framework:

- Build a `cmd.exe` application that is capable of running .NET applications from the host PC even though .NET might not be installed.
- You can add your application files into the precaptured .NET project and build a single executable file that runs without installation of .NET.

Use this information only for .NET applications that do not require installation of COM components or third-party components requiring registry entries.

- To include Crystal Reports in your package, include them in your capture.
- To include COM components in your package, include them in your capture.
- To include customized registry settings, include them in your capture.

Professional Setup

Use professional setup if you are developing your own application. This procedure enables you to copy your updated executable files and DLLs into an existing project and invoke build.bat without having to use Setup Capture every time.

To generate a single executable that runs without cmd.exe

1 Copy your .NET application, DLL files, and data files to `c:\project\%ProgramFilesDir%\myapp`

2 Edit `c:\project\Package.ini`.

```
Source=%SystemSystem%\cmd.exe
Source=%ProgramFilesDir%\myapp\myapp.exe
```

(`myapp.exe` is your application's executable)

3 Run `c:\project\build.bat`.

Reducing Package Size in .NET Applications

Whether you download precaptured .NET Framework projects or you capture the .NET installation yourself, the size of packages is important. Here are a few tips for reducing your package size:

- Turn on compression – The easiest way to generate smaller packages is to enable compression in your `Package.ini` file. To do this, replace `Compression=None` with `Compression=Fast` in the `Package.ini` file.
- Delete any `MSI` files left behind by the installer – Many applications leave behind their original installer in `MSI` format so that they can reinstall if they become corrupted, or if any features of the application were installed using "install on demand." You can safely delete the `MSI` files captured during the installation of the .NET Framework.
- Delete native images – .NET uses a technique called just-in-time compilation to convert IL byte code into native processor code at runtime. During installation, native image files are generated to the hard drive. These files can double or triple your package size. By deleting them, you can make your packages much smaller. In .NET 1.1, this process is very fast and it can occur at startup very quickly. In .NET 2.0 and higher, the native image compilation is significantly slower. Startup time slows if you decide to delete native images because these images are generated in memory rather than be pre-existing in your ThinApp package.

  You can delete native images by deleting the following directory:

  ```
  %SystemRoot%\assembly\NativeImages*.*
  ```

##Attributes.ini

This file controls per directory settings.

```
[Isolation]
DirectoryIsolationMode=WriteCopy | Merged | Full
```

Example

```
DirectoryIsolation=WriteCopy
```

If isolation is not specified, it is inherited from parent directory.

```
[Compression]
CompressionType=None | Fast | Small
BlockSize=64k | 128k | 256k | 512K | 1M
BlockType=Fixed | Rolling

[FileList]
ExcludePattern=*.tmp,*.obj,FullName
```

Plug-Ins and Add-Ins

Many applications have the ability to install plug-ins or load external components. Plug-ins can be packaged together or separately from an application.

To create a single captured application that contains both the main application and its plug-ins, install the plug-ins during the capture process after you have captured your application. The virtualized application can load and use the add-ins as if they are installed on the host PC.

Alternatively, you can use the Application Link feature to link a virtual application to its plug-ins or add-ins. For more information, see "Application Link" on page 51.

Using Setup Capture to capture an application and its plug-ins or add-ins might require specific knowledge of how the application works and adjustments to the package to enable a virtualized application to see plug-ins installed on the host PC. For example, applications commonly load plug-ins based on specific registry values or file system locations.

For default projects created by Setup Capture, some registry subtrees and directories are set up with full isolation. Full isolation prevents the application from detecting host PC registry keys or files so the application might not see plug-ins that exist on the host PC. If you are not sure which registry subtrees or directories are used to locate and load plug-ins, Log Monitor can help you find that information. By changing the isolation mode from Full to WriteCopy or Merged, you enable the application to see and load plug-ins from the host PC.

You can use a virtualized `cmd.exe` file to start a system-installed application. The system-installed application runs in the virtualized environment. The application can detect and load your virtualized plug-ins. You can also start the system-installed application by running the `cmd.exe /c c:\myapp.exe` file.

Large Packages

ThinApp supports packages of unlimited size. The Windows shell (the `explorer.exe` file) on some Windows platforms has limitations on how it displays icons, and might not display the icon for an executable file that reaches a certain size.

The `Explorer.exe` file tries to map the entire executable file to memory so that it can access the icon resources. If the virtual memory address space available is insufficient, this file mapping operation fails. Typically, an application has a maximum of 2GB of address space. However, DLLs, the process heap, and other objects break up the virtual address space, so a smaller maximum block is actually available. On most computers, Windows XP does not display icons for executable files larger than 600MB. Windows Vista does not have this limitation.

When building a large package, create the following two files:

- One large file containing the package data
- One small file containing icon information

For example, consider the following `Package.ini` file:

```
------------------------------------------------
[LargeApplication.exe]
Source=%ProgramFilesDir%\LargeApplication.exe
ReadOnlyData=bin\Package.ro.tvr
------------------------------------------------
```

To work around the problem, change the contents of the file to:

```
------------------------------------------------
[LargeApplication.dat]
Source=%ProgramFilesDir%\LargeApplication.exe
ReadOnlyData=bin\Package.ro.tvr

[LargeApplication.exe]
Source=%ProgramFilesDir%\LargeApplication.exe
Shortcut=LargeApplication.dat
------------------------------------------------
```

Configuring the Package.ini File

3

The `Package.ini` file contains parameters that configure your captured applications during the build process. By editing the contents of the `Package.ini` file, you can configure the isolation mode and several build options, including MSI settings, Application Link, Application Sync, user-specified entry points for your virtual applications, and other general settings.

The `Package.ini` file contains one BuildOptions and one or more individual Application.exe sections.

You set many of your `Package.ini` file settings when you use the Setup Capture wizard to capture an application. For more information, see Chapter 2, "Capturing and Configuring Applications," on page 15.

The BuildOptions apply to all applications, and any options set in this section are inherited by individual applications, unless the application section specifically overrides these settings.

The Application.exe parameters provide the application entry points that you create during the build process.

This chapter describes the parameters of the `Package.ini` file and contains the following sections:

- "Application Sync" on page 58
- "MSI Generation" on page 62
- "Sandbox Control" on page 66

Isolation and Virtualization Granularity

The following sections describe the isolation and virtualization granularity parameters.

ChildProcessEnvironmentDefault

`ChildProcessEnvironmentDefault`—Determines if child processes are run in the virtual environment by default.

ThinApp's default behavior is to create all child processes inside the virtual environment. This option allows you to change ThinApp's behavior so that new child processes are created outside of the virtual environment.

If certain processes need to be created outside of the virtual environment, specify them in the `ChildProcessEnvironmentException` attribute.

The following example specifies that default behavior is to create child processes as virtual (default):

```
[BuildOptions]
ChildProcessEnvironmentExceptions=AcroRd.exe;notepad.exe
ChildProcessEnvironmentDefault=Virtual

Specify that default behavior is to create child processes outside of the
virtual environment
[BuildOptions]
ChildProcessEnvironmentDefault=External
```

DirectoryIsolationMode

`DirectoryIsolationMode`—Controls default isolation mode for directories in package.

`DirectoryIsolationMode` controls the default isolation mode for the package. This setting applies to any directories that do not have an explicitly specified setting. The default setting is `Merged`.

For example, consider a package that looks like the following:

```
ThinAppProject
Package.ini
%ProgramFilesDir%\MyApp\##Attributes.ini
```

The `Package.ini` file sets the default isolation mode for the project, but individual `##Attributes.ini` files might change the isolation mode for specific directories and their children. Any directories that are not specified, such as `C:\myfolder` inherits the isolation mode from the `Package.ini` file. Directories that are created under `Program Files\myapp` inherits the isolation mode from the `##Attributes.ini` file.

Following are examples of this parameter:

WriteCopy isolation allows the application to read from the host PC but not write to it (default).

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

Merged isolation allows the application to write to any location on the PC, except where the package specifies otherwise

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

ExternalCOMObjects

`ExternalCOMObjects`—Controls whether a specific COM object CLSID is created by ThinApp or by Windows.

By default, ThinApp creates all COM objects inside of the virtual environment instead of Windows. COM supports out-of-process (.exe) servers as well as service-based COM objects. If an application can create such COM objects on the host PC and cause these COM objects to modify the host PC, then the integrity of the host PC cannot be assured. However, if ThinApp executes out-of-process and services-based COM objects inside of the virtual environment, all changes made by the COM objects are stored in the sandbox.

For example, the following instructs ThinApp to execute 2 COM objects outside of the virtual environment if they are created by the application.

```
[BuildOptions]
ExternalCOMObjects={8BC3F05E-D86B-11D0-A075-00C04FB68820};{7D096C5F-AC08-4F1F
-BEB7-5C22C517CE39}
```

ExternalDLLs

`ExternalDLLs`—Force some DLLs to be loaded by Windows.

By default, ThinApp determines whether it loads DLLs itself or passes the loading on to Windows. If the DLL is located in the virtual file system, ThinApp loads the DLL itself. In some circumstances, it is required to have Windows load the DLL, even if it is in the virtual file system. An example of this is a DLL that is injected in other processes, using a mechanism known as Windows hooks. For hooks to work, the DLL implementing the hook must be available on the host file system and be loaded by Windows. When you specify a DLL in ExternalDLLs, the DLL is extracted from the virtual file system into the sandbox and Windows is instructed to load it from there.

Note that the usefulness of this option is limited. If the DLL depends on other DLLs that are located in the virtual file system Windows cannot load it.

For example, this instructs ThinApp to pass loading of `inject.dll` and `injectme2.dll` on to Windows.

```
[BuildOptions]
ExternalDLLs=inject.dll;injectme2.dll
```

IsolatedMemoryObjects

`IsolatedMemoryObjects`—List specific shared memory objects to isolate from other applications.

Shared memory objects are created by applications using `CreateFileMapping` and `OpenFileMapping`. Shared memory objects can be named or anonymous. When the objects are named, they are visible to other applications running in the same user account. Sometimes, it is desirable to isolate shared memory objects so that virtual applications cannot see system objects and vice versa. By default, ThinApp only isolates shared memory objects used by embedded Internet Explorer instances, because a conflict between `explorer.exe` and `iexplore.exe` when they map sandboxed files. You can use this option to isolate additional named shared memory objects so they are visible only to other virtual applications using the same sandbox.

`IsolatedMemoryObjects` accepts a list of entries that are separated using the semicolon `character`. Each entry can have wildcard characters `* and ?` to match variable patterns.

For example, isolate two shared memory objects, matching anything with "outlook" in the name and one matching exactly "My Shared Object."

```
[BuildOptions]
IsolatedMemoryObjects=*outlook*;My Shared Object
```

IsolatedSynchronization

`IsolatedSynchronizationObjects`—List specific synchronization objects to isolation from other applications

Windows has several different named Synchronization objects:

- `Mutex`, accessed using OpenMutex and CreateMutex
- `Semaphore`, accessed using OpenSemaphore and CreateSemaphore
- `Events`, accessed using OpenEvent and CreateEvent

By default, ThinApp does not isolate synchronization objects. You can specify a list of synchronization objects to isolate from other applications not running in the same virtual namespace. A namespace is defined by the location of the application's sandbox. If two applications share the same sandbox path, they have the same namespace for isolated synchronization objects. If two applications have the same sandbox name, but the path to the sandbox is different, the applications have separate namespaces.

`IsolatedSychronizationObjects` accept a list of entries that are separated using the semicolon character. Each entry can have wildcards characters * and ? to match variable patterns.

For example, isolate two synchronization objects, matching anything with "outlook" in the name and one matching exactly "My Shared Object."

```
[BuildOptions]
IsolatedSychronizationObjects=*outlook*;My Shared Object
```

RegistryIsolationMode

`RegistryIsolationMode`—Controls default isolation mode for registry keys in package.

`RegistryIsolationMode` controls the default isolation mode for the package. This setting applies to any registry keys that do not have an explicitly specified setting.

For example, this enables the application to read keys from the host PC but not write to it (default). If no registry isolation mode is specified in the `Package.ini` file, the default value will be WriteCopy.

```
[Isolation]
RegistryIsolationMode=WriteCopy
```

Allows the application to write to any key on the PC, except where the package specifies otherwise.

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

VirtualizeExternalOutOfProcessCOM

`VirtualizeExternalOutOfProcessCOM`—Controls whether external out-of-process COM objects are run in the virtual environment.

Captured applications can create COM objects that are registered in the virtual environment, as well as COM objects from the host system.

This option determines how to treat out-of-process COM objects that are not part of a ThinApp package and not registered in the virtual registry. By default, ThinApp executes external out-of-process COM objects in the virtual environment, so such COM objects cannot modify the host PC. If you run into a compatibility issue with an external COM object running in the virtual environment, this option can be used to allow such objects to be created by and run on the host system. If you want to run only specific COM objects outside of the virtual environment, you can list each COM object's CLSID explicitly using ExternalCOMObjects.

For example, this directs ThinApp to execute all external out-of-process COM objects in the system context, not in the virtual environment.

```
[BuildOptions]
VirtualizeExternalOutOfProcessCOM=0
```

This instructs ThinApp to execute all external out-of-process COM objects in the virtual environment (default).

```
[BuildOptions]
VirtualizeExternalOutOfProcessCOM=1
```

General Options

The following sections describe the general options parameters.

AddPageExecutePermission

`AddPageExecutePermission`—Used to fix applications that don't work in DEP environments.

Windows XP SP2, Windows Server 2003, and above have a feature called Data Execution Prevention that is designed to protect against some security exploits that occur with buffer overflows. Because this feature causes a number of compatibility issues, it is turned off by default on XP SP2 and it is possible to use an machine-specific opt-in or opt-out list of the applications to apply DEP protection to. Opt-in and opt-out

policies can be difficult to manage when a large number of machines and applications are involved. This option instructs ThinApp to add Execution Permission to pages allocated by an application so that it can run on machines that have DEP protection enabled, without having to modify the opt-out list.

For example:

```
[BuildOptions]
;Disable some Data Execution protections for this particular application
AddPageExecutionPermission=1

[BuildOptions]
;Do not change DEP protections (default)
AddPageExecutionPermission=0
```

AllowUnsupportedExternalChildProcesses

`AllowUnsupportedExternalChildProcesses` – When set to 0, an attempt by the virtualized application to create a child MS-DOS or 64-bit process will fail. When set to non-zero (default), such a process will be created natively outside of the virtual environment. If this value is not specified, the default is to allow unsupported external processes.

For example:

The default setting allows MSDOS and 64-bit applications to run outside the virtual environment:

```
[BuildOptions]
AllowUnsupportedExternalChildProcesses=1
```

This setting enables you to execute 64-bit child process tasks on applications when they are running on 64-bit systems. Running the print spooler is an example of a 64-bit child process task.

Use the following setting to block MSDOS and 64bit child processes from being spawned outside of virtual environment:

```
AllowUnsupportedExternalChildProcesses=0
```

AutoShutdownServices

`AutoShutdownServices`—Controls whether to automatically shutdown virtual services when the last non-service process exits.

By default, ThinApp automatically shuts down virtual services when the last non-service-based child process exits. This option instructs ThinApp to keep virtual services running even when all other processes exit. This option does not have any effect on non-virtual services.

Following are examples of this parameter:

Keep virtual services running when the application exits.

```
[BuildOptions]
AutoShutdownServices=0
```

Stop virtual services when the last non-service application exits (default).

```
[BuildOptions]
AutoShutdownServices=1
```

AutoStartServices

`AutoStartServices`—Controls whether to automatically start virtual service when the first application starts.

By default, ThinApp automatically starts virtual services that were installed with the startup type of Automatic. The virtual services are started when the first parent process is executed by the user. This option can be used to disable auto-starting of virtual services.

Following are examples of this parameter:

Do not start virtual services.

```
[BuildOptions]
AutoStartServices=0
```

Start virtual services when first process is launched (default).

```
[BuildOptions]
AutoStartServices=1
```

BlockSize

`BlockSize`—Controls the size of compression blocks used when compressing files during build.

Using a larger block size can achieve higher compression. However, larger block sizes can have a negative impact on performance.

- The build process slows down with larger block sizes.
- Startup time and file reads for applications are slower with large block sizes.
- More memory is required at runtime when larger block sizes are used.

`BlockSize` can be specified in two places:

1 `Package.ini` file: in this case, the block size becomes the default for all files in the project unless otherwise specified.

2 `##Attributes.ini`: in this case, the block size overrides the block size for the present directory and all subdirectories. In the manner you can use different block sizes for different directories within a single project.

For example, the default block size is 64K

```
[Compression]
BlockSize=64k
```

Other block size options follow:

```
'BlockSize=128k
'BlockSize=256k
'BlockSize=512k
'BlockSize=1M
```

CachePath

`CachePath`–Sets the path to the cache directory where font files are stored. This setting can contain macros like `%Local AppData%` which will be expanded before use. If the path is relative, it is interpreted relative to the directory where the package is stored.

The setting can be overridden at runtime by the environment variable, `THINAPP_CACHE_DIR`.

If neither the `CachePath` setting nor the `THINAPP_CACHE_DIR` environment variable are present, a default is used. The default depends on the presence of a `SandboxPath` setting in the `Package.ini` file. If there is a `SandboxPath` setting and it is a relative path, then `CachePath` defaults to the same path. If there is no `SandboxPath` setting or the path setting is absolute, then `CachePath` will default to `%Local AppData%\ThinApp\Cache`.

For example, set the cache directory to `C:\VirtCache`:

```
CachePath=C:\VirtCache
```

With the package located in `C:\VirtApps`, this will set the cache to `C:\VirtApps\Cache`:

```
CachePath=Cache
```

In a typical USB key scenario, you force the sandbox to the USB key. If the packages are stored in a subdirectory, `\VirtApps`, on the USB key, enter the following to force the cache directory to the USB key too:

`CachePath=Sandbox`

CompressionType

`CompressionType`—Controls what type of compression is used.

ThinApp supports two compression models: `None` and `Fast`.

`None`—None is the default when capturing a package. This option is useful for building your application quickly for testing purposes. No compression also improves application startup time on older PCs, or when the application is launched multiple times. On subsequent executions of the same application, the Windows disk cache can provide data faster without compression enabled.

`Fast`—This option is recommend for most packages. Fast compression has a very fast rate of decompression, so it has very little impact on application startup time; it also has very little impact on memory consumption at runtime. Fast compression achieves similar compression ratios as the ZIP algorithm.

The following table displays sample compression ratios and startup times for a Microsoft Office 2003 package running from a local hard drive:

Compression Type	None	Fast
Size	448,616k	257,373k
Compression ratio	100%	57%
Startup time (1st execution)	6 sec	6 sec
Startup time (2nd execution)	0.1 sec	1 sec
Build Time (1st build)	3 mins	19 mins
Build Time (2nd build)	2 mins	1.2 mins

CompressionType can be specified in two places:

- `Package.ini` file: The compression type becomes the default for all files in the project unless otherwise specified.
- `##Attributes.ini` file: The compression type overrides the compression algorithm for the present directory and all subdirectories. In the manner you can use different compression algorithms for different directories within a single project.

For example, use no compression for the fastest build time and fastest load time (default):

```
[Compression]
CompressionType=None
```

Use fast compression for a slow build time, good compression ratio, and fast load time:

```
[Compression]
CompressionType=Fast
```

DisableTracing

`DisableTracing`—Prevents `.trace` file generation when Log Monitor is running.

ThinApp's Log Monitor utility can be used to produce a `.trace` file for applications run by ThinApp. This option can be used to disable the ability to produce trace files for specific applications.

This prevents an application from creating a `.trace` file even if Log Monitor is running:

```
[BuildOptions]
DisableTracing=1
```

This allows Log Monitor to create a `.trace` file (default):

```
[BuildOptions]
DisableTracing=0
```

You might want to turn off `.trace` file generation capabilities for several reasons:

- For security purposes, it might be preferable to hide the execution history.
- During testing, it might be useful to turn off tracing for specific captured applications that are known to work, since producing extra .trace files represents a waste of disk space and CPU time.

ExcludePattern

`ExcludePattern`—Enables you to exclude specified files during the build process.

This option can be used to exclude specific files or directories from a project determined dynamically at build time. The list of patterns is specified with a comma separator where * matches 0 or more following characters and ? matches exactly one character similar to the DOS `dir` command. Unlike DOS `dir` command syntax, wildcard characters can be applied to both directory names and filenames.

`ExcludePattern` can be specified in two places:

- `Package.ini` file: The pattern exclusion applies to the entire directory structure.
- `##Attributes.ini`: The pattern exclusion is added to the current list of exclusions and apply only to this directory and subdirectories. In this manner, you can have a different exclusion list for different directories in your project.

For example:

```
[FileList]
```

Exclude any path that ends with .bak or .msi.

```
ExcludePattern=*.bak,*.msi
```

Exclude any directories called .svn or .cvs (and all subdirectories).

NOTE This pattern does not match filenames or directories that contain `.svn` or `.cvs` in the middle of the string.

```
ExcludePattern=\.svn,\.cvs
```

FileTypes

`FileTypes`–List of file extensions that `ThinReg` should associate with an executable. No separators are needed (or allowed) between the file extensions in the list, so a typical list would be .doc.docx.

This setting only makes sense in an `[app.exe]` section, not in the `[BuildOptions]` section.

Normally, Setup Capture will place the FileTypes list in the Package.ini file that it generates. You can manually remove extensions that you do not want to associate with the virtual package. For example, if you are virtualizing Microsoft Office 2007 and have Office 2003 installed natively, you can remove the .doc extension from the FileTypes list and leave .docx, so .doc files are opened by Word 2003 and .docx by Word 2007.

For example, use ThinReg to create file type associations for .doc and .dot extensions, linking them to Microsoft Word:

```
[Microsoft Office Word 2003.exe]
ReadOnlyData=bin\Package.ro.tvr
Source=%ProgramFilesDir%\Microsoft Office\OFFICE11\WINWORD.EXE
FileTypes=.doc.dot
```

LogPath

`LogPath`—Controls where to store `.trace` files when logging

For example, direct ThinApp to store log files in `c:\ThinappLogs`.

NOTE Unlike most paths in ThinApp, LogPath cannot contain macros such as %AppData% or %Temp%.

```
[BuildOptions]
LogPath=C:\ThinappLogs
```

NetRelaunch

`NetRelaunch`—Controls whether to relaunch an application from the local disk when running from a net share or removable disk.

By default, ThinApp automatically detects if an application is being run from a network drive or removable disk, and relaunches the application using a local fixed disk. This is to resolve a problem in which Symantec AntiVirus tries to perform a complete scan of an executable under some conditions, and this scan can have a big impact on launch times for large executable files located network shares.

Symantec AntiVirus performs a full file-scan if an executable is launched from a network share or removable disk and when an executable makes its first network connection.

Because a large number of desktops have Symantec AntiVirus installed, ThinApp automatically compensates for this by allowing applications to launch from a network share without incurring the lengthy scan times. It does so by creating a small stub executable in the user's sandbox that is then relaunched. Because the small executable can be scanned quickly, it loads the remainder of the application data from the original source location. You can disable this default behavior by adding the `Package.ini` option.

If your application is small in size or you know that Symantec AntiVirus is not installed on the desktops you are deploying to, you might want to turn off NetRelaunch for better first-time launch performance.

For example, disable relaunch of the application locally on a fixed disk.

```
[BuildOptions]
NetRelaunch=0
```

If launched from a network drive, re-launch using a local stub (default).

```
[BuildOptions]
NetRelaunch=1
```

RuntimeEULA

`RuntimeEULA` – Controls the display of the ThinApp EULA display. By default the EULA will not be displayed. Depending on licensing terms for how ThinApp was licensed, you may be required to display an end user license agreement.

Following are examples of this parameter:

```
[BuildOptions]
;Default: do not show an Eula
RuntimeEULA=0
;Turn on display of EULA
RuntimeEULA=1
```

Shortcuts

`Shortcuts`–List of locations where ThinReg should create a shortcut to a virtual application. The list consists of entries separated by semicolons. Each entry can contain a macro value. Use this setting in an `[app.exe]` section only, not in a `[BuildOptions]` section.

For example, use ThinReg to create a shortcut to the virtual Word 2003 application in the "Microsoft Office" folder that is shown in the Start menu:

```
[Microsoft Office Word 2003.exe]
ReadOnlyData=bin\Package.ro.tvr
Source=%ProgramFilesDir%\Microsoft Office\OFFICE11\WINWORD.EXE
Shortcuts=%Programs%\Microsoft Office
```

UpgradePath

`UpgradePath`—Location to probe for application updates.

By default, ThinApp looks for application upgrades in the same directory as the main executable file. This option can be used to specify an alternate location to look for application upgrades.

For example, this instructs ThinApp to probe for application upgrades under `C:\Program Files\MyAppUpgrades.`

```
[BuildOptions]
UpgradePath=%ProgramFilesDir%\MyAppUpgrades
```

When Application Sync downloads an update from a server, it stores the update with a temporary name in the `UpgradePath` directory. After the download is complete, the temporary file will be renamed with a .1 extension (or .2 if .1 already exists) the next time the application is started (no other applications can be using the same sandbox). ThinApp will attempt to change the .1 name (still in the UpgradePath directory) to the original file name (which might be in another directory). If ThinApp is unable to make this change, the file keeps the .1 extension in the `UpgradePath` directory. That file will still be accessed when you execute the original application.

VirtualComputerName

`VirtualComputerName`–A string returned by `GetComputerName` and `GetComputerNameEx` APIs in a ThinApp captured application.

For the computer name, many applications use the name of the machine on which they are installed. Some applications connect to a database and use the name of the computer in the connection string. For captured applications, the computer name is virtualized so it runs well on any machine regardless of the name.

Before you run Setup Capture, rename the clean machine `LOCALHOST`. After you run Setup Capture, the `Package.ini` file contains the following entry:

```
VirtualComputerName=LOCALHOST
```

Because you renamed the machine before you ran Setup Capture, the application uses `LOCALHOST` as its computer name. If you enter a `GetComputerName` or `GetComputerNameEx` command, the machine will return `LOCALHOST`.

If your system requires that the `GetComputerName` and `GetComputerNameEx` APIs behave normally, do not rename the machine `LOCALHOST`. The `VirtualComputerName` parameter is commented out in the `Package.ini` file, as shown below (`%OriginalMachineName%` will be substituted by the machine name).

```
;VirtualComputerName=%OriginalMachineName%
```

VirtualDrives

`VirtualDrives`—Specifies additional drive letters that should be available to the application at runtime.

Virtual drives can help solve issues in which applications rely on hard-coded paths to specific drive letters that might not be available on the client PCs you are distributing to. For example, some legacy applications might be designed to expect that the D drive is a CD-ROM and that various data files are available at `D:\media`.

Virtual drives are visible only to applications running in the virtual environment; they do not have any impact on the real Windows environment. Isolation modes for virtual drives are inherited from the project's default isolation mode unless you specific override it, as shown below. If you configure your virtual drive with the isolation mode setting set to `IsolationMode=Merged`, any writes to that drive fail if it does not exist on the real system.

Projects created by Setup Capture automatically list virtual drive information for drives that were present at the time of capture.

`VirtualDrives` is specified as a single string that can hold information for multiple drive letters and optional parameters for those drive letters. The format for VirtualDrives should look like this:

```
VirtualDrives= Drive=A, Serial=12345678, Type=REMOVABLE; Drive=B,
Serial=9ABCDEF0, Type=FIXED
```

The semicolon is used to separate drive letters, and comma is used to separate parameters for individual drive letters.

`Drive=`(single character between a and z)

`Serial=`(8-digit hex number)

`Type=`

- `FIXED`—The drive has fixed media. For example, a hard drive or internal Flash drive.
- `REMOVABLE`—The drive has removable media. For example, a floppy drive, thumb drive, or flash card reader.
- `CD–ROM`—The drive is a CD-ROM drive.
- `RAMDISK`—The drive is a RAM disk.

The simplest usage is to specify a single virtual drive letter. By default, a drive is assigned a serial number and type of FIXED.

```
[Build]
lDrives=Drive=M
```

This specifies three virtual drive letters (X, D, and Z):

```
[BuildOptions]
VirtualDrives=Drive=X, Serial=ff897828, Type=REMOVABLE; Drive=D, Type=CDROM;
Drive=Z
```

Drive X appears to be a removable disk with Serial number ff797828

Drive D appears to be a CD-ROM drive with an automatically assigned serial number

Drive Z appears to be a FIXED disk with an automatically assigned serial number

---

NOTE The ; character is used to separate information assigned to different drive letters.

To change the isolation mode for a virtual drive

1 Add the folder %Drive_X% to your ThinApp project.

2 In the new directory, place an ##Attributes.ini file to specify the isolation mode for this drive letter.

Access Control

The following sections describe the access control parameters.

AccessDeniedMsg

AccessDeniedMsg—Contains an error message to display to a user if he or she does not have permission to run a package.

The default setting is "You are not currently authorized to run this application. Please contact your Administrator."

For example:

```
[BuildOptions]
PermittedGroups=Administrator;OfficeUsers
AccessDeniedMsg=You do not have permission to execute this application,
please call support @ 1-800-822-2992
```

This setting controls the string that is displayed to the user if they are not authorized to execute the application.

PermittedGroups

`PermittedGroups`—Restricts usage of a package to a specific set of Active Directory groups.

For example, specify a list of Active Directory Group names separated by a semicolon. `[BuildOptions]` can set default for applications in project that can be overwritten by individual `[App.exe]` sections.

```
[BuildOptions]
PermittedGroups=Administrator;OfficeUsers
AccessDeniedMsg=You do not have permission to execute this application,
please call support @ 1–800–822–2992
```

Overwrite global PermittedGroups just this specific application

```
[App1.exe]
PermittedGroups=Guest
AccessDeniedMsg=You do not have permission to execute this application,
please call support @ 1–800–822–2992
```

Inherit PermittedGroups from [BuildOptions]

```
[App2.exe]
...
```

While building a package, ThinApp looks up the specified Group Names into SID values so the Group Names specified must be valid at the time of build. ThinApp can resolve group ownership at runtime using cached credentials so laptop users can continue to be authenticated even when they are offline.

If the user does not have access to execute the package, the AccessDeniedMsg can be customized to instruct the user.

Application-Specific Parameters

The following sections describe the application-specific parameters.

Disabled

`Disabled`—Indicates that a build target is a placeholder and does not generate an executable.

The following example disables generation of the `app.exe` file during build. Changing Disabled to 0 or removing the line enables generation of app.exe.

```
[app.exe]
Source=%ProgramFilesDir%\myapp\app.exe
Disabled=1
```

CommandLine

`CommandLine`—Specifies command-line arguments for a shortcut executable.

For shortcut executables, you can specify the command line that is passed to the virtual application. You should include the application name as the first parameter.

For example, enter `/SomeOption SomeParameter` as your command line arguments.

```
[MyShortcutApp.exe]
Source=%ProgramFilesDir%\Myapp\MyShortcutApp.exe
Shortcut=HostApp.exe
CommandLine="%ProgramFilesDir%\Myapp\MyShortcutApp.exe" /SomeOption
SomeParameter
```

Use folder macros for your path name conventions. See "Folder Macros" on page 96 for more information.

Icon

`Icon`—Indicates the icon file to use for the generated executable file.

By default, each generated application uses the main group icon from its source executable and the individual icon resource pointed to by the group icon. You can use an alternate icon by specifying an `.ico` file or executable file.

For example, specify `NULL` to generate an executable with no icons:

NOTE NULL cannot be used if the file types directive is used, because one icon per file type is allocated in the executable image.

```
[myapp.exe]
Source=%ProgramFilesDir%\myapp\app.exe
Icon=NULL
```

Specify application icon using an executable different from the `Source` executable:

```
[myapp.exe]
Source=%ProgramFilesDir%\myapp\app.exe
Icon=%ProgramFilesDir%\myapp\app2.exe
```

You can optionally specify the set to use by appending `,1 ,2` to the end of the Icon path name like this:

```
[myapp.exe]
Source=%ProgramFilesDir%\myapp\app.exe
Icon=%ProgramFilesDir%\myapp\app2.exe,1
```

Specify the application icon using a `.ico` file:

```
[myapp.exe]
Source=%ProgramFilesDir%\myapp\app.exe
Icon=%ProgramFilesDir%\myap\myicon.ico
```

NoRelocation

`NoRelocation`—Strips relocation information from the resulting executable.

Windows executables can optionally contain base relocation information that enables Windows to load the executable image at an alternate starting memory address. If the base address is above 0x40000 the executable is always loaded at its specified base address, so relocation information is not needed and can be safely stripped out of the resulting image. Relocation information is typically small on disk, so removing this information does not have a big impact on the size of the executable files that are generated.

For example, strip the relocation information from the generated executable file.

```
[app.exe]
Source=%ProgramFilesDir%\myapp\app.exe
NoRelocation=1
```

ReserveExtraAddressSpace

`ReserveExtraAddressSpace`—Indicates how much extra address space needs to be reserved for the captured executable.

Normally, TLink sets the SizeOfImage field in the generated executable based on the SizeOfImage field of the source executable. The SizeOfImage field is used by the Windows loader to determine how much virtual address space to reserve for the executable. In special circumstances (when you are building a package based on a source executable that is not included in the package), you can reserve virtual address space by specifying the `ReserveExtraAddressSpace` option. The value is the number of bytes to reserve. Optionally, you can follow the number by K to indicate kilobytes or M to indicate megabytes. The default value is `0`, which specifies address space to be reserved.

For example, the following tells the Windows loader to reserve 512KB of address space.

```
[app.exe]
Source=%ProgramFilesDir%\myapp\app.exe
ReserveExtraAddressSpace=512K
```

This tells the Windows loader to not reserve any extra address space (default).

```
[app.exe]
Source=%ProgramFilesDir%\myapp\app.exe
ReserveExtraAddressSpace=0
```

RetainAllIcons

`RetainAllIcons`—Indicates that all of the `Source` executable's original icons should be retained in the generated captured executable file.

By default, TLink constructs a new executable using a source executable. To reduce disk space, the new executable image contains only icons viewable from the system shell, while all other icons are stored inside of the package and are still accessible to the application while it is running. The icons that are accessible by the system cannot be compressed, so their disk size is larger. In some cases it might be desirable to have all of the application's original icons visible to the system shell.

Following are examples of this parameter:

Instruct Tlink to retain all of the application's original icons.

```
[app.exe]
Source=%ProgramFilesDir%\myapp\app.exe
RetainAllIcons=1
```

Instruct Tlink to strip out unused icons from the system visible portion of the executable file (default).

```
[app.exe]
Source=%ProgramFilesDir%\myapp\app.exe
RetainAllIcons=0
```

Shortcut

`Shortcut`—Points to the name of the ThinApp-generated executable that hosts the package data..

For example, this creates a "shortcut" application that references `HostApp.exe.`

`HostApp.exe` should be listed somewhere in the package and also needs to be present in the same directory in order for `MyShortcutApp.exe` to run.

```
[MyShortcutApp.exe]
Source=%ProgramFilesDir%\Myapp\MyShortcutApp.exe
Shortcut=HostApp.exe
```

Source

`Source`—Points to the executable file that is initially loaded by ThinApp.

`Source` is specified on a per-executable basis. If an application suite has 3 accessible user entry points; for example, `Winword.exe`, `Powerpnt.exe`, and `Excel.exe`, your `Package.ini` lists three application entries, and each entry has a different source entry.

The following are examples of this parameter:

Creates a user-accessible entry point for `C:\Program Files\Myapp\app1.exe.`

```
[app1.exe]
Source=%ProgramFilesDir%\Myapp\app1.exe
```

Creates an user-accessible entry point for `C:\Program Files\Myapp\app2.exe.`

```
[app2.exe]
Source=%ProgramFilesDir%\Myapp\app2.exe
```

StripVersionInfo

`StripVersionInfo`—Removes all version information from the source executable when building the target application.

Version information in an executable can be found in the Properties dialog box from the Windows shell. Typically, this includes copyright, trademark, and version number information. By default, ThinApp copies all version information from the source executable (specified using Source) (See "Source" on page 50). This option can be used to strip version information from the resulting application.

For example, this generates a target application with no version information:

```
[app.exe]
Source=%ProgramFilesDir%\myapp\app.exe
StripVersionInfo=1
```

WorkingDirectory

`WorkingDirectory`—Sets the current working directory before the application starts.

This option can set the CWD (Current Working Directory) for individual applications. The CWD setting is applied before the application starts. The CWD value does not need to exist on the system.

For example, this sets the current working directory to `C:\Program Files\My Application.`

```
[Application.exe]
WorkingDirectory=%ProgramFilesDir%\My Application
```

Version.XXXX

`Version.XXXX`—Used to override executable version strings or add new version strings.

Version resources are normally copied from the original executable. You can override the version resource strings (and add new ones) using a "Version.<string_name>=string_value" setting.

For example, this sets the `VersionInfo` field `ProductName` to the value `My New Product Name`.

```
[Application.exe]
Version.ProductName=My New Product Name
Version.Description=This Product is great!
```

Application Link

In the `Package.ini` file, you can use the features `OptionalAppLinks and RequiredAppLinks` to dynamically combine ThinApp packages together at runtime on end-user PCs. See "OptionalAppLinks" on page 57 and "RequiredAppLinks" on page 56. This enables you to package, deploy, and update component pieces separately while retaining the benefits of application virtualization.

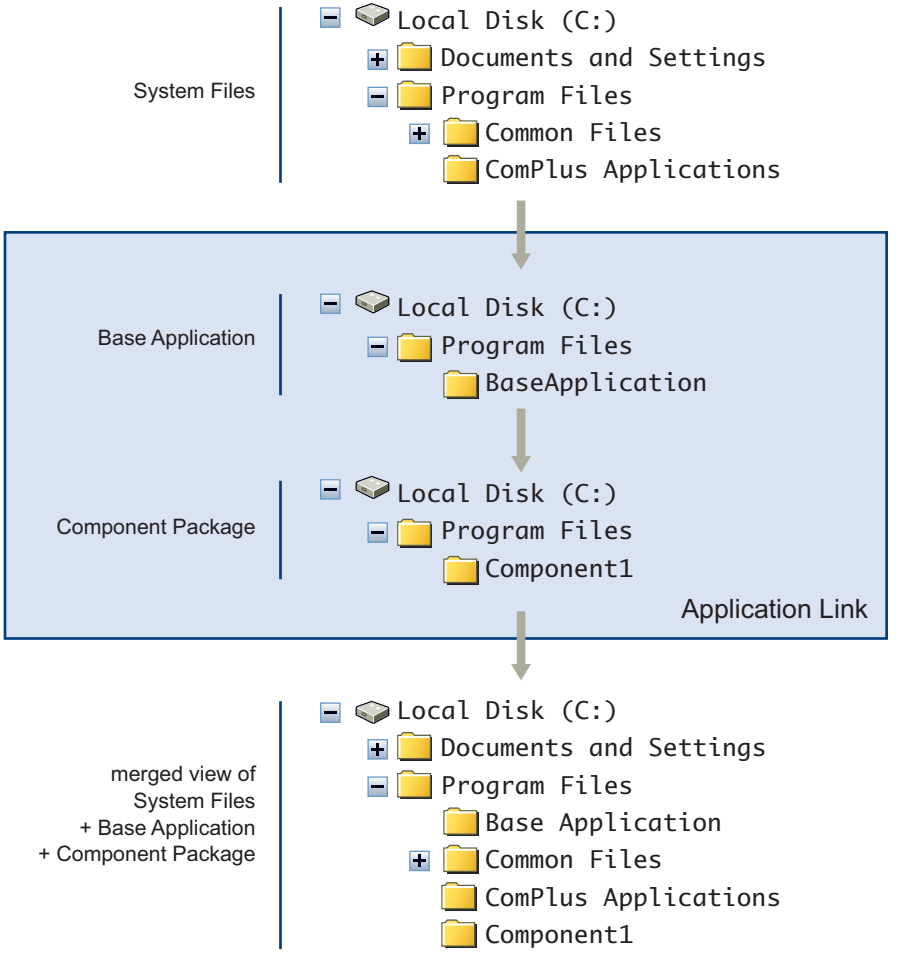
Following are several examples for how you might use Application Link:

- Linking large shared libraries/frameworks – You can link runtime components such as .NET, JRE, or ODBC drivers with applications that have dependencies on these applications.
- Linking add-ons and plug-ins – You might want to package and deploy application-specific add-ons and plug-ins separately from the base application. Application Link allows you to easily deploy a single virtualized Microsoft Office to all users and deploy individual add-ons on a per-user basis.
- Linking upgrades and service packs – As part of the process for rolling out updates and service packs, it is important that you can quickly roll back to a previous version if users experience significant issues with the new version of the application. Application Link allows you to deploy minor patches and upgrades to applications as a single file, so rollbacks can be reduced to removing this file.

NOTE ThinApp does not support recursive use of Application Link. For example, if ThinApp package A.exe depends on ThinApp package B.exe and package B.exe depends on ThinApp package C.exe, when running ThinApp package A.exe, the contents of ThinApp package B.exe will be imported into the virtual environment but contents of ThinApp package C.exe will not.

At runtime, Application Link presents the running application with a merged view of the system, the base application, and all linked components. Files, registry keys, services, COM objects, environment variables from dependency packages will be visible to the base application. See Figure 3-1.

Figure 3-1. Application Link Presents Merged View of System, the Base Application, and All Linked Components



Example Workflow for Application Link

This example uses Application Link to link a base application called `MyApp.exe` with a separate package containing the .NET 2.0 Framework. The first step is to capture the main base application and allow its dependencies to be in separate packages. For this example, we will capture two projects:

1. `C:\Captures\dotnet` (dependency for the main base application)
2. `C:\Captures\MyApp` (the main base application)

The order of capture is important because the main base application must be captured without the .NET framework being included in the project.

The following procedure requires a clean PC or virtual machine. See "Using a Clean PC for Capturing" on page 17. This procedure assumes that you are familiar with the process of capturing an application in ThinApp. See Chapter 2, "Capturing and Configuring Applications," on page 15.

To link a base application called with a separate package containing the .NET 2.0 Framework

1 Capture the installation of .NET framework only.

   a Start the capture process on a clean PC or virtual machine that does not have .NET or the main base application.

   b Before saving your .NET framework capture, follow the prompt to select at least one user accessible entry point.

   c Rename the final `exe` produced by ThinApp from a `.exe` to a `.dat` filename extension so users are not confused and accidently run the application.

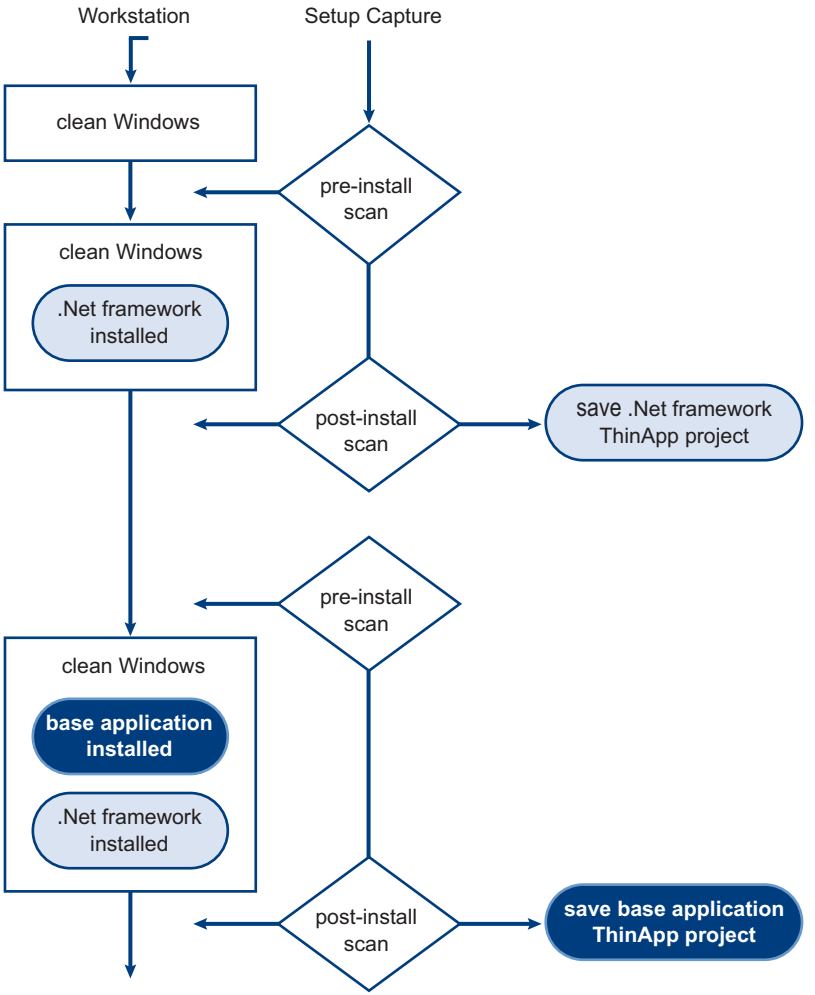
   The name of the `exe` you select does not matter and `cmd.exe` is a suitable choice. The .NET `exe` produced will be used only for linking purposes. Users will not run the .NET `exe` directly.

   d Save the .NET project to `C:\Captures\dotnet`

2 Capture the base application.

a Using the same PC or virtual machine resulting from the end of step 1, capture the installation of the application with the .NET framework already installed on the PC, as shown in the following illustration.

b Save the project to `C:\Captures\MyApp`.



3 Enable the optional AppLinks parameter for the base application package.

   a Open the text file `c:\Captures\MyApp\Package.ini` in a text editor.

   b Add the following line after the line `[BuildOptions]`.
   `RequiredAppLinks=dotnet.dat`

4 Build `.net` and base application packages

   a Run the batch file `c:\Captures\MyApp\build.bat`

   a Run the batch file `c:\Captures\dotnet\build.bat`

   By running these batch files, you build two separate ThinApp packages.

5 Deploy your application to an end-user desktop, at the location `c:\Program Files\MyApp`.

   a Copy `c:\Captures\MyApp\bin\MyApp.exe` to
   `\\enduserdesktop\ProgramFilesShare\MyApp\MyApp.exe`

   b `Copy c:\Captures\dotnet\bin\cmd.exe` to
   `\\enduserdesktop\ProgramFilesShare\MyApp\dotnet.dat`

NOTE If you plan to use `.msi` distribution to deploy packages linked using AppLink, change the RequiredAppLinks to the value "`..\dotnet\dotnet.dat`" for the base package and make sure the `.net` ThinApp package installs to `c:\program files\dotnet`.

Application Link Pathname Formats

Application Link supports the following pathname formats:

- Pathnames can be relative to the base EXE. For example, "`RequiredAppLinks=..\SomeDirectory`" will result in "`c:\MyDir\SomeDirectory`" if the base EXE is deployed to "`c:\MyDir\SubDir\Dependency.exe`".
- Pathnames can be an absolute pathname. For example "`RequiredAppLinks=c:\SomeDirectory`"
- Pathnames can be located on a network share or use a UNC path. For example "`RequiredAppLinks=\\share\somedir\Dependency.exe`"
- Pathnames can contain environment variable and dynamically expand to any of the above. For example "`RequiredAppLinks=%MYAPP_ADDONS%\Dependency.exe`"

- Multiple links or dependencies can be specified by using the ';' character to separate individual filenames. For example "`RequiredAppLinks=Dependency1.exe; Dependency2.exe;`"

Collisions Within Linked Packages

If the base application contains a file or registry entry at the same location as a dependency package, a collision occurs. When this happens, the order of import determines which package wins. The last package imported will win in such cases and the file or registry contents from that package will be visible to the running application. Application Links are imported in the order they are listed in the `RequiredAppLinks` or `OptionalAppLinks` parameter. If either parameter specifies a wildcard character that matches more than one file, alphabetical order will be used to determine which package is imported first.

If two or more packages include VB scripts, the order of execution for the VB Scripts will be alphabetical order by the name of the package. If two packages contain a VB script with the same name, the "last import wins" policy will be used to execute only the version of the VB script from the last imported package containing a script with that name. Because VB Script name collisions could cause scripts from other packages not to be executed, it is important to use unique name for VB Script filenames.

Following is an example of package import order:

```
OptionalAppLinks=a.exe;b.exe;plugins\*.exe
```

For this example, the import order is:

1 The base application

2 `a.exe`

3 `b.exe`

4 `plugins\a.exe`

5 `plugins\b.exe`

RequiredAppLinks

`RequiredAppLinks`—Specifies a list of external ThinApp packages to import into the current package at runtime.

If any specified package fails to import, an error message will be displayed and the parent executable file will exit. To continue even if load errors occur, use `OptionalAppLinks` instead. If a wild-card pattern is used to specify a package, no error message is displayed if no files match the wildcard pattern.

Importing packages will:

- Run vb scripts from imported packages
- Start auto-start services from imported packages
- Register fonts from imported packages
- Relocate SxS DLLs from XP->Vista (if required)

You cannot import a shortcut package. You can only import the Primary Data Container.

OptionalAppLinks

`OptionalAppLinks` – Operates exactly like `RequireAppLinks` except that if an import fails to load, the error is ignored and the main application will start executing. See "RequiredAppLinks" on page 56.

Collisions and Order of Import

ThinApp uses a "last import wins" policy to determine what happens when two packages are imported that have the same files or registry keys. See "Collisions Within Linked Packages" on page 56.

Security and Authorization

You must be a member of all PermittedGroups sections for all imported packages. Otherwise you will be receive an Access Denied message and the application will fail to load.

Following are limitations of this feature:

- ThinApp supports importing up to 250 packages at a time, and each package may be any arbitrary size.
- Packages that have been updated via AppSync will not have updates visible to the parent executable.
- Sandbox changes from packages being imported will not be visible to the parent executable.

The following are examples:

This will import a single package located in the same directory as the parent executable:

```
RequiredAppLinks=Plugin.exe
```

This will import a single package located in a subdirectory of the parent executable:

```
RequiredAppLinks=plugins\Plugin.exe
```

This will import all executables located in the plugins directory:

NOTE If any executable fails to import because it is not a proper Thinapp package or because of a security issue, the parent executable will fail to load.

```
RequiredAppLinks=plugins\*.exe
```

This will import all EXEs located at the absolute path `n:\plugins`:

```
RequiredAppLinks=n:\plugins\*.exe
```

This expands the environment variable, `PLUGINS`, and imports all executables found at this location:

```
RequiredAppLinks=%PLUGINS%\*.exe
```

This loads two specified plug-ins and a list of executables found under plugins:

```
RequiredAppLinks=plugin1.exe;plugin2.exe;plugins\*.exe
```

Application Sync

Application Sync enables you to automatically keep deployed virtual applications up to date. When an application starts up, Application Sync can query a Web server to see if an updated version of the package is available. If an update is available, the differences between the existing package and the new package will be downloaded and used to construct an updated version of the package. The updated package will be used for future deployments.

You can use Application Sync to do the following:

- Distribute runtime components separately from the applications that use them. For example, the Java Runtime Environment (JRE) or ODBC drivers.
- Link add-ons and plug-ins to applications. For example, Microsoft Office add-ons or Adobe Photoshop plug-ins can be linked using Application Sync.
- Apply layered service packs to your applications. Application Sync enables you to distribute service packs and roll back to previous versions, if necessary.

You can configure Application Sync with Setup Capture or by editing the package.ini file. For more information on using Setup Capture, see "Using Setup Capture" on page 18.

To make changes manually, open your `Package.ini` file. The default location of your `Package.ini` file is the `Program Files\VMware\ThinApp\Captures\` directory.

Following are the default settings for Application Sync:

```
AppSyncURL=https://example.com/some/path/PackageName.exe
AppSyncUpdateFrequency=1d
AppSyncExpirePeriod=30d
AppSyncWarningPeriod=5d
AppSyncWarningFrequency=1d
AppSyncWarningMessage=This application will become unavailable for use in
AppSyncWarningPeriod days if it cannot contact its update server. Check your
network connection to ensure uninterrupted service
AppSyncExpireMessage=This application has been unable to contact its update
server for AppSyncExpirePeriod days, so it is unavailable for use. Check your
network connection and try again
AppSyncUpdatedMessage=
AppSyncClearSandboxOnUpdate=0
```

The following sections describe the Application Sync parameters.

NOTE If you use Application Sync, VMware recommends that you disable automatic application updates that are configured in your virtual application. Conflicts might occur between the linked packages and the software that is automatically updated.

If an automatic update feature updates an application, it stores the updates in the sandbox. If Application Sync then updates the application to a different version, the updates stored in the sandbox take precedence over the files contained in the version that Application Sync created. The order of precedence for the update files are those in the sandbox, then the virtual operating system, and then the physical machine.

AppSyncURL

`AppSyncURL`–URL of the Web server where updates are stored. Application Sync works over both the HTTP (unsecure) and HTTPS (secure) protocol. Part of HTTPS is that the identity of the Web server is checked. You can include a username and password in the AppSyncURL that will be used for basic authentication. The standard Windows/Internet Explorer proxy setting is respected.

For example:

```
AppSyncURL=https://example.com/some/path/PackageName.exe
```

AppSyncUpdateFrequency

`AppSyncUpdateFrequency`–By default, a package will connect to the Web server once per day to see if an updated version is available. You can set the frequency by modifying this setting.

For example:

```
AppSyncUpdateFrequency=1d
```

The Application Sync update frequency can be set to...

AppSyncExpirePeriod

`AppSyncExpirePeriod`–Sets the update frequency in minutes (m), hours (h), or days (d). If the Web server cannot be reached, the package will continue to work until the AppSyncExpirePeriod is reached. This default setting is 30 days but you can change that setting by modifying this setting.

For example:

```
AppSyncExpirePeriod=30d
```

If you do not want the package to expire, set this as follows:

```
AppSyncExpirePeriod=never
```

AppSyncWarningPeriod

`AppSyncWarningPeriod`–Sets the start of the warning period before a package expires.

For example:

```
AppSyncWarningPeriod=5d
```

AppSyncWarningFrequency

`AppSyncWarningFrequency`–Sets the frequency of warnings before the package expires.

With the default of one day the warning message will be displayed once per day only. To configure the warning to pop up on every application launch, use 0.

After the warning period has started, the Web server will be checked on every launch of an application, overriding the setting.

As long as a package has not expired, this parameter checks for new versions and downloads will occur in the background. The user can continue to use the old version. If the application is terminated by the user before the download is complete, the download will resume when a virtual application is launched again. After the download completes, the new version will be activated on the next launch.

When the package has expired, the version check and download will happen in the foreground. A progress bar will be shown during the download phase.

For example:

```
AppSyncWarningFrequency=1d
```

AppSyncWarningMessage

`AppSyncWarningMessage`–If the connection to the Web server fails, a message box will be shown. The default message is shown in the example below.

For example:

```
AppSyncWarningMessage=This application will become unavailable for use in
AppSyncWarningPeriod days if it cannot contact its update server. Check your
network connection to ensure uninterrupted service
```

AppSyncExpireMessage

`AppSyncExpireMessage`–After the expiration limit has been reached and a virtual application is started, it will try to connect to the Web server and check for a new version. If the connection fails, a message box will be shown and execution will be terminated. The default message is shown in the example below.

For example:

```
AppSyncExpireMessage=This application has been unable to contact its update
server for AppSyncExpirePeriod days, so it is unavailable for use. Check your
network connection and try again
```

AppSyncUpdatedMessage

`AppSyncUpdatedMessage`–When an updated package is first launched, an information message can be shown.

For example:

```
AppSyncUpdatedMessage=Your application has been updated.
```

AppSyncClearSandboxOnUpdate

`AppSyncClearSandboxOnUpdate`–Gives you the option to clear the sandbox after an update. By default, the sandbox is not cleared. Set this to 1 to clear the sandbox or 0 to leave the sandbox uncleared.

For example:

```
AppSyncClearSandboxOnUpdate=0
```

MSI Generation

The following parameters are used for MSI file generation.

MSIDefaultInstallAllUsers

`MSIDefaultInstallAllUsers` – Sets the default installation mode of the MSI database. The default setting is 1.

This option has an effect only when generation of a Windows Installer database is requested using the `MSIFilename` option ("MSIFilename" on page 63).

When set to 1, the default installation mode of the generated MSI database is set to `per-machine`. If you want the default mode to be `per-user`, set this option to 0. Setting it to 2 results in a default mode of per-machine for administrators and a per-user for non-administrators.

For example, this directs ThinApp to generate an MSI that is installed on a per-user basis by default.

NOTE By suppling command-line arguments to `msiexec`, you can force a per-machine installation

For example `msiexec.exe mymsi.msi ALLUSERS=1`

```
[BuildOptions]
MSIFilename=mymsi.msi
MSIDefaultInstallAllUsers=0
```

For example, this directs ThinApp to generate an MSI that is installed on a per-machine basis by default.

NOTE By supplying command-line arguments to `msiexec`, you can force a per user-machine installation.

For example `msiexec.exe mymsi.msi ALLUSERS=1`

```
[BuildOptions]
MSIFilename=mymsi.msi
MSIDefaultInstallAllUsers=1
```

For example, this directs ThinApp to generate an MSI that is installed on a per-machine basis for administrators and per-user for non-administrators if the user does not have permission to install on a per-machine basis.

NOTE By supplying command-line arguments to msiexec you can force a per user-machine installation

For example msiexec.exe mymsi.msi ALLUSERS=1

```
[BuildOptions]
MSIFilename=mymsi.msi
MSIDefaultInstallAllUsers=2
```

MSIFilename

`MSIFilename` – Enables the generation of an MSI database and specifies its filename.

If set, this option causes the build to produce a Windows Installer with the specified filename in the output directory. For more information, see Chapter 10, "Generating MSI Files," on page 145.

For example, this directs ThinApp to generate an MSI during build and replace mymsi.msi with your own filename.

```
[BuildOptions]
MSIFilename=mymsi.msi
```

MSIInstallDirectory

`MSIInstallDirectory` – Specifies the path of the default installation directory.

This option has an effect only when generation of a Windows installer database is requested using the `MSIFilename` option.

By default, packages are installed in the `%ProgramFilesDir%\<InventoryName>` directory (for a per-machine install). You can change the default installation path by using the `MSIInstallDir` option. When using a relative path, the path is relative to `%ProgramFilesDir%` for per-machine installations (`MSIDefaultInstallAllUsers=1`) and relative to `%AppData%` for per-user installations (`MSIDefaultInstallAllUsers=0`).

For example, if you set `MSIInstallDirectory=ExampleDir` the default installation directory (for per-machine installations) is `%ProgramFilesDir%\ExampleDir`.

The following example shows how to create an `.msi` file that is installed to `C:\Program Files\My Application\`.

```
[BuildOptions]
MSIFilename=mymsi.msi
MSIInstallDirectory=My Application
```

MSIManufacturer

`MSIManufacturer` – Specifies the manufacturer to put in the MSI database. The default setting is `VMware, Inc.`

This option has an effect only when generation of a Windows Installer database is requested using the `MSIFilename` option.

Set this option to the name of your organization. It is displayed when you show the properties of the database, but has no effect otherwise.

For example, this creates an MSI file with the manufacturer set to `My Company Name`.

```
[BuildOptions]
MSIFilename=mymsi.msi
MSIManufacturer=My Company Name
```

MSIProductCode

`MSIProductCode` – Specifies a product code for the MSI database.

This option has an effect only when generation of a Windows Installer database is requested using the `MSIFilename` option.

Each MSI database needs a product code. Setup Capture generates a suitable default product code and place it in the `Package.ini` file. If you change the product code, make sure the new value is a valid GUID (Globally Unique IDentifier).

The following example creates an `MSI` file with a specific product code.

```
[BuildOptions]
MSIFilename=mymsi.msi
MSIProductCode={590810CE-65E6-3E0B-08EF-9CCF8AE20D0E}
```

MSIProductVersion

`MSIProductVersion` – Specify a product version number for the MSI database. The default setting is `1.0`.

This option has an effect only when generation of a Windows Installer database is requested using the `MSIFilename` option.

The product version number is displayed when you show the properties of the database. When you deploy a package to a machine that already has the package installed, Windows Installer checks the version numbers and refuses to install an older version over a newer version. You must manually uninstall the old version first.

The following example creates an MSI file with a specific product version:

```
[BuildOptions]
MSIFilename=mymsi.msi
MSIProductVersion=1.0
```

MSIRequireElevatedPrivileges

`MSIRequireElevatedPrivileges` – Marks the MSI database as "requires elevated privileges." The default setting is `1`.

This option has an effect only when generation of a Windows Installer database is requested using the `MSIFilename` option. This option affects Microsoft Vista only.

When this option is set to `1`, the generated MSI database is marked as requiring elevated privileges. If your system is set up for UAC prompts, this results in a UAC prompt when installing.

When you set the option to 0, no UAC prompt is given, but you will not be able to install machine-wide.

This creates an MSI file that always prompts for elevated privileges on Vista.

```
[BuildOptions]
MSIFilename=mymsi.msi
MSIRequireElevatedPrivileges=1
```

MSIUpgradeCode

`MSIUpgradeCode` – Specifies an upgrade code for the MSI database.

This option has an effect only when generation of a Windows Installer database is requested using the `MSIFilename` option.

VMware recommends that each MSI database has an `UpgradeCode`. Setup Capture generates a suitable default `UpgradeCode` and places it in the `Package.ini` file. Do not change the `UpgradeCode` value. If you change it, make sure the new value is a valid GUID (globally unique identifier).

This creates an MSI file with a specified upgrade code.

```
[BuildOptions]
MSIFilename=mymsi.msi
MSIUpgradeCode={D89F1994-A24B-3E11-0C94-7FD1E13AB93F}
```

Sandbox Control

The following sections describe the sandbox control parameters.

SandboxName

`SandboxName`—Sets the name of directory where the sandbox is created and stored

For example:

```
[BuildOptions]
SandboxName=My Application 1.0
```

The `SandboxName` is used when creating a new sandbox. For the example above, the default sandbox path is:

```
C:\Documents and Settings\USERNAME\Application Data\Thinapp\My Application
1.0
```

When upgrading an application, the sandbox name plays an important role in determining if the user retains their previous personal settings or use new settings. By changing the `SandboxName` with new deployments, you can determine whether the user creates a new sandbox with different settings or retains the same sandbox.

SandboxPath

`SandboxPath`—Controls the path where ThinApp creates a new sandbox by default

Following are examples of this parameter:

The following example shows how to create the default sandbox in the same directory as the executable:

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

The following example shows how to create the default sandbox in a subdirectory subordinate to the executable location:

```
[BuildOptions]
SandboxPath=LocalSandbox\Subdir1
```

The following example shows how to create the default sandbox in the user's `AppData` folder under the subdirectory `Thinapp`:

```
[BuildOptions]
SandboxPath=%AppData%\Thinapp
```

The following example shows how to create the default sandbox on a network mapped drive:

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

If an application is meant to run only from portable media such as USB Flash devices, use `SandboxPath` to force the application to use a local sandbox. You can also control the default location of the sandbox using environment variables or by creating a `Thinapp` directory. Environment variables and a local `Thinapp` directory takes precedence over the path specified by `SandboxPath`.

InventoryName

`InventoryName`—Optional string that is used for package identification by inventory control systems.

For example:

`InventoryName` is typically a version independent string that tracks a specific resource of interest to inventory scanning applications.

```
[BuildOptions]
InventoryName=Microsoft Office 2003
```

`InventoryName` is meant to be a version-independent string that can be used to track a ThinApp package with inventory-scanning software.

When deploying new versions of an application, you might want to change the `SandboxName` so that the new version has isolated user settings. However, `InventoryName` can be left constant across versions of the same application.

`InventoryName` is not used by ThinApp during build or execution.

SandboxNetworkDrives

`SandboxNetworkDrives`—Determines if network-mapped drives have sandboxing applied.

By default, ThinApp enables you to write directly to network-mapped drives without applying sandboxing.

For example, the following prevents the user from writing directly to network-mapped drives and changes the sandbox instead.

```
[BuildOptions]
SandboxNetworkDrives=1
```

This example enables you to write directly to network-mapped drives without changes going to the sandbox.

```
(default)
[BuildOptions]
S9andboxNetworkDrives=0
```

SandboxRemovableDisk

`SandboxRemovable`—Determines if removable drives have sandboxing applied.

By default, ThinApp enables you to write directly to removable drives without applying sandboxing. Removable disks include USB Flash and removable hard drives.

For example, this prevents you from writing directly to removable disks and instead, changes go to the sandbox.

```
[BuildOptions]
SandboxRemovableDisk=1
```

This example enables you to write directly to removable disks without changes going to the sandbox (default).

```
[BuildOptions]
SandboxRemovableDisk=0
```

RemoveSandboxOnExit

`RemoveSandboxOnExit`—Resets the application by deleting the sandbox when the last child process exits.

All application modifications to registry and file system locations that have isolation modes set to WriteCopy or Full go to a sandbox directory. By default, the sandbox directory is left behind so that settings are consistent across multiple executions of the application. In some cases, you might want to delete the sandbox each time the application exits.

If the application creates child processes, the sandbox is not deleted until all child processes exit. In some cases, applications might leave children behind by design which can stop the clean-up operation from occurring. For example, Office 2003 leaves behind a process called ctfmon.exe. It might be necessary to use a script to kill ctfmon.exe and similar children to force this cleanup operation to occur.

NOTE You can also decide dynamically at runtime whether to delete the sandbox on exit by using the Script API function, `RemoveSandboxOnExit`.

The following example shows how to direct ThinApp to delete the sandbox when the application exits:

```
[BuildOptions]
RemoveSandboxOnExit=1
```

The following example shows how to direct ThinApp to leave the sandbox behind when the application exits (default):

```
[BuildOptions]
RemoveSandboxOnExit=0
```

Configuring the Sandbox 4

This chapter describes the sandbox and contains the following topics:

Introduction to the Sandbox

The sandbox holds runtime modifications that applications make as they are running. The original executable that you built never changes, so it can be placed in a shared folder with read-only access.

The name of a sandbox (`SANDBOXNAME`) is specified in the `Package.ini` file prior to building your package.

Under normal circumstances, the sandbox is stored under `%AppData%\Thinapp\SANDBOXNAME`. On Windows XP this path is `C:\Documents and Settings\USERNAME\Application Data\SANDBOXNAME`.

During startup, ThinApp searches for an existing sandbox in the following locations in the order listed below. If a sandbox already exists, ThinApp uses the existing sandbox rather than creating a new one.

To search using an application-specific environment variable

1 `%SANDBOXNAME_SANDBOX_DIR%.ComputerName`

   For example, if SANDBOXNAME="Mozilla Firefox 1.0", the environment variable Mozilla Firefox 1.0_SANDBOX_DIR.ComputerName is consulted.

2 %SANDBOXNAME_SANDBOX_DIR%

To search using a global environment variable that applies to all captured applications

1 %THINAPP_SANDBOX_DIR%\SANDBOXNAME.ComputerName

   Example: THINAPP_SANDBOX_DIR.ComputerName

2 %THINAPP_SANDBOX_DIR%\SANDBOXNAME

To search for an application-specific local sandbox

1 ExeDirectory\SANDBOXNAME.ComputerName

2 ExeDirectory\SANDBOXNAME

To search for a local sandbox for all captured applications

1 ExeDirectory\Thinapp\SANDBOXNAME.ComputerName

2 ExeDirectory\Thinapp\SANDBOXNAME

To use the "SandboxPath=" Package.ini value to set a custom sandbox path

1 SANDBOXPATH\SANDBOXNAME.ComputerName

2 SANDBOXPATH\SANDBOXNAME

To search in the user's "Application Data\Thinapp" folder for a sandbox

1 %AppData%\Thinapp\SANDBOXNAME.ComputerName

2 %AppData%\Thinapp\SANDBOXNAME

If an existing sandbox cannot be found, ThinApp creates a new sandbox. ThinApp creates a new sandbox using the following logic:

1 If the SANDBOXNAME_SANDBOX_DIR environment variable is set, try to create a sandbox at this location.

2 If the THINAPP_SANDBOX_DIR environment variable is set, try to create a sandbox at this location.

3 If the SANDBOXPATH Package.ini option is set, try to create a sandbox at this location.

4 Try to create a sandbox in the user %AppData%\Thinapp folder.

NOTE SANDBOXNAME and SANDBOXPATH come from the Package.ini file.

The following examples show you how to use the procedures shown above:

- You want to run from USB Flash and have your sandbox load and save to the same directory where the executable is located.

  In your `Package.ini` file in the `[BuildOptions]` section, set `SandboxPath=.`

- Your users have a mapped drive to store their files and you want the sandbox stored on the mapped drive.

  Use `SandboxPath=z:\Sandbox` if the drive is always mapped to the same letter or in the user login script, set the environment variable `THINAPP_SANDBOX_DIR=z:\Sandbox`. Any captured application run after setting this variable uses `z:\Sandbox` for its sandbox storage location.

Sandbox Structure

The sandbox is stored using a file structure nearly identical to the build project structure. Macro names are used for shell folder locations instead of hard-coded paths. This enables the sandbox to migrate to different PCs dynamically when the application is run from new locations.

In addition to shell folders like `%AppData%`, ThinApp also creates three registry files:

- `Registry.rw.tvr` (contains all registry modifications made by the application)
- `Registry.rw.lck` (zero byte lock file used to prevent other computers from simultaneously using a registry located on a network share)
- `Registry.tvr.backup` (contains a backup of the `.tvr` file from the last good execution if corruption is detected. On startup the .tvr file is restored.)

Additionally, if an application executes a child process, ThinApp creates a directory name that looks like the following:

```
40000026500002i\
```

This directory stores a dynamically generated executable stub for the child process so the correct name is displayed in Explorer for that child process. You can delete these directories and they are regenerated the next time they are required.

ThinApp stores all of its file system information in the virtual registry. This enables it to optimize file system access in the virtual environment. For example, when an application tries to open a file, ThinApp does not need to consult the real file system twice to determine if the file exists or not (once for the real system location, and once for the sandbox location). Instead, ThinApp is able to check for the file's existence by only consulting the virtual registry. Because of this, ThinApp has great runtime performance.

One artifact of this optimization is that if you copy files into the sandbox directory structure directly, the files are not visible to the application. If the file already exists in the sandbox you can overwrite and update the file. VMware recommends leaving the sandbox alone and performing all modifications in the virtual environment instead.

The command-line utility `vregtool` can be used to list the contents of a virtual registry file as shown below. Some of the virtual file system data is stored in the registry tree, `HKLM\FS`.

```
vregtool registry.rw.tvr printkeys

sb_only HKEY_LOCAL_MACHINE\FS\%Profile%\.gimp-2.2\unitrc
deleted HKEY_LOCAL_MACHINE\FS\%Profile%\.fonts.cache-1.NEW
writecopy HKEY_LOCAL_MACHINE\Software writecopy
HKEY_LOCAL_MACHINE\Software\Microsoft
writecopy HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
writecopy HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion
writecopy
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer
writecopy
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell
Folders
writecopy
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\User
Shell Folders
writecopy HKEY_LOCAL_MACHINE\Software\Thinapp
full  HKEY_LOCAL_MACHINE\Software\Thinapp\ProcessList
full  HKEY_LOCAL_MACHINE\Software\Thinapp\RuntimeObjects
full  HKEY_LOCAL_MACHINE\Software\Thinapp\SxS
writecopy HKEY_CURRENT_USER\Software writecopy
HKEY_CURRENT_USER\Software\Microsoft
writecopy HKEY_CURRENT_USER\Software\Microsoft\Windows
writecopy HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion
writecopy
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer
writecopy
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoi
nts2
writecopy
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoi
nts2\{c1664 b80-e468-11db-8906-806d6172696f}
writecopy
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoi
nts2\{c1664 b81-e468-11db-8906-806d6172696f}
writecopy
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoi
nts2\{c1664 b82-e468-11db-8906-806d6172696f}
```

```
writecopy
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoi
nts2\{c1664 b83-e468-11db-8906-806d6172696f}
writecopy HKEY_CURRENT_USER\Software\NVIDIA Corporation
writecopy HKEY_CURRENT_USER\Software\NVIDIA Corporation\Global
writecopy HKEY_CURRENT_USER\Software\NVIDIA Corporation\Global\nView
```

Controlling the Location of the Sandbox

The sandbox is the directory where all changes made by the application are stored. The next time the user executes the application, those changes are remembered from the sandbox. By deleting the sandbox directory, you can instantly revert the application back to its captured state.

By default the sandbox is located in the users `%AppData%` directory:

```
c:\documents and settings\username\Application Data\Thinapp
```

Using Active Directory, the user's `%AppData%` is often mapped to a shared network drive to allow for easy backups. When this is the case, users can log into any machine and retain their application settings.

ThinApp transparently remaps registry and file data while the application is running to enable shared application profile information to instantly migrate to different operating systems. For example, if the application registers DLLs to `c:\winnt\system32` while running on Windows 2000, the user can quit the application and login on a Windows XP machine. On the Windows XP machine, the files automatically appear to exist at `c:\windows\system32` and all related registry keys automatically point to `c:\windows\system32`.

On Windows Vista, ThinApp automatically moves Windows SxS DLLs and policy information to match Vista versus XP file path styles. This functionally allows most applications to instantly migrate to newer or older operating systems.

ThinApp provides SxS support for applications running on Windows 2000 even though the underlying operating system does not. This enables most applications captured on Windows XP to run on Windows 2000 without changes.

NOTE Only one computer can be actively using a shared sandbox. If a sandbox is already in use by another computer, ThinApp displays a warning and creates a new sandbox to allow you to continue working until the previous copy closes.

Portable Applications

To make an application portable so that it can be executed from a USB Flash device, IPod, or other similar device, the sandbox must operate in local mode. In this mode, the sandbox is in a subdirectory relative to the executable. This mode is not recommended for shared environments like Terminal Server unless each user has his own copy of the application. Sandboxes are location-specific, not user-specific.

To set an application to run with a local sandbox, create a directory called `Thinapp` in the same directory as your captured application. You can also move the `Thinapp` directory from `%AppData%` to the application directory and make the existing application settings operate in local mode.

Deploying and Working with Virtual Applications

5

This chapter describes how to work with virtual applications that you have captured with ThinApp. This chapter contains the following topics:

Stream ThinApp Packages

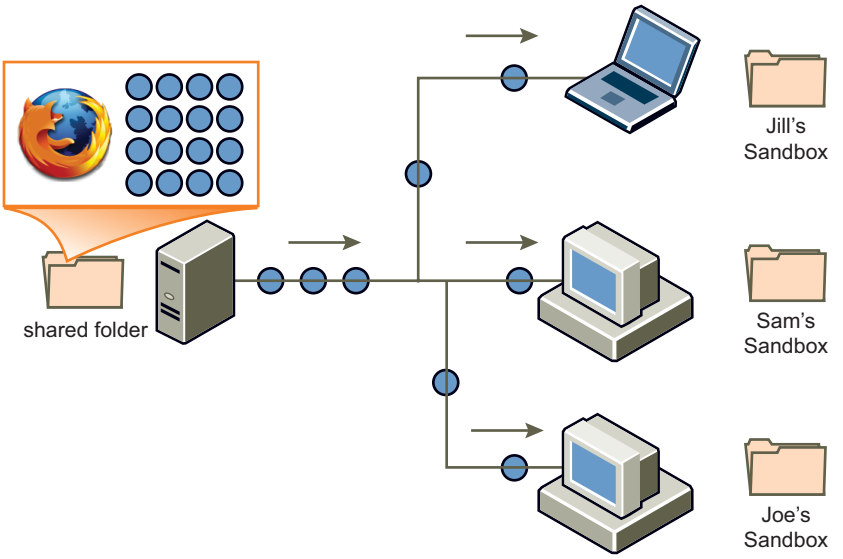
Any network storage device can serve as a streaming server for hundreds or thousands of client PCs.

To use ThinApp packages in a streaming fashion

1 Place your ThinApp package in a location that is accessible to client PCs.

2 Send a link to end users to run the application directly.

Figure 5-1. Block-Based Streaming Communicating over a Network Share



On the end user's desktop, you can also create shortcuts that point to the centrally hosted executable package(s). When the user clicks on the shortcut, the application begins streaming to the client PC. During the initial streaming startup process, the ThinApp status bar informs the user of the progress.

When the application can begin running, the status bar slides down and you can begin using the application. As more parts of the application are required, they are taken from the streaming server.

Network Requirements for Streaming ThinApp Applications

ThinApp does not require a special server software to provide streaming capability. Any Windows file share, NAS device, or SMB share can provide this capability. The amount of data that needs to transfer before the application can begin running varies widely. Microsoft Office 2003 requires that only a fraction of the package contents stream before an application can run.

VMware recommends that you use ThinApp streaming on LAN-based environments with 100MB networks.

For WAN and Internet deployments, in which unexpected disconnections are more frequent, VMware recommends one of the following solutions:

- Deploy applications by using a URL.
- Use a desktop deployment solution to push the package to the background. Allow it to run only after the entire package downloads.

These solutions reduce failures and eliminate situations in which the application requires unstreamed portions during a network outage. A company with many branch offices typically designates one application repository that mirrors a central shared folder at each branch office. This setup optimizes local performance for clients that are located at the branch office.

Security and Streaming

For security purposes, VMware recommends that you make a central shared directory read-only. Users can then read the package contents but not change the executable contents. As multiple users stream a package from a shared location, changes that the application makes are stored in the user's sandbox. A user's sandbox location defaults to `%AppData%\Thinapp\APPLICATION_NAME`. You can configure the sandbox location at runtime or at package time.

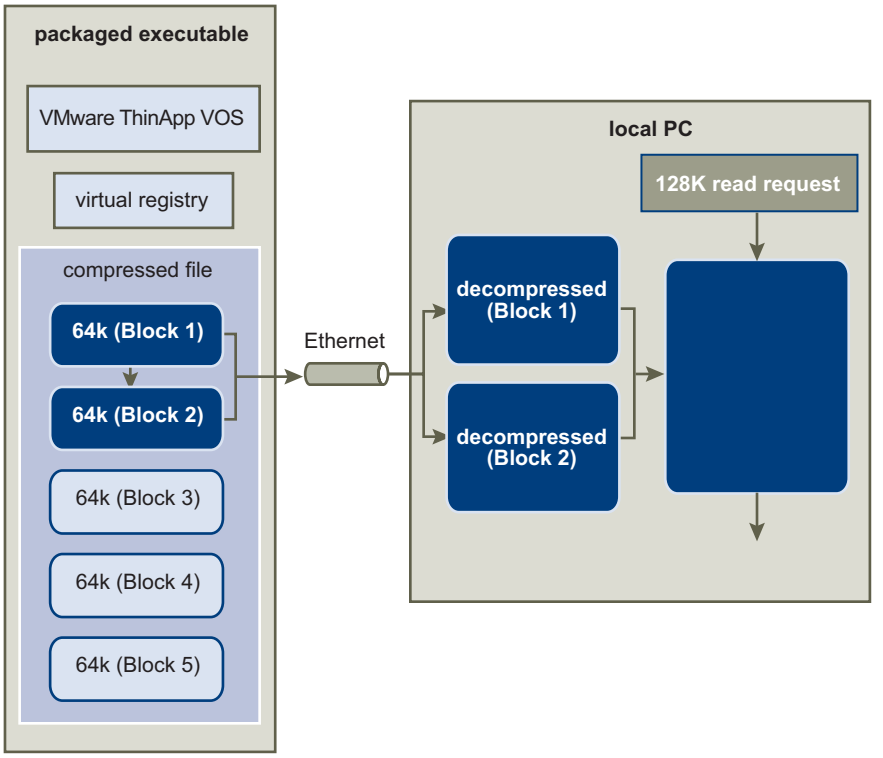
A common configuration is to locate the user's sandbox on another central storage device. The user can use any PC and retain individual application settings at a central share. When packages are streaming from a central share, they remain locked until all users have exited the application. ThinApp provides a mechanism for updating applications while they are still running. For more information, see "Upgrading Captured Applications" on page 87.

How ThinApp Application Streaming Works

When you place compressed ThinApp executable files on a network share or USB Flash drive, the contents from the executable file stream to client PCs in a block-based fashion. As an application requests specific parts of data files, this information is read in the compressed format over the network using standard Windows file-sharing protocol.

After a client PC receives data, the data is decompressed directly to memory and provided to the application. Because no data is written to disk, the process is fast. Also, client PCs need minimal disk space to host the operating system. When the application makes subsequent read requests for the same data, the Windows disk cache provides data without requiring a network read operation. If the client PC runs low on memory, Windows discards some of its disk cache and provides the memory resource to other applications.

Figure 5-2. Application Streaming



A large package does not necessarily take a long time to load over the network. The size of the package does not affect the startup time of an application. If you add an extra 20GB file to a package that is not used at runtime, the package loads at the same speed. If the application opens and reads 32K of data from the 20GB file, only 32KB of data is sent.

The client is part of the executable package, but it is small (400K). The load time of the client across a network is a few milliseconds. After the client is loaded to memory on the client PC, the client calculates which blocks of data are required from the server and reads them based on application activity.

Using Captured Applications with Other System Components

Captured applications can interact with other applications installed on the desktop. The following sections describe the various ways applications typically interact with the other components on the desktop.

Cut and Paste

- **Pasting from system installed applications to captured application**s – This scenario has no limitations. The virtual application can receive any standard clipboard formats (text, graphics, HTML, and so on). The virtual application can receive OLE objects.
- **Pasting from captured applications to system applications** – OLE objects created in virtual applications are converted to system native objects when they are pasted into nonvirtual applications.

Printers

A captured application has full access to any printer that is installed on the PC it is running on. Captured applications and system-installed applications have the same printing ability. You cannot use ThinApp to virtualize printer drivers. You must manually install printer drivers on a PC.

Drivers

A captured application has full access to any device driver installed on the PC it is running on. Captured applications and system-installed applications have the same relationship with installed device drivers. If an application requires a device driver, you must install the driver separately from the ThinApp package.

In some cases, an application without an associated driver might function correctly but in a limited fashion. For example, Adobe Acrobat installs a printer driver that allows applications system-wide to render PDF files using their print mechanism. When you use a captured version of Adobe Acrobat, you can use it to load, edit, and save PDF files without installation. Other applications do not detect a new printer driver unless the driver is installed.

Access to Local Disk, Removable Disk, and Network Shares

When you create a project structure, ThinApp configures isolation modes for directories and registry subtrees. The isolation modes control which directories the application can read and write to on the local PC. The default configuration options are:

- **Hard disk (for example, `c:\`)** – The user can write to their `Desktop` and `My Documents` folders. Other modifications that the application makes go into the user application sandbox. The sandbox is located by default in the `Application Data` directory.
- **Removable disk** – By default, users can read or write to any location on a removable disk assuming the user has access rights to do so.
- **Network mapped drives** – By default, users can read or write to any location on a network mapped disk assuming the user has access rights to do so.
- **UNC network paths** – Users can read or write to any location on a removable disk if the user has access rights to do so.

Access to the System Registry

By default, captured applications can read the full system registry as permitted by access permissions. Specific parts of the registry are isolated from the system during the package creation process. This isolation reduces conflicts between different versions of virtual applications and system-installed applications. By default, all registry writes from captured applications are saved in an isolated sandbox and the system remains unchanged.

Networking and Sockets

Captured applications have normal access to networking capability. Captured applications can bind to local ports and make remote connections if the user has access permissions to perform these operations.

Shared Memory Named Pipes

Captured applications can interact with other applications on the system by using shared memory, named pipes, mutex objects, and semaphores.

ThinApp can isolate shared memory objects and synchronization objects. This isolation makes them invisible to other applications, and other application objects are not visible to a captured application.

COM, DCOM, and Out-of-Process COM

Captured applications can create COM controls from the virtual environment and the system. If a COM control is installed as an out-of-process COM, the control runs as a virtual process when a captured application uses it. You can control modifications that the captured applications make.

COM objects that virtualized applications provide are invisible to other applications on the system unless the application is running in the same virtual environment. You can run system applications in a virtual environment so they can access COM objects that virtual environments provide. For example, a system-installed application can access virtualized Microsoft Office components by running the system application in the virtualized environment.

Services

Captured applications can start and run system-installed services and virtual services. By default, system services are run in the virtual environment so that the virtual environment controls modifications that the services make.

File Type Associations

Captured applications can execute system- installed applications using file type association. File type association can be added to the local PC's registry to point to captured executable files on a per-user or per-machine basis.

Access Control by Using Active Directory

You can control access to applications using Active Directory groups. When you build a package, ThinApp converts Active Directory group names into SID values. A SID is a small binary value that uniquely identifies an object, similar to a GUID. SIDs are not unique for a few special groups, such as the administrator group. Because SID values are stored in packages for later validation, the following considerations apply:

- You must be connected to your Active Directory domain during the build process and the groups you specified must exist. ThinApp looks up the SID value during build.
- If you delete a group and recreate it, the SID might change so you need to rebuild the package in order to authenticate against the "new" group.
- When users go offline, they can be authenticated using cached credentials. If the user can log into his or her machine, ThinApp Active Directory authentication still works. You can set the period of time cached credentials are valid using a group policy.
- Cached credentials might not refresh on clients until the next Active Directory refresh cycle. You can force a group policy on a client by using the `gpupdate` command. This command refreshes local group policy settings as well as group policy settings and security settings that are stored in Active Directory. Sometimes you might need to log off before Active Directory credentials can be recached.
- Special groups like the administrator group and the everyone group have the same SID on every Active Directory domain and workgroup. Other groups you create have a domain-specific SID. Users cannot create their own local group with the same name to bypass authentication.

Example Package.ini Files for Active Directory Access Control

In the following example, App1 and App2 will both inherit PermittedGroups from `BuildOptions`.

```
[BuildOptions]
PermittedGroups=Administrators;OfficeUsers
[App1.exe]
...
..

[App2.exe]
...
...
```

In the following example, App1 users can use `App1.exe` and members of the `Everyone` group can use `App2.exe` The default message for denied user is also changed for App1.

```
[BuildOptions]
PermittedGroups=Everyone

[App1.exe]
PermittedGroups=App1Users
AccessDeniedMsg=Sorry, you can't run this application
..
[App2.exe]
...
...
```

Application Updates

If an application has built-in auto updating capabilities, its update mechanism still functions when running under ThinApp. If the application downloads and runs an installer or patching program, this runs inside the virtual environment, and all of the changes made by the update software are written to the sandbox. When the application restarts, it uses the version of the executable in the sandbox, and not from the original package.

For example, if you capture Firefox 1.5, your auto update mechanism might ask if you want to upgrade to Firefox 2.0. If you do, it downloads updates, writes them to the sandbox, and prompts you to restart. When you run the captured application again, you are running Firefox 2.0. If you delete the sandbox, Firefox reverts back to version 1.5.

The VMware ThinApp Sandbox Merge utility can be used to merge changes made by an application's update mechanism into the original package to build an executable to redeploy to end user machines. For more information on sbmerge, see "Using Sandbox Merge (sbmerge)" on page 127.

ThinApp opens new possibilities for updating applications dynamically without requiring administrator rights. For example, .NET-based applications that download new DLL files from the Internet as part of their update process need to execute `ngen.exe` to generate native image assemblies in order to improve startup performance. Normally `ngen.exe` writes to `HKLM` and `c:\windows`, both of which are off-limits for non-administrator accounts. When running under VMware ThinApp, `ngen.exe` can install Native Image assemblies even on guest user accounts. However these changes are stored in a user-specific directory.

You might want to update the package on a central computer and push the changes to clients or central network shares as a new captured executable file.

Create a ThinApp project that contains the application updates. Choose one of these approaches:

- Apply patches during capture.
- Apply patches inside the virtual environment. If any application offers auto-update capabilities, it can update itself as described earlier. If the application update is in the form of a `patch.exe` file, the patch program can be run in the virtual environment, creating a `cmd.exe` entry point and running the patch program from there. Changes occur in your sandbox during auto-update or manual update so you can easily revert to the original version by deleting the sandbox. You can apply patches in the virtual environment either on an end user desktop machine or on a central packaging machine. In the second case, sandbox changes made by the patch or update can be merged into the application using the ThinApp sandbox merge utility.
- Apply patches to already captured project. If you need to update a small set of specific files or registry keys, replace the files in your captured project. This approach is well suited for software developers who can integrate ThinApp builds into their normal process.

Alternatively, you can use Application Sync to synchronize updates to the captured applications you have deployed. Application Sync enables you to set your captured applications to periodically check for updates. If you make an update to a specified captured application available, Application Sync initiates the update process and updates deployed applications. For more information about Application Sync, see "Application Sync" on page 58.

NOTE If you use Application Sync to perform application updates, make sure you disable the auto update feature of your captured application.

Upgrading Captured Applications

ThinApp allows you to upgrade or roll back an application version that it is still running. The upgrade process occurs automatically when the user quits the application and runs it a second time. In Terminal Server environments, you can have multiple users executing different versions at the same time during the transition period. The process for doing in-place upgrades or roll backs is very simple and is described in the following sections.

Locked Files

When applications are running, the executable package is locked and it cannot be replaced, deleted, or moved. This file lock ensures that any computer or user who accesses a specific version of an application continues to have that version available as long as their application processes and subprocesses are running.

If an application is hosted in a central location and run by many users, this file lock means a user cannot replace a packaged executable with a new version until all users have released their locks on the file.

Deploying Application Upgrades

ThinApp has an upgrade mechanism that enables you to deploy application upgrades even while older versions are still in use by users. This mechanism is as simple as copying the new version into the old deployment directory with a. .1, .2, .3, ... filename extension.

The following is an example:

1 Deploy Version 1.0 of `myapp.exe`.

   Copy the application to a central share at `\\server\share\myapp.exe` (`c:\Program Files\myapp\myapp.exe` is also another typical location)

   Push a desktop or start menu shortcut to the user's desktop that points to a shared executable location at `\\server\share\myapp.exe.`

2 After some time, two users (user1 and user2) launch `myapp.exe` and continue to use it.

3 Now, you'd like to deploy Version 2.0 of `myapp.exe`, but you cannot because the file is locked by 2 users.

   Copy Version 2.0 of `myapp.exe` to the central share at `\\server\share\myapp.1.`

After step 3, any time a new user executes `\\server\share\myapp.exe`, it automatically relaunches using the package data in `myapp.1`.

Shortcuts do not need to be updated. When you upgrade an application, the shortcuts continue to point to `myapp.exe`.

If user1 and user2 quit the original version 1.0 of `myapp.exe` and restart it, they are using version 2.0.

You can deploy version 3.0 of `myapp.exe` by placing it in the same directory with a higher number at the end.

Copy Version 2.0 of `myapp.exe` to central share at `\\server\share\myapp.2`

Once `myapp.1` becomes unlocked, you can delete it, but `myapp.exe` should remain in place since the user shortcuts continue to point there. ThinApp always uses the filename that has the highest version number. If you need to perform a roll back to an earlier version and the most recent version is still locked, copy the old version so it has the highest version number.

Sandbox Considerations

When you release an upgrade to an application, you can control whether the user continues to use his or her previous settings by keeping the sandbox name consistent in your `Package.ini` file. By deploying a new application version with a new sandbox name, each user creates a separate sandbox and starts with a new installation maintaining isolated settings. If you deploy the package with the same sandbox name, users keep their previous settings.

Configuring Isolation Modes

6

This chapter describes isolation modes and contains the following topics:

Understanding Isolation Modes

Isolation modes are used to control the read and write access for specific system directories and system registry subkeys.

ThinApp's Isolation Modes help solve the following classes of problems:

Problem 1: An application fails to run due to previous or future versions existing simultaneously or not uninstalling correctly.

Solution: ThinApp hides host PC files and registry keys from the application when the host PC files are located in the same directories and subkeys created by the application's installer. ThinApp calls this full isolation. For directories and subkeys that have full isolation set, the applications will only see virtual files and subkeys. Any system values that may exist at the same location will be invisible to the application.

Problem 2: An application fails because it was not designed or tested for multi-user environments. The application should be able to modify files and keys without impacting other users.

Problem 3: An application fails because it should have write permission to global locations and was not designed for locked-down desktop environments found in corporate environments or Windows Vista.

Solution to Problems 2 and 3: ThinApp makes copies of registry keys and files written by the application and performs all the modifications in a user-specific sandbox. ThinApp calls this WriteCopy Isolation. For directories and subkeys that have WriteCopy isolation, the application recognizes both the host PC's files and virtual files. However, all write operations convert host PC files into virtual files in the sandbox.

ThinApp has three different isolation modes that are automatically determined by Setup Capture. Setup Capture has some rules for determining what isolation mode to apply to a registry subtree or directory during capture:

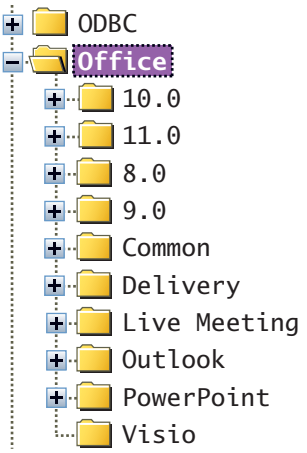
- If the application created a new directory or registry subtree during its installation (on a clean PC), the isolation mode is set to full isolation.
- User-specific storage areas like `Desktop` and `My Documents` are set to merged isolation so the application has direct write access to these locations.
- All other directories and subkeys default to WriteCopy isolation.

NOTE Network shares are not affected by isolation modes. Read and write operations to network shares occur unchanged by ThinApp.

For example, the following image shows a section of the Windows registry for a PC that has various older Office applications installed. Office 2003 creates the registry subtree:

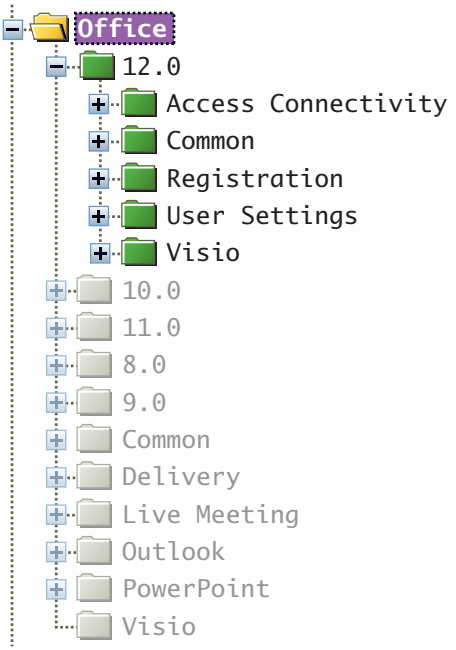
`HKEY_LOCAL_MACHINE\Software\Microsoft\Office\11.0`

The following is the registry as seen by Windows Regedit:



When running a captured version of Visio 2007, ThinApp sets the `HKLM\Software\Microsoft\Office` registry subtree to full isolation. This setting prevents Visio 2007 from failing because of registry settings that might pre-exist on the host PC at the same location.

The following is the registry as seen by a captured Visio 2007 application:



Working with Isolation Modes

Isolation modes can be controlled on a per-directory and per-registry subtree basis. By default, Setup Capture produces projects that are configured for a locked-down desktop model. In a locked-down PC environment, users typically have write permission only to their desktop, their `My Documents` folder, network shares, and removable disks.

The default isolation mode for the project can be changed in the `Package.ini` file. This property is inherited by child directories unless you override the isolation mode. For non-locked-down desktop models, setting the default isolation mode in the `project.ini` file to Merged instead of WriteCopy enables the user to save files directly to the host file system at any location.

The main risk for changing the default isolation mode to Merged is the potential for the application to leave behind residue when executing. For example, many shareware applications try to leave first-execution markers in random locations on the PC as part of their licensing system. Any disk writes to directories that are set with `WriteCopy` or `Full` isolation is redirected to the sandbox. The following Isolation modes can be applied to subdirectories and registry subkeys:

Merged

- System elements at this location are visible to the application. If a system element and virtual element exist at the same location, the application sees the virtual element.
- Modifications to virtual elements go to the sandbox.
- Modifications to system elements go to the system.
- New elements are created on the system.

WriteCopy

- System elements at this location are visible to the application. If a system element and virtual element exist at the same location, the application sees the virtual element.
- Modifications to virtual elements go to the sandbox.
- Modifications to system elements go to the sandbox.
- New elements are created in the sandbox.

Full

- System elements at this location are not visible to the application.
- Modifications to virtual elements go to the sandbox.
- System elements cannot be read or modified.
- New elements are created in the sandbox.

To change the isolation mode for a directory, edit the `##Attibutes.ini` file in your package located in the directory where you want to modify the default isolation mode.

Edit the HKEY_XXXXXX.txt file to change the isolation mode for a registry subtree. Each registry subtree begins with the isolation mode.

NOTE ThinApp caches the isolation modes for the registry and file system at runtime in the sandbox. If you change the isolation mode for your project and then rebuild the executable file, you might need to delete the sandbox for the change to take effect.

Using the Virtual File System

7

This chapter describes the ThinApp virtual file system. The virtual file system is represented in three different stages:

- The package build format. This format stores files directly on the normal file system as they are to be compiled into the embedded virtual file system. Folder Macros ("Folder Macros" on page 96) are used to represent Windows standard shell folder locations.
- The embedded virtual file system (read only). This file system is embedded in executable files during the build process. The read only file system might be compressed and provides block-based streaming to client PCs.
- The file system sandbox (read write). This directory structure holds file data that was modified by the application. Any file modification operation causes embedded virtual files to be extracted to the sandbox, including the following:
  - Changing a file's timestamp or attributes
  - Opening a file with write access
  - Truncating a file
  - Renaming or moving a file

Prior to building an application, you can adjust isolation modes on a per-directory basis using `##Attributes.ini` files. Isolation modes enable you to specify whether the application is presented with a merged view of the virtual file system or only see virtual files. Isolation modes also control what happens when an application tries to write to a directory. For more information on isolation modes, see Chapter 6, "Configuring Isolation Modes," on page 89.

Both the embedded and sandbox file systems uses folder macros to enable file paths to dynamically expand at runtime.

Folder Macros

ThinApp uses macros to represent file system path locations that might relocate when running on different operating systems or PCs. The table below list the available macros. When creating a new ThinApp project using Setup Capture, these macros are automatically hard-coded paths with macro forms. See Table 7-1 for example folder macros.

Table 7-1. Folder Macros

Macro Name	Shfolder.dll ID	Typical Location
%AppData%	CSIDL_APPDATA	`C:\Documents and Settings\username\ Application Data`
%Common AppData%	CSIDL_COMMON_APPDATA	`C:\Documents and Settings\All Users\Application Data`
%Common Desktop%	CSIDL_COMMON_DESKTOPDIRECTORY	`C:\Documents and Settings\All Users\Desktop`
%Common Favorites%	CSIDL_COMMON_FAVORITES	`C:\Documents and Settings\All Users\Favorites`
%Common Programs%	CSIDL_COMMON_PROGRAMS	`C:\Documents and Settings\All Users\Start Menu\Programs`
%Common StartMenu%	CSIDL_COMMON_STARTMENU	`C:\Documents and Settings\All Users\Start Menu`
%Common Startup%	CSIDL_COMMON_STARTUP	`C:\Documents and Settings\All Users\ Start Menu\Programs\ Startup`
%Desktop%	CSIDL_DESKTOPDIRECTORY	`C:\Documents and Settings\username\ Desktop`
%Drive_c%		`C:\`

Table 7-1. Folder Macros (Continued)

Macro Name	Shfolder.dll ID	Typical Location
%Drive_m%		M:\
%Favorites%	CSIDL_FAVORITES	C:\Documents and Settings\username\ Favorites
%Fonts%	CSIDL_FONTS	C:\Windows\Fonts
%Local AppData%	CSIDL_LOCAL_APPDATA	C:\Documents and Settings\username\ Local Settings\ Application Data
%My Pictures%	CSIDL_MYPICTURES	C:\Documents and Settings\username\My Documents\My Pictures
%My Videos%	CSIDL_MYVIDEO	C:\Documents and Settings\username\My Documents\My Videos
%NetHood%	CSIDL_NETHOOD	C:\Documents and Settings\username\ NetHood
%Personal%	CSIDL_PERSONAL	C:\Documents and Settings\username\My Documents
%Profile%	CSIDL_PROFILE	C:\Documents and Settings\username
%Profiles%	CSIDL_PROFILES	C:\Documents and Settings
%Program Files Common%	CSIDL_PROGRAM_FILES _COMMON	C:\Program Files\Common Files
%ProgramFilesDir%	CSIDL_PROGRAM_FILES	C:\Program Files
%Programs%	CSIDL_PROGRAMS	C:\Documents and Settings\username\ Start Menu\Programs
%Recent%	CSIDL_RECENT	C:\Documents and Settings\username\My Recent Documents
%SendTo%	CSIDL_SENDTO	C:\Documents and Settings\username\ SendTo

Table 7-1. Folder Macros (Continued)

Macro Name	Shfolder.dll ID	Typical Location
%Startup%	CSIDL_STARTUP	C:\Documents and Settings\username\ Start Menu\Programs\ Startup
%SystemRoot%	CSIDL_WINDOWS	C:\Windows
%SystemSystem%	CSIDL_SYSTEM	C:\Windows\System32
%TEMP%		C:\Documents and Settings\<username>\ Local Settings\Temp
%Templates%	CSIDL_TEMPLATES	C:\Documents and Settings\username\ Templates

Macros requiring shfolder.dll version 5.0 or higher include %ProgramFilesDir%, %Common AppData%, %Local AppData%, %My Pictures%, and %Profile%. Macros requiring shfolder.dll version 6.0 or higher include %My Videos%, %Personal%, and %Profiles%.

ThinApp uses shfolder.dll to obtain the location of Shell folders. Older versions of shfolder.dll do not support some macro names. If there is a requirement for a specific version of shfolder.dll on the host OS, it is listed.

Special processing for %SystemRoot% When running in a Terminal Services environment, there are two Windows directories, a shared Windows directory (for example, C:\Windows) and a private Windows directory (for example, C:\Documents and Settings\User\Windows). For Terminal Services environments, ThinApp uses the user-specific directory for %Systemroot%.

8 Using the Virtual Registry

Operating systems and applications store a large amount of settings in the system registry. The virtual operating system intercepts requests to open files and redirects them to the virtual file system, it also intercepts requests to look up and store values in the registry and redirects them to the virtual registry.

By storing the settings in the virtual registry, VMware ThinApp ensures that the settings that are needed by the application to function correctly are available to it, without actually changing the system registry.

ThinApp registry information exists in three forms:

- Build format—Unicode text files such as `HKEY_LOCAL_MACHINE.txt`, the build process converts this format into the embedded format.
- Embedded format (read only)—Compressed and stored inside the application executable file in a binary format.
- Sandbox format (read-write)—Stores the differences from the embedded format as the application makes registry writes.

> **NOTE** Because applications can be upgraded by dropping in a new file in the same directory as the main application, the directory (and not just the file) where an application is hosted must be read-only for normal users.

After you build a package, all registry values and files exist in a read-only image inside of the executable you are distributing. At runtime, the read-only image presents a view of the registry for new users of a package. As the application modifies the registry, these changes are saved in the sandbox persistent file with the extension .tvr. The VRegTool utility can be used to list and modify data contained in a `.tvr` file.

Build Format

The build format shows what registry data looks like after you perform Setup Capture. In this format, each registry subkey maps directly to a directory on your build file system. Each registry value also maps directly to a file. An example view looks like this:

```
c:\mypackage\HKEY_LOCAL_MACHINE\Software\ApplicationX\Value1.txt
c:\mypackage\HKEY_CLASS_ROOT\.txt
```

Because registry subkeys and value names can have some characters that do not map to filenames, any unmappable character must be escaped in the filename like this:

Registry Subkey: `HKEY_CLASS_ROOT\*.*`

Maps to filename: `c:\mypackage\HKEY_CLASS_ROOT\#42.#42.`

Because the character * is not a valid filename character, it has been escaped as #42.

Because # is used to escape characters, it must also be escaped.

To represent a default value, the filename `##default` is used.

Because FAT32 has severe limits on maximum filename lengths, store all build registry data either on an NTFS drive or use a short parent directory name to store registry build data. At runtime, FAT32 limitations do not impact the registry.

Registry Value Data

Registry files are saved as plain text or unicode text. Unicode text files start with the two-byte sequence hex(ff), hex(fe). Notepad and Wordpad automatically create unicode and ANSI text files.

```
TYPE=VALUE
```

TYPE can be one of the following:

```
REG_NONE, REG_SZ, REG_EXPAND_SZ, REG_BINARY, REG_DWORD,
REG_DWORD_LITTLE_ENDIAN, REG_DWORD_BIG_ENDIAN, REG_MULTI_SZ,
REG_RESOURCE_LIST, REG_FULL_RESOURCE_DESCRIPTOR,
REG_RESOURCE_REQUIREMENTS_LIST
```

If TYPE is string data (REG_SZ, REG_EXPAND_SZ, or REG_MULTI_SZ), then VALUE is listed as a sequence of characters terminated by a new line or carriage return character. Escape any unprintable characters into two hex characters:

```
REG_SZ=This is a line#0awith a carriage return as part of the data#00
REG_MULTI_SZ=This is a line1#00This is line2#00#00
```

Because the # represents an escape character, all # characters must also be escaped. The % character is used for ThinApp macro expansion, it must also be escaped when the value must remain unexpanded.

Example of Macro Expansion

```
REG_SZ=%ProgramFilesDir%\ApplicationX\control.ocx
```

At runtime, when the program accesses this registry value, it is expanded by ThinApp.

Text Format for Virtual Registry

The text format for the virtual registry is similar to regedit `.reg` format, however, it supports macro expansion for string registry values names and value data.

The format looks like the following:

```
isolation_mode FULL_SUBKEY_NAME

Value=VALUE_NAME
VALUE_TYPE=VALUE_DATA
```

or

```
Value=VALUE_NAME
VALUE_TYPE~MACRO_VALUE_DATA
```

`isolation_mode`—This can be one of ThinApp's isolation modes as specified by `isolation_full`, `isolation_merged`, or `isolation_writecopy`.

`VALUE_TYPE`—This specifies the value data registry type. Use one of the following:

```
REG_SZ, REG_EXPAND_SZ, REG_BINARY, REG_DWORD, REG_DWORD_BIG_ENDIAN,
REG_LINK, REG_MULTI_SZ,
REG_RESOURCE_LIST, REG_FULL_RESOURCE_DESCRIPTOR,
REG_RESOURCE_REQUIREMENTS_LIST
```

If you need to represent a non-published registry type, you can specify a decimal (base 10) number. For example:

```
453=#00
```

`VALUE_DATA`—Specifies value data. Data is represented differently depending on if the data type is a string or non-string type.

String types (`REG_SZ, REG_MULTI_SZ, REG_EXPAND_SZ`)—Values are represented as escaped string values. A unicode text file must be used to represent non-ANSI characters. ThinApp uses unicode text files by default. The #, new line character, tab character, carriage return, end-of-file, and NULL characters must be escaped if they exist in the original string.

Escaped characters begin with # and are followed by two hex characters to represent the ANSI value for the character. Non-ANSI unicode characters do not need to be escaped but can only exist in unicode text files.

The following are examples:

`REG_SZ=Both#00`

This example represents a five-character string value, the fifth character being a NULL (0) character.

`REG_MULIT_SZ=String1#00String2#00#00`

This example represents a multi-string value that contains a list of two values `String1` and `String2`:

`REG_DWORD=#27#c6#00#02`

This example represents a 4-byte binary value.

For binary values, data is stored as a string of escaped bytes. For DWORD values, data is stored in native x86 order (little endian) so no special processing needs to be performed for binary data.

`MACRO_VALUE_DATA` — Specifies value data to be macro-expanded prior to use by the application. For example, the value `%AppData%` is expanded at runtime to the location of the user's `Application Data` directory. If value data is not macro-expanded, the application receives the literal string `%AppData%` when queries this registry value. Macro-expanded data is only supported for string value types (REG_SZ, REG_EXPAND_SZ, and REG_MULTI_SZ).

The following are examples:

`REG_SZ~%AppData%`

This represents the value of the Application Data folder on the host PC on which ThinApp is running.

The length of this registry value changes depending on what machine it runs on.

`REG_SZ~#23AppData#23`

This represents the value of the actual value `%AppData%`. This value is always the literal string `%AppData%` regardless of the current shell folder location.

```
isolation_full
HKEY_LOCAL_MACHINE\Software\Classes\CLSID\{047a9a40-657e-11d3-8d5b-
00104b35e7ef}\InprocServer32
Value=ThreadingModel
REG_SZ=Both#00
Value=
REG_SZ~%SystemSystem%\mscoree.dll#00
```

##Attributes.ini

This file controls per-registry subkey settings.

RegistryIsolationMode=WriteCopy | Merged | Full

The following is an example:

```
RegistryIsolation=WriteCopy
```

If isolation is not specified, it is inherited from parent subkey.

The following is an example of command-line usage:

NOTE All parameters are case insensitive.

```
VREGTOOL regfile.tvr ExportDir output_directory [HKEY_LOCAL_MACHINE\Software]
VREGTOOL regfile.tvr ImportDir input_directory
VREGTOOL regfile.tvr ImportReg regedit.reg [–Merged|–WriteCopy|–Full]
[–NoReplace][–NoMacros]
VREGTOOL regfile.tvr ExportReg filename.reg [HKEY_LOCAL_MACHINE\Software]
VREGTOOL regfile.tvr PrintKeys [HKEY_LOCAL_MACHINE\Software] [–ShowValues]
[–ShowData] [–ExpandMacros]
VREGTOOL regfile.tvr PrintStats
VREGTOOL regfile.tvr SysCompare [HKEY_LOCAL_MACHINE\Software] [–Exact]
VREGTOOL regfile.tvr DelSubkey HKEY_LOCAL_MACHINE\Software [–NoMark]
```

Exporting Registry Data to ThinApp Registry Directory Format

Use the following commands to export registry data to ThinApp registry directory format:

```
VREGTOOL regfile.tvr ExportDir output_directory registry_tree
[HKEY_LOCAL_MACHINE\Software]
```

`regfile.tvr` : Data file in ThinApp virtual registry file format.

`output_directory registry_tree`: Output directory where registry data is written

`HKEY_LOCAL_MACHINE\Software` : You can optionally specify a registry subtree to export. If no subtree is specified, ThinApp exports the entire contents of the `.tvr` file. If the specified registry subkey has a space in the name, specify the key using quotes like this: `"HKEY_LOCAL_MACHINE\Software\Key with space"`

Importing Registry Data from ThinApp Registry Directory Format

Use the following commands to import registry data from ThinApp registry directory format:

```
VREGTOOL regfile.tvr ImportDir input_directory
```

where ImportDir is a keyword.

`regfile.tvr` : Data file in ThinApp virtual registry file format

`input_directory` : Input directory in which registry data is read

The following is an example:

```
vregtool c:\tmp\test.tvr ImportDir
"C:\jc\code\VOS2\thinapp\setup_capture\Debug\Captures\05-07-2006 01.14"
```

Importing Registry Data from Regedit Format

Use the following commands to import registry data from Regedit format:

```
VREGTOOL regfile.tvr ImportReg regedit.reg [-Merged|-WriteCopy|-Full]
               [-NoReplace][-NoMacros]
```

`regfile.tvr` : Data file in ThinApp virtual registry file format.

`regedit.reg` : The `.reg` file to import, entries imported are added to the specified `.tvr` file. This file can be in REGEDIT 4.0 (ansi text) or 5.0 (unicode text) format.

Isolation Mode options. If no isolation mode is specified, `WriteCopy` is used.

`Merged` : Specifies that registry keys that do not already exist have isolation mode set to `Merged`.

`WriteCopy`: Specifies that registry keys that do not already exist have the isolation mode set `toWriteCopy`.

`Full` : Specifies that registry keys that do not already exist have the isolation mode set to `Full`.

`NoReplace` : Do not replace or modify existing registry values. If this option is not selected, when a `.tvr` already contains a value specified in the `.reg` file, this value is overwritten with the value specified in the `.reg` file.

`NoMacros` : Do not perform macro substitution for registry values that contain paths to short path name or shell folders. When this flag is not set, the values contained in `.tvr` are replaced with macro versions of paths.

For example, if the `.reg` file contains a string registry value of `C:\windows\system32\kernel32.dll`, the `.tvr` file contains the value `%systemsystem%\kernel32.dll`. When the application requests the value of this registry key, it receives the value `C:\windows\system\kernel32.dll` when running on Windows 98, ME, and XP+. If the application runs on Windows NT or Windows 2000, it receives the value `C:\winnt\system32\kernel32.dll`.

Usually macro substitution is preferable. However, in some cases, you might want to disable this and store a hard-coded path in the registry.

Exporting Registry Data to Regedit Format

Use the following commands to export registry data to Regedit format:

```
VREGTOOL regfile.tvr ExportReg filename.reg [HKEY_LOCAL_MACHINE\Software]
```

`regfile.tvr` : Data file in ThinApp virtual registry file format.

`regedit.reg` : The `.reg` file to export to. The file produced is in REGEDIT 5.0 format (unicode text).

`HKEY_LOCAL_MACHINE\Software` : You can optionally specify a registry subtree to export. If no subtree is specified, vregtool exports the entire contents of the `.tvr` file. If the specified registry subkey has a space in the name, you should specify the key using quotes like this: `"HKEY_LOCAL_MACHINE\Software\Key with space."`

When exporting to .reg format, some information is lost, including the following:

- Filename macros are expanded on export. If they are not converted back to macros when you import, information might be lost.
- Each Registry subkey in .tvr format has a specified isolation mode. Because .reg format does not have a concept of isolation modes or metadata for subkeys, this information is lost.
- Registry values that cannot be represented in .reg format are lost. For example, a key that is REG_SZ cannot have more than one NULL character in .reg format. In such a case, the registry value data is prematurely truncated in .reg format. Another example where .reg files cannot represent values accurately is a REG_SZ value that is not null-terminated. For these reasons, it is preferable to export registry data to ThinApp registry directory format and then re-import into the `.tvr` file, when you need to process the data.

The following is an example:

```
vregtool c:\tmp\test.tvr ExportReg c:\tmp\test.reg
"HKEY_CURRENT_USER\Software\Adobe\Save For Web 3.0"
```

Listing all Registry Keys in a ThinApp .tvr File

Use the following commands to list all registry keys in a ThinApp .tvr file:

```
VREGTOOL regfile.tvr PrintKeys [HKEY_LOCAL_MACHINE\Software] [–ShowValues]
[–ShowData] [–ExpandMacros]
```

`regfile.tvr` : Data file in ThinApp virtual registry file format

`HKEY_LOCAL_MACHINE\Software` : You can optionally specify a registry subtree to print. If no subtree is specified, vregtool prints the entire contents of the `.tvr` file. If the specified registry subkey has a space in the name, your should specify the key using quotes like this: `"HKEY_LOCAL_MACHINE\Software\Key with space"`

`ShowValues` : Optionally prints the names of all virtual values contained in virtual subkeys

`ShowData` : Optionally prints the data associated with each virtual value

`ExpandMacros` : Optionally expands macros contained in registry values and data before printing

The following is an example:

```
vregtool c:\tmp\test.tvr PrintKeys "HKEY_CURRENT_USER\Software\Adobe\Save For
Web 3.0"
```

This operation prints all the virtual registry keys contained in a `.tvr` file to the console.

Listing Diagnostic Information About a Thinapp.tvr File

Use the following command to list diagnostic information about a ThinApp .tvr file.

```
VREGTOOL regfile.tvr PrintStats
```

`regfile.tvr` : Data file in ThinApp virtual registry file format

This option is used mainly for diagnosing issues with .tvr files

The following is an example:

```
vregtool c:\tmp\test.tvr PrintStats
```

Comparing VIrtual Registry Information with Host PC Registry Information

Use the following command toComparing VIrtual Registry Information with Host PC Registry Information

```
VREGTOOL regfile.tvr SysComapre [HKEY_LOCAL_MACHINE\Software] [–Exact]
```

`regfile.tvr` : Data file in ThinApp virtual registry file format

`HKEY_LOCAL_MACHINE\Software` : Optionally specifies the registry subkey to begin comparing. If this is specified, only subkeys at this level and below are considered in the comparison.

`–Exact`: Optionally specifies that system comparison reports all differences between the virtual registry and system registry. If this option is not specified, vregtool does not print the system registry keys that do not exist in the virtual registry if the virtual registry subkey is set to Merged or WriteCopy isolation mode (but prints the differences for subkeys with isolation mode set to “Full”).

This option is useful in finding differences between a working system registry and virtual registry file. The following is an example:

```
vregtool c:\tmp\test.tvr SysCompare "HKEY_CURRENT_USER\Software\Adobe"
```

Deleting a Registry Subkey

Use the following command to delete a registry subkey:

```
VREGTOOL regfile.tvr DelSubkey HKEY_LOCAL_MACHINE\Software [–NoMark]
```

`regfile.tvr` : Data file in ThinApp virtual registry file format

`HKEY_LOCAL_MACHINE\Software` : Specified the registry subkey to delete. If the specified registry subkey has a space in the name, specify the key using quotes like this: `"HKEY_LOCAL_MACHINE\Software\Key with space"`

`NoMark` : Optionally specifies that the subkey should be deleted if it exists, but not marked as deleted. When a subkey is marked as deleted, applications running under ThinApp does not see the specified subkey even if it exists on the local system.

NOTE When `–NoMark` is not used, vregtool marks the subkey as deleted whether or not it originally exists in the `.tvr` file.

Example:

```
vregtool c:\tmp\test.tvr DelSubkey "HKEY_CURRENT_USER\Software\Adobe\Save For
               Web 3.0
```

Using ThinApp Utilities and Tools

9

This chapter describes ThinApp utilities and tools and contains the following topics:

ThinApp utilities are stored in the default installation directory (for example, `c:\Program files\Thinapp.VS`).

Using Log Monitor

Log Monitor enables you to record detailed information about any application's execution history for later review. A few things that are recorded:

- Win32 API calls made by applications running in the ThinApp VOS, along with parameter and results information
- A list of potential errors, exceptions, and security events within the application
- A list of all DLLs loaded by the application and address ranges

Log monitor can be found in your ThinApp installation directory (typically `C:\Program Files\Thinapp.VS`).

It can also be run by selecting the Log Monitor menu entry.

To run Log Monitor

1 **Start Menu > Program Files > VMware > ThinApp Log Monitor**.

Any new VMware ThinApp process that starts after Log Monitor, appears in the list if the application was built with logging options enabled or with "just-in-time" logging options enabled.

2 Select a process entry.

3 Click **`Delete`** to remove any trace files selected in the process list (#1). This does not delete any text .log files you might have generated.

4 Click **`Kill`** to stop a currently running process. This is an easy way to stop a process from logging additional entries after an error condition occurs.

5 Enter a a trace file name or click **`Browse`** to select a trace file on your system.

6 Enter an output report file or click **`Browse`** to select an output report file on your system. This file is generated when you click **`Generate text trace report`**. View the file with a text editor that supports UNIX-style line breaks such as Msdev, Wordpad, or Word. Do not use Notepad.

Locating Errors

ThinApp logging functionality (created using Log Monitor) provides a large amount of information. Here are a few tips you can follow to more easily investigate problems:

- Use TextPad to view your log files, as it can handle fairly large files. Most text file viewers cannot handle large log files, and ThinApp trace files can easily be larger than 100MB. If your trace file exceeds 200MB, you might need to cut the beginning of the file using a utility such as tail.exe.
- Look at the "Potential Errors" section first. The most important part of the log for diagnosing problems comes at the end of the log, in the section labeled "----Potential Errors Detected ---." It is located at the bottom of the `.txt` trace file.

  Many entries listed in the potential errors section do not actually indicate an error. ThinApp lists each Win32 API call where the Windows error code changed, and most of these do not indicate a real problem.

- Exceptions

  Many applications generate exceptions while they are running. Exceptions do not always indicate an error or fault, but are generally a good indicator. Exceptions have different types, (C++,.NET, Floating point, Access Violation, Data Execution Prevention, etc.). The trace file records the type of exception generated and the DLL that generated the exception. If the application created an exception from self-generating code, as is sometimes the case for .NET and Java applications, the trace indicates "unknown_module."

  The following is an example trace entry for an exception:

  ```
  *** Exception EXCEPTION_ACCESS_VIOLATION on read of 0x10 from
  unknown_module:0x7c9105f8
  ```

  NOTE Because VB6 applications throw many floating point exceptions during their normal execution, you can ignore these. If you find an exception, look higher in the trace file to see if you can locate the source of the exception.

- MSI error messages

  The MSI system starts for applications that are not installed correctly and tries to perform self repair. Unless a feature was installed using on-demand installation, the feature usually indicates that there is something wrong in the environment. Luckily, MSI provides good indicators and error messages in the trace file before it starts self-repair. Search your trace file for calls to *FormatMessage*, which provides you with important information from the MSI installer.

- The error might occur in a child process, make sure you look at the correct trace file.

  Log Monitor produces one trace file for each process that is started. If your application starts several child processes, the first step is to determine which process is causing the problem. In some cases such as out-of-process COM, a parent application uses COM to launch a child process, executes a function remotely, and then continues executing.

- When running applications from a network share, two processes are always created. Ignore the first one.

  To work-around Symantec AntiVirus slow performance issues, ThinApp always relaunches processes when running from a network share.

- Search for the error message you see in dialog boxes.

  Many applications call the Win32 API function MessageBox to display unexpected errors at runtime. You can search your trace file for either "MessageBox" or the contents of the string displayed in the error dialog. For example, if the application displayed a cryptic error message, such as "Unexpected child node / Press OK to continue," you could search for this string (or part of the string) and find what the application was doing just before the dialog box appeared.

- Terminate the application as soon as it encounters an error situation.

  By terminating the application as soon as possible, you have less information to sift through in the trace file. If you kill the application immediately after it has an error, chances are good that you can look at the end of the trace and scan upwards to see the source of the problem.

- You can generate `.txt` versions of `.trace` files while an application is still running.

  If needed, you can convert a `.trace` file into a text report while the application is still running.

- Narrow your focus on calls originating from a specific DLL and thread.

  ThinApp's log format ("Log Format" on page 112) enables you to see which DLL and thread is making a call. When tracking back from the source of an error, you can usually ignore other threads and often ignore calls from system DLLs.

Log Format

General API Log Message Format

The general format for API calls looks like this:

```
000257 0a88 mydll.dll :4ad0576d–>kernel32.dll:7c81b1f0 SetConsoleMode (IN
HANDLE
hConsoleHandle=7h, IN DWORD dwMode=3h)
000258 0a88 mydll.dll :4ad0576d<–kernel32.dll:7c81b1f0 SetConsoleMode
–>BOOL=1h ()
```

`000257`—Indicates the log entry number. Each log entry has a unique number. The "----Potential Errors Detected ---" refers to individual log entries using their log entry numbers.

`0a88`—Indicates the currently executing thread ID. If the application has only one thread, this number does not change.. Because log entries are recorded in order of execution, if two or more threads are recording data to the log file, you might need to use the thread ID to follow thread-specific sequential actions.

`mydll.dll`—Indicates the DLL that is making the API call.

`4ad0576d`—Indicates the return address for the API call made by mydll.dll. Typically this tells you the address in the code where the call is originating.

–> Indicates the call is being entered. For call entry log element, all of the input parameters are displayed (in and in/out parameters)

<–Indicates the call is returning to the original caller. For call exit log entries, all of the output parameters are displayed (out and in/out parameters).

`kernel32.dll`—Indicates the DLL where the API call is landing.

`7c81b1f0`—Indicates the address of the API inside of kernel32 where the call is landing. If you disassemble kernel32.dll at address 7c81b1f0, you find the code for the function SetConsoleMode

`–>BOOL=1h` Indicates the API returned the value 1 and the return code has type BOOL.

Application Startup Information

```
000001 0a88 Logging started for Module=C:\test\cmd_test\bin\cmd.exe
Using archive=
PID=0xec
CommandLine = cmd
000002 0a88 Logging options: CAP_LEVEL=9 MAX_CAP_ARY=25 MAX_CAP_STR=150
MAX_NEST=100
VERSION=3.090

000003 0a88 System Current Directory = C:\test\cmd_test\bin Virtual Current
Directory = C:\test\cmd_test\bin

000004 0a88 |start_env_var| =::=::\
000005 0a88 |start_env_var| =C:=C:\test\cmd_test\bin
000006 0a88 |start_env_var| =ExitCode=00000000
000007 0a88 |start_env_var| ALLUSERSPROFILE=C:\Documents and Settings\All
Users.WINDOWS
...
...
...
```

List of DLLs Loaded into Memory During Runtime

The section labeled "---Modules loaded ---" is located near the end of the log and describes all of the DLLs that were loaded into memory at runtime and the addresses they were loaded to. This list also describes whether DLLs were loaded by Windows or by ThinApp.

```
---Modules loaded --
PRELOADED_MAP 00400000-00452fff, C:\Program Files\Adobe\Reader
8.0\Reader\AcroRd32.exe
PRELOADED_BY_SYSTEM 00400000-00452fff, C:\Program Files\Adobe\Reader
8.0\Reader\AcroRd32.exe
SYSTEM_LOADED 00400000-00452fff, C:\test\AcroRd32.exe
SYSTEM_LOADED 00df0000-00df8fff, C:\WINDOWS\system32\Normaliz.dll
MEMORY_MAPPED_ANON 013b0000-020affff, C:\Program Files\Adobe\Reader
8.0\Reader\AcroRd32.dll
SYSTEM_LOADED 035a0000-035b4fff, C:\WINDOWS\system32\nvwddi.dll
SYSTEM_LOADED 035a0000-03698fff, C:\WINDOWS\system32\nvwimg.dll
SYSTEM_LOADED 035e0000-03ba9fff, C:\WINDOWS\system32\ieframe.dll
SYSTEM_LOADED 04730000-04828fff, C:\WINDOWS\system32\nvwimg.dll
MEMORY_MAPPED_ANON 05000000-050a8fff, C:\Program Files\Adobe\Reader
8.0\Reader\ACE.dll
MEMORY_MAPPED_ANON 06000000-064b0fff, C:\Program Files\Adobe\Reader
8.0\Reader\AGM.dll
MEMORY_MAPPED_ANON 07000000-07019fff, C:\Program Files\Adobe\Reader
8.0\Reader\BIB.dll
MEMORY_MAPPED_ANON 08000000-08239fff, C:\Program Files\Adobe\Reader
8.0\Reader\CoolType.dll
SYSTEM_LOADED 0ffd0000-0fff7fff, C:\WINDOWS\system32\rsaenh.dll
SYSTEM_LOADED 10000000-10165fff, C:\WINDOWS\system32\nview.dll
SYSTEM_LOADED 20000000-202c4fff, C:\WINDOWS\system32\xpsp2res.dll
 MEMORY_MAPPED_ANON 20800000-20fdafff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\AcroForm.api
MEMORY_MAPPED_ANON 22100000-224f2fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Annots.api
MEMORY_MAPPED_ANON 23000000-2311afff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\DigSig.api
MEMORY_MAPPED_ANON 23800000-23951fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\EScript.api
MEMORY_MAPPED_ANON 24000000-24023fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\EWH32.api
MEMORY_MAPPED_ANON 25800000-25817fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\IA32.api
MEMORY_MAPPED_ANON 26800000-2680ffff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\eBook.api
MEMORY_MAPPED_ANON 28000000-285d1fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\PPKLite.api
MEMORY_MAPPED_ANON 28800000-2885dfff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\reflow.api
MEMORY_MAPPED_ANON 29000000-29227fff, C:\Program Files\Adobe\Reader
```

```
8.0\Reader\plug_ins\MakeAccessible.api
MEMORY_MAPPED_ANON 29800000-2985afff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Accessibility.api
MEMORY_MAPPED_ANON 29a00000-29a1dfff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\ReadOutLoud.api
MEMORY_MAPPED_ANON 2a000000-2a018fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Search5.api
MEMORY_MAPPED_ANON 2a300000-2a359fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Search.api
MEMORY_MAPPED_ANON 2a800000-2a821fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\SendMail.api
MEMORY_MAPPED_ANON 2b000000-2b045fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Spelling.api
MEMORY_MAPPED_ANON 2b800000-2b865fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\PDDom.api
MEMORY_MAPPED_ANON 2d800000-2d94efff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Multimedia.api
MEMORY_MAPPED_ANON 2e000000-2e02ffff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\weblink.api
MEMORY_MAPPED_ANON 30800000-30829fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Updater.api
MEMORY_MAPPED_ANON 31800000-31810fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\HLS.api
MEMORY_MAPPED_ANON 32000000-3204dfff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\SaveAsRTF.api
MEMORY_MAPPED_ANON 40800000-40821fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\DVA.api
MEMORY_MAPPED_ANON 45800000-458cffff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Checkers.api
MEMORY_MAPPED_ANON 46800000-46873fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\ImageViewer.API
SYSTEM_LOADED 59a60000-59b00fff, C:\WINDOWS\system32\dbghelp.dll
SYSTEM_LOADED 5ad70000-5ada7fff, C:\WINDOWS\system32\uxtheme.dll
SYSTEM_LOADED 5d090000-5d129fff, C:\WINDOWS\system32\COMCTL32.dll
SYSTEM_LOADED 61410000-61533fff, C:\WINDOWS\system32\urlmon.dll
SYSTEM_LOADED 629c0000-629c8fff, C:\WINDOWS\system32\LPK.DLL
SYSTEM_LOADED 63380000-633f7fff, C:\WINDOWS\system32\jscript.dll
SYSTEM_LOADED 6e850000-6e894fff, C:\WINDOWS\system32\iertutil.dll
SYSTEM_LOADED 71aa0000-71aa7fff, C:\WINDOWS\system32\WS2HELP.dll
SYSTEM_LOADED 71ab0000-71ac6fff, C:\WINDOWS\system32\ws2_32.dll
SYSTEM_LOADED 71bf0000-71c02fff, C:\WINDOWS\system32\SAMLIB.dll
SYSTEM_LOADED 746c0000-746e8fff, C:\WINDOWS\system32\msls31.dll
SYSTEM_LOADED 746f0000-74719fff, C:\WINDOWS\system32\msimtf.dll
SYSTEM_LOADED 74720000-7476afff, C:\WINDOWS\system32\MSCTF.dll
SYSTEM_LOADED 74d90000-74dfafff, C:\WINDOWS\system32\USP10.dll
SYSTEM_LOADED 755c0000-755edfff, C:\WINDOWS\system32\msctfime.ime
SYSTEM_LOADED 75cf0000-75d80fff, C:\WINDOWS\system32\MLANG.dll
SYSTEM_LOADED 75e90000-75f3ffff, C:\WINDOWS\system32\SXS.DLL
SYSTEM_LOADED 76390000-763acfff, C:\WINDOWS\system32\IMM32.DLL
SYSTEM_LOADED 76780000-76788fff, C:\WINDOWS\system32\shfolder.dll
```

```
SYSTEM_LOADED 769c0000-76a72fff, C:\WINDOWS\system32\USERENV.dll
SYSTEM_LOADED 76b40000-76b6cfff, C:\WINDOWS\system32\WINMM.dll
SYSTEM_LOADED 76bf0000-76bfafff, C:\WINDOWS\system32\PSAPI.DLL
SYSTEM_LOADED 76f60000-76f8bfff, C:\WINDOWS\system32\WLDAP32.dll
SYSTEM_LOADED 76fd0000-7704efff, C:\WINDOWS\system32\CLBCATQ.DLL
SYSTEM_LOADED 77050000-77114fff, C:\WINDOWS\system32\COMRes.dll
SYSTEM_LOADED 77120000-771abfff, C:\WINDOWS\system32\OLEAUT32.dll
SYSTEM_LOADED 771b0000-7727efff, C:\WINDOWS\system32\WININET.dll
SYSTEM_LOADED 773d0000-774d2fff,
C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.
2600.2982_x-ww_ac3f9c03\comctl32.dll
SYSTEM_LOADED 774e0000-7761cfff, C:\WINDOWS\system32\ole32.dll
SYSTEM_LOADED 77690000-776b0fff, C:\WINDOWS\system32\NTMARTA.DLL
SYSTEM_LOADED 77920000-77a12fff, C:\WINDOWS\system32\SETUPAPI.dll
SYSTEM_LOADED 77b40000-77b61fff, C:\WINDOWS\system32\appHelp.dll
SYSTEM_LOADED 77c00000-77c07fff, C:\WINDOWS\system32\VERSION.dll
SYSTEM_LOADED 77c10000-77c67fff, C:\WINDOWS\system32\msvcrt.dll
SYSTEM_LOADED 77dd0000-77e6afff, C:\WINDOWS\system32\ADVAPI32.dll
SYSTEM_LOADED 77e70000-77f00fff, C:\WINDOWS\system32\RPCRT4.dll
SYSTEM_LOADED 77f10000-77f56fff, C:\WINDOWS\system32\GDI32.dll
SYSTEM_LOADED 77f60000-77fd5fff, C:\WINDOWS\system32\SHLWAPI.dll
SYSTEM_LOADED 77fe0000-77ff0fff, C:\WINDOWS\system32\Secur32.dll
MEMORY_MAPPED_ANON 78130000-781cafff,
C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.163_x-
ww_681e29fb\msvcr80.dll
MEMORY_MAPPED_ANON 7c420000-7c4a6fff,
C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.163_x-
ww_681e29fb\msvcp80.dll
MEMORY_MAPPED_ANON 7c4c0000-7c53cfff,
C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.163_x-ww_
681e29fb\msvcm80.dll
SYSTEM_LOADED 7c800000-7c8f3fff, C:\WINDOWS\system32\kernel32.dll
SYSTEM_LOADED 7c900000-7c9affff, C:\WINDOWS\system32\ntdll.dll
SYSTEM_LOADED 7c9c0000-7d1d4fff, C:\WINDOWS\system32\SHELL32.dll
SYSTEM_LOADED 7e410000-7e49ffff, C:\WINDOWS\system32\USER32.dll
SYSTEM_LOADED 7e830000-7eb9efff, C:\WINDOWS\system32\mshtml.dll
```

SYSTEM_LOADED—Indicates the DLL was loaded by Windows, the file must exist on the disk.

MEMORY_MAPPED_ANON—Indicates the DLL was loaded by ThinApp, the file might be loaded from the virtual file system.

46800000-46873fff—Indicates the address ranges in virtual memory where the DLL was loaded.

```
----Timing Report: list of slowest 150 objects profiled ---

8255572220 total cycles (2955.56 ms): |sprof| thinapp_LoadLibrary2
```

```
765380728 cycles (274.01 ms) on log entry 21753
428701805 cycles (153.48 ms) on log entry 191955
410404281 cycles (146.93 ms) on log entry 193969
231503734 cycles (82.88 ms) on log entry 188438
227419794 cycles (81.42 ms) on log entry 190209
211952538 cycles (75.88 ms) on log entry 197416
202095103 cycles (72.35 ms) on log entry 189394
200356604 cycles (71.73 ms) on log entry 194646
192420627 cycles (68.89 ms) on log entry 190812
183214731 cycles (65.59 ms) on log entry 195836
... 438 total calls
7847975891 total cycles (2809.64 ms): |sprof| ts_load_internal_module
764794646 cycles (273.80 ms) on log entry 21753
426837866 cycles (152.81 ms) on log entry 191955
408570540 cycles (146.27 ms) on log entry 193969
228790905 cycles (81.91 ms) on log entry 188438
224240114 cycles (80.28 ms) on log entry 190209
209789307 cycles (75.11 ms) on log entry 197416
200287437 cycles (71.70 ms) on log entry 189394
198429210 cycles (71.04 ms) on log entry 194646
190612618 cycles (68.24 ms) on log entry 190812
180322247 cycles (64.56 ms) on log entry 195836
... 94 total calls
4451728477 total cycles (1593.76 ms): |sprof| ts_lookup_imports
544327945 cycles (194.87 ms) on log entry 21758
385149968 cycles (137.89 ms) on log entry 193970
187246661 cycles (67.04 ms) on log entry 190210
173617241 cycles (62.16 ms) on log entry 194647
173481875 cycles (62.11 ms) on log entry 19065
148587579 cycles (53.20 ms) on log entry 195837
133165053 cycles (47.67 ms) on log entry 189395
126806624 cycles (45.40 ms) on log entry 197417
125894370 cycles (45.07 ms) on log entry 200296
121213253 cycles (43.40 ms) on log entry 200657
... 34 total calls
1099873523 total cycles (393.76 ms): |sprof| new_thread_start
561664565 cycles (201.08 ms) on log entry 151922
531551734 cycles (190.30 ms) on log entry 152733
1619002 cycles (0.58 ms) on log entry 72875
1554448 cycles (0.56 ms) on log entry 637896
1481627 cycles (0.53 ms) on log entry 72881
1091972 cycles (0.39 ms) on log entry 580771
```

Potential Errors

The potential errors list is a collection of all log entries that have *** in their string. ThinApp marks entries that could potentially be a problem by adding *** to the log entry output. See Locating Errors for more tips on interpreting this section.

```
----Potential Errors Detected ---

006425 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\Microsoft.Windows.Common-Controls.DLL' flags=2 -> 0 (failed ***)
006427 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\Microsoft.Windows.Common-Controls\Microsoft.Windows.Common-Control
s.DLL' flags=2
-> 0 (failed ***)
006428 0000089c nview.dll :1005b94b<-kernel32.dll:7c80ae4b *** LoadLibraryW -
>HMODULE=7c800000h () *** GetLastError() returns 2 [0]: The system cannot
find the file specified.
007062 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\en-US\Microsoft.Windows.Common-Controls.DLL' flags=2 -> 0 (failed
***)
010649 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\en-US\Microsoft.Windows.Common-Controls\Microsoft.Windows.Common-C
ontrols.DLL'
flags=2 -> 0 (failed ***)
019127 0000075c MSVCR80.dll :781348cc<-msvcrt.dll :77c10396 ***
GetEnvironmentVariableA -
>DWORD=0h (OUT LPSTR lpBuffer=*0h <bad ptr>) *** GetLastError() returns 203
[0]: The system
could not find the environment option that was entered.
019133 0000075c MSVCR80.dll :78133003<-nview.dll :1000058c *** GetProcAddress
-
>FARPROC=*0h () *** GetLastError() returns 127 [203]: The specified procedure
could not be found.
019435 0000075c MSVCR80.dll :78136e08<-dbghelp.dll :59a60360 *** Getfile type
->DWORD=0h ()
*** GetLastError() returns 6 [0]: The handle is invalid.
019500 0000075c MSVCR80.dll :78134481<-nview.dll :1000058c *** GetProcAddress
-
>FARPROC=*0h () *** GetLastError() returns 127 [0]: The specified procedure
could not be found.
019530 0000075c MSVCR80.dll :78131dcd<-dbghelp.dll :59a603a1 ***
GetModuleHandleA -
>HMODULE=0h () *** GetLastError() returns 126 [0]: The specified module could
not be found.
```

Troubleshooting Using Log Monitor

Log Monitor can be used to delve into application problems. This section uses an example in which `cmd.exe` is packaged with ThinApp and run with logging recorded.

To simulate an application behaving incorrectly, a simple invalid command is issued. In this case, we have requested `cmd.exe` to execute the command `foobar`, and `cmd.exe` prints the message "foobar is not recognized as an internal or external command." By viewing the resulting trace file you can dig into what `cmd.exe` is doing in much greater detail and learn how it operates. All applications manifest problems in different ways, so there is more than one method to track issues.

The first place to check in a log file is the section near the end labeled "----Potential Errors Detected ---."

In this section, you can find all the API functions in which the `GetLastError` code was modified. The paths highlighted in bold indicate locations that `cmd.exe` was looking for foobar, and paths in red indicate locations in the virtual file system that were probed for these file system probes.

```
----Potential Errors Detected ---
*** Unable to determine if any services need to be auto-started, error 2
001550 *** FindFirstFileW 'C:\test\cmd_test\bin\foobar.*' ->
INVALID_HANDLE_VALUE *** failed
[system probe C:\test\cmd_test\bin\foobar.* -> ffffffffh][no virtual or
system matches]
*** FindFirstFileW ->HANDLE=ffffffffh .. *** GetLastError() returns 2 [203]:
The system cannot
find the file specified.
*** FindFirstFileW 'C:\test\cmd_test\bin\foobar' -> INVALID_HANDLE_VALUE ***
failed [FS
missing in view 0][fs entry not found %drive_C%\test\cmd_test\bin\foobar][fs
entry not found
%drive_C%\test\cmd_test\bin]
*** FindFirstFileW 'C:\WINDOWS\system32\foobar.*' -> INVALID_HANDLE_VALUE ***
failed [system
probe C:\WINDOWS\system32\foobar.* -> ffffffffh][no virtual or system
matches]
*** FindFirstFileW 'C:\WINDOWS\system32\foobar' -> INVALID_HANDLE_VALUE ***
failed [FS missing
in view 0][fs entry not found %SystemSystem%\foobar]
*** FindFirstFileW 'C:\WINDOWS\foobar.*' -> INVALID_HANDLE_VALUE *** failed
[system probe
C:\WINDOWS\foobar.* -> ffffffffh][no virtual or system matches]
*** FindFirstFileW 'C:\WINDOWS\foobar' -> INVALID_HANDLE_VALUE *** failed [FS
missing in view
0][fs entry not found %SystemRoot%\foobar]
*** FindFirstFileW 'C:\WINDOWS\System32\Wbem\foobar.*' ->
INVALID_HANDLE_VALUE *** failed
```

```
[system probe C:\WINDOWS\System32\Wbem\foobar.* -> ffffffffh][no virtual or
system matches]
*** FindFirstFileW 'C:\WINDOWS\System32\Wbem\foobar' -> INVALID_HANDLE_VALUE
*** failed [FS
missing in view 0][fs entry not found %SystemSystem%\Wbem\foobar]
*** FindFirstFileW 'c:\Program Files\subversion\bin\foobar.*' ->
INVALID_HANDLE_VALUE ***
failed [system probe c:\Program Files\subversion\bin\foobar.* ->
ffffffffh][no virtual or
system matches]
*** FindFirstFileW 'c:\Program Files\subversion\bin\foobar' ->
INVALID_HANDLE_VALUE *** failed
[FS missing in view 0][fs entry not found
%ProgramFilesDir%\subversion\bin\foobar][fs entry
not found %ProgramFilesDir%\subversion\bin]
*** FindFirstFileW 'c:\Program Files\Microsoft SQL
Server\90\Tools\binn\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe c:\Program Files\Microsoft SQL
Server\90\Tools\binn\foobar.* -> ffffffffh][no virtual or system matches]
*** FindFirstFileW 'c:\Program Files\Microsoft SQL
Server\90\Tools\binn\foobar' ->
INVALID_HANDLE_VALUE *** failed [FS missing in view 0][fs entry not found
%ProgramFilesDir%\Microsoft SQL Server\90\Tools\binn\foobar][fs entry not
found
%ProgramFilesDir%\Microsoft SQL Server\90\Tools\binn]
*** FindFirstFileW 'c:\bin\foobar.*' -> INVALID_HANDLE_VALUE *** failed
[system probe
c:\bin\foobar.* -> ffffffffh][no virtual or system matches]
*** FindFirstFileW 'c:\bin\foobar' -> INVALID_HANDLE_VALUE *** failed [FS
missing in view
0][fs entry not found %drive_c%\bin\foobar][fs entry not found %drive_c%\bin]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\Tools\WinNT\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe C:\Program Files\Microsoft
Visual
Studio\Common\Tools\WinNT\foobar.* -> ffffffffh][no virtual or system
matches]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\Tools\WinNT\foobar' ->
INVALID_HANDLE_VALUE *** failed [FS missing in view 0][fs entry not found
%ProgramFilesDir%\Microsoft Visual Studio\Common\Tools\WinNT\foobar][fs entry
not found
%ProgramFilesDir%\Microsoft Visual Studio\Common\Tools\WinNT]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\MSDev98\Bin\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe C:\Program Files\Microsoft
Visual
Studio\Common\MSDev98\Bin\foobar.* -> ffffffffh][no virtual or system
matches]
```

```
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\MSDev98\Bin\foobar' ->
INVALID_HANDLE_VALUE *** failed [FS missing in view 0][fs entry not found
%ProgramFilesDir%\Microsoft Visual Studio\Common\MSDev98\Bin\foobar][fs entry
not found
%ProgramFilesDir%\Microsoft Visual Studio\Common\MSDev98\Bin]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\Tools\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe C:\Program Files\Microsoft
Visual
Studio\Common\Tools\foobar.* -> ffffffffh][no virtual or system matches]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\Tools\foobar' ->
INVALID_HANDLE_VALUE *** failed [FS missing in view 0][fs entry not found
%ProgramFilesDir%\Microsoft Visual Studio\Common\Tools\foobar][fs entry not
found
%ProgramFilesDir%\Microsoft Visual Studio\Common\Tools]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\VC98\bin\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe C:\Program Files\Microsoft
Visual
Studio\VC98\bin\foobar.* -> ffffffffh][no virtual or system matches]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual Studio\VC98\bin\foobar'
->
INVALID_HANDLE_VALUE *** failed [FS missing in view 0][fs entry not found
%ProgramFilesDir%\Microsoft Visual Studio\VC98\bin\foobar][fs entry not found
%ProgramFilesDir%\Microsoft Visual Studio\VC98\bin]
```

As you can see, the potential errors does a good job of highlighting possible areas where the application is failing.

Deeper Examination

```
001550 *** FindFirstFileW 'C:\test\cmd_test\bin\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe
```

To determine why `cmd.exe` is probing the location `c:\test\cmd_test\bin`, search the log for this line of text using the *log entry number* and find out what is occurring before this call. In the **bold** excerpts below, two possible places are seen where `cmd.exe` obtained the path `c:\test\cmd_test`.

The first is obtained by calling `GetCurrentDirectoryW`, and the second is from calling `GetFullPathNameW` with "." as the path specifies. In both cases, this returns the path for the current working directory, making it clear exactly how `cmd.exe` is obtaining this path.

You can see in the log file how `cmd.exe` creates the `c:\test\cmd_test\bin>` prompt. This is accomplished by querying the environment variable "PROMPT" that returns "PG" and uses the API function WriteConsoleW to print the prompt to the screen after internally expanding "PG" to `c:\test\cmd_test\bin>`.

```
000824 0a88 cmd.exe :4ad0697a<-ADVAPI32.dll:77dd038f FormatMessageW
->DWORD=29h
(OUT LPWSTR lpBuffer=*4AD38BA0h->L"(C) Copyright 1985-2001 Microsoft
Corp.\0Dh\0Ah")
000825 0a88 cmd.exe :4ad069d1->ADVAPI32.dll:77dd038f FormatMessageW (IN DWORD
dwFlags=1800h, IN LPCVOID lpSource=*0h, IN DWORD dwMessageId=2334h, IN DWORD
dwLanguageId=0h, IN DWORD nSize=2000h, IN *Arguments=*13DD40h->...)
000826 0a88                        FormatMessageW
FORMAT_MESSAGE_FROM_HMODULE FORMAT_MESSAGE_FROM_SYSTEM
line_width=unlimited lpSource=0x0, dwMessageId=0x2334, dwLanguageId=0x0
                                   -> 0x29 ((C) Copyright 1985-2001 Microsoft
                  Corp.
 )
000827 0a88 cmd.exe  :4ad069d1<-ADVAPI32.dll:77dd038f FormatMessageW
->DWORD=29h
(OUT LPWSTR lpBuffer=*4AD38BA0h->L"(C) Copyright 1985-2001 Microsoft
Corp.\0Dh\0Ah")
000828 0a88 cmd.exe  :4ad08d01->kernel32.dll:7c835484 WriteConsoleW (IN
HANDLE
hConsoleOutput=7h, IN const *lpBuffer=*4AD38BA0h, IN DWORD
nNumberOfCharsToWrite=29h, IN
LPVOID lpReserved=*0h)
000829 0a88 cmd.exe  :4ad08d01<-kernel32.dll:7c835484 WriteConsoleW ->BOOL=1h
(OUT
LPDWORD lpNumberOfCharsWritten=*13DD24h->29h)
000830 0a88 cmd.exe  :4ad048f4->msctfime.ime:755c039b GetModuleHandleW (IN
LPCWSTR
lpModuleName=*4AD0498Ch->L"KERNEL32.DLL")
000831                             0a88 GetModuleHandleW 'KERNEL32.DLL' ->
7c800000
000832 0a88 cmd.exe  :4ad048f4<-msctfime.ime:755c039b GetModuleHandleW -
->HMODULE=7c800000h ()
000833 0a88 cmd.exe  :4ad04907->AcGenral.DLL:6f880364 GetProcAddress (IN
HMODULE hModule=7c800000h, IN LPCSTR lpProcName=*4AD04980h->"CopyFileExW")
000834 0a88                        GetProcAddress
mod=7c800000/C:\WINDOWS\system32\kernel32.dll () 'CopyFileExW' -> 7feb1fcf
000835 0a88 cmd.exe  :4ad04907<-AcGenral.DLL:6f880364 GetProcAddress -
->FARPROC=*7FEB1FCFh ()
000836 0a88 cmd.exe  :4ad04919->AcGenral.DLL:6f880364 GetProcAddress (IN
HMODULE
hModule=7c800000h, IN LPCSTR lpProcName=*4AD0496Ch->"IsDebuggerPresent")
000837 0a88                        GetProcAddress
mod=7c800000/C:\WINDOWS\system32\kernel32.dll () 'IsDebuggerPresent' ->
7fec0dfa
```

```
000838 0a88 cmd.exe :4ad04919<-AcGenral.DLL:6f880364 GetProcAddress -
->FARPROC=*7FEC0DFAh ()
000839 0a88 cmd.exe  :4ad0492b->AcGenral.DLL:6f880364 GetProcAddress (IN
HMODULE
hModule=7c800000h, IN LPCSTR
lpProcName=*4AD04954h->"SetConsoleInputExeNameW")
000840 0a88                      GetProcAddress
mod=7c800000/C:\WINDOWS\system32\kernel32.dll () 'SetConsoleInputExeNameW' ->
7fe90c21
000841 0a88 cmd.exe  :4ad0492b<-AcGenral.DLL:6f880364 GetProcAddress -
>FARPROC=*7FE90C21h ()
000842 0a88 cmd.exe  :4ad02c97->ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=3h)
000843 0a88                    Getfile type 3 -> 0x2
000844 0a88 cmd.exe  :4ad02c97<-ole32.dll :774e03f0 Getfile type ->DWORD=2h
()
000845 0a88 cmd.exe  :4ad02cc0->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF6h)
000846 0a88 cmd.exe  :4ad02cc0<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=3h ()
000847 0a88 cmd.exe  :4ad02ccd->kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=3h)
000848 0a88 cmd.exe  :4ad02ccd<-kernel32.dll:7c81af14 GetConsoleMode
->BOOL=1h (OUT
LPDWORD lpMode=*13DDCCh->A7h)
000849 0a88 cmd.exe  :4ad05b74->ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=7h)
000850 0a88                    Getfile type 7 -> 0x2
000851 0a88 cmd.exe  :4ad05b74<-ole32.dll :774e03f0 Getfile type ->DWORD=2h
()
000852 0a88 cmd.exe  :4ad05b9d->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF5h)
000853 0a88 cmd.exe  :4ad05b9d<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=7h ()
000854 0a88 cmd.exe  :4ad05baa->kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=7h)
000855 0a88 cmd.exe  :4ad05baa<-kernel32.dll:7c81af14 GetConsoleMode
->BOOL=1h (OUT
LPDWORD lpMode=*13DA80h->3h)
000856 0a88 cmd.exe  :4ad05dec->kernel32.dll:7c835484 WriteConsoleW (IN
HANDLE
hConsoleOutput=7h, IN const *lpBuffer=*4AD38BA0h, IN DWORD
nNumberOfCharsToWrite=2h, IN
LPVOID lpReserved=*0h)
000857 0a88 cmd.exe  :4ad05dec<-kernel32.dll:7c835484 WriteConsoleW ->BOOL=1h
(OUT
LPDWORD lpNumberOfCharsWritten=*13DAACh->2h)
```

```
000858 0a88 cmd.exe  :4ad01ba8->USERENV.dll :769c03b9 GetEnvironmentVariableW
(IN
LPCWSTR lpName=*4AD34624h->L"PROMPT," IN DWORD nSize=2000h)
000859 0a88                    GetEnvironmentVariable PROMPT -> $P$G
000860 0a88 cmd.exe  :4ad01ba8<-USERENV.dll :769c03b9 GetEnvironmentVariableW
-
>DWORD=4h (OUT LPWSTR lpBuffer=*4AD2BA20h->L"$P$G")
000861 0a88 cmd.exe  :4ad01580->USERENV.dll :769c0396 GetCurrentDirectoryW
(IN DWORD
nBufferLength=104h)
000862 0a88                    GetCurrentDirectoryW -> 0x14
(C:\test\cmd_test\bin)
000863 0a88 cmd.exe  :4ad01580<-USERENV.dll :769c0396 GetCurrentDirectoryW
->DWORD=14h
(OUT LPWSTR lpBuffer=*4AD34400h->L"C:\test\cmd_test\bin")
000864 0a88 cmd.exe  :4ad05b74->ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=7h)
000865 0a88                    Getfile type 7 -> 0x2
000866 0a88 cmd.exe  :4ad05b74<-ole32.dll :774e03f0 Getfile type ->DWORD=2h
()
000867 0a88 cmd.exe  :4ad05b9d->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF5h)
000868 0a88 cmd.exe  :4ad05b9d<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=7h ()
000869 0a88 cmd.exe  :4ad05baa->kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=7h)
000870 0a88 cmd.exe  :4ad05baa<-kernel32.dll:7c81af14 GetConsoleMode
->BOOL=1h (OUT
LPDWORD lpMode=*13DA84h->3h)
000871 0a88 cmd.exe  :4ad05dec->kernel32.dll:7c835484 WriteConsoleW (IN
HANDLE
hConsoleOutput=7h, IN const *lpBuffer=*4AD2B1E0h, IN DWORD
nNumberOfCharsToWrite=15h, IN
LPVOID lpReserved=*0h)
000872 0a88 cmd.exe  :4ad05dec<-kernel32.dll:7c835484 WriteConsoleW ->BOOL=1h
(OUT
LPDWORD lpNumberOfCharsWritten=*13DAB0h->15h)
000873 0a88 cmd.exe  :4ad0bf00->ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=3h)
000874 0a88                    Getfile type 3 -> 0x2
000875 0a88 cmd.exe  :4ad0bf00<-ole32.dll :774e03f0 Getfile type ->DWORD=2h
()
000876 0a88 cmd.exe  :4ad02c97->ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=3h)
000877 0a88                    Getfile type 3 -> 0x2
000878 0a88 cmd.exe  :4ad02c97<-ole32.dll :774e03f0 Getfile type ->DWORD=2h
()
```

```
000879 0a88 cmd.exe   :4ad02cc0->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF6h)
000880 0a88 cmd.exe   :4ad02cc0<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=3h ()
000881 0a88 cmd.exe   :4ad02ccd->kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=3h)
000882 0a88 cmd.exe   :4ad02ccd<-kernel32.dll:7c81af14 GetConsoleMode
->BOOL=1h (OUT
LPDWORD lpMode=*13DD50h->A7h)
000883 0a88 cmd.exe   :4ad02c97->ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=3h)
000884 0a88                    Getfile type 3 -> 0x2
000885 0a88 cmd.exe   :4ad02c97<-ole32.dll :774e03f0 Getfile type ->DWORD=2h
()
000886 0a88 cmd.exe   :4ad02cc0->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF6h)
000887 0a88 cmd.exe   :4ad02cc0<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=3h ()
000888 0a88 cmd.exe   :4ad02ccd->kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=3h)
000889 0a88 cmd.exe   :4ad02ccd<-kernel32.dll:7c81af14 GetConsoleMode
->BOOL=1h (OUT
LPDWORD lpMode=*13DD50h->A7h)
000890 0a88 cmd.exe   :4ad0b9d4->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF5h)
000891 0a88 cmd.exe   :4ad0b9d4<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=7h ()
000892 0a88 cmd.exe   :4ad0ba16->kernel32.dll:7c81bc2b
GetConsoleScreenBufferInfo (IN
HANDLE hConsoleOutput=7h)
000893 0a88 cmd.exe   :4ad0ba16<-kernel32.dll:7c81bc2b
GetConsoleScreenBufferInfo ->BOOL=1h
(OUT PCONSOLE_SCREEN_BUFFER_INFO lpConsoleScreenBufferInfo=*13DD08h->struct
{COORD dwSize=struct {SHORT X=50h, SHORT Y=12Ch}, COORD
dwCursorPosition=struct
{SHORT X=15h, SHORT Y=5h}, WORD wAttributes=7h, SMALL_RECT srWindow=struct
{SHORT
Left=0h, SHORT Top=0h, SHORT Right=4Fh, SHORT Bottom=18h}, COORD
dwMaximumWindowSize=struct {SHORT X=50h, SHORT Y=53h}})
000894 0a88 cmd.exe  :4ad0ba71->kernel32.dll:7c871a6c ReadConsoleW (IN HANDLE
hConsoleInput=3h, IN DWORD nNumberOfCharsToRead=2000h, IN LPVOID
lpReserved=*13DD20h)
001518 0a88 cmd.exe  :4ad0ba71<-kernel32.dll:7c871a6c ReadConsoleW ->BOOL=1h
(OUT
LPVOID lpBuffer=*4AD2FAE0h, OUT LPDWORD lpNumberOfCharsRead=*13DD70h->8h)
001519 0a88 cmd.exe  :4ad02c97->ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=3h)
001520 0a88                    Getfile type 3 -> 0x2
```

```
001521 0a88 cmd.exe  :4ad02c97<–ole32.dll :774e03f0 Getfile type –>DWORD=2h
()
001522 0a88 cmd.exe  :4ad02cc0–>kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF6h)
001523 0a88 cmd.exe  :4ad02cc0<–kernel32.dll:7c812f39 GetStdHandle
–>HANDLE=3h ()
001524 0a88 cmd.exe  :4ad02ccd–>kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=3h)
001525 0a88 cmd.exe  :4ad02ccd<–kernel32.dll:7c81af14 GetConsoleMode
–>BOOL=1h (OUT
LPDWORD lpMode=*13DD50h–>A7h)
001526 0a88 cmd.exe  :4ad0bb9c–>kernel32.dll:7c81b18f GetConsoleOutputCP ()
001527 0a88 cmd.exe  :4ad0bb9c<–kernel32.dll:7c81b18f GetConsoleOutputCP
–>UINT=1B5h ()
001528 0a88 cmd.exe  :4ad0bbad–>kernel32.dll:7c812e76 GetCPInfo (IN UINT
CodePage=1B5h)
001529 0a88 cmd.exe  :4ad0bbad<–kernel32.dll:7c812e76 GetCPInfo –>BOOL=1h
(OUT LPCPINFO
lpCPInfo=*4AD33BA0h–>struct {UINT MaxCharSize=1h, char[2] DefaultChar=['?',
'\00h'], char[12]
LeadByte=['\00h', '\00h', '\00h', '\00h', '\00h', '\00h', '\00h', '\00h',
'\00h', '\00h', '\00h', '\00h']})
001530 0a88 cmd.exe  :4ad01680–>kernel32.dll:7c81b258 SetThreadUILanguage (==
no prototype
available ==)
001531 0a88 cmd.exe  :4ad01680<–kernel32.dll:7c81b258 SetThreadUILanguage (==
no prototype
available ==)
001532 0a88 cmd.exe  :4ad01b0d–>kernel32.dll:7c80ac0f SetErrorMode (IN UINT
uMode=0h)
001533 0a88 cmd.exe  :4ad01b0d<–kernel32.dll:7c80ac0f SetErrorMode –>UINT=0h
()
001534 0a88 cmd.exe  :4ad01b13–>kernel32.dll:7c80ac0f SetErrorMode (IN UINT
uMode=1h)
001535 0a88 cmd.exe   :4ad01b13<–kernel32.dll:7c80ac0f SetErrorMode –>UINT=0h
()
001536 0a88 cmd.exe  :4ad01b24–>IMM32.DLL :7639039b GetFullPathNameW (IN
LPCWSTR
lpFileName=*1638C0h–>L"," IN DWORD nBufferLength=208h)
001537 0a88                  GetFullPathNameW . –> 20
(buf=C:\test\cmd_test\bin,
file_part=bin)
001538 0a88 cmd.exe  :4ad01b24<–IMM32.DLL :7639039b GetFullPathNameW
–>DWORD=14h
(OUT LPWSTR lpBuffer=*163D60h–>L"C:\test\cmd_test\bin," OUT
*lpFilePart=*13D8D4h–
>*163D82h–>L"bin")
001539 0a88 cmd.exe   :4ad01b29–>kernel32.dll:7c80ac0f SetErrorMode (IN UINT
uMode=0h)
```

```
001540 0a88 cmd.exe   :4ad01b29<–kernel32.dll:7c80ac0f SetErrorMode –>UINT=1h
()
001541 0a88 cmd.exe   :4ad01ba8–>USERENV.dll :769c03b9
GetEnvironmentVariableW (IN
LPCWSTR lpName=*4AD34618h–>L"PATH," IN DWORD nSize=2000h)
001542 0a88                    GetEnvironmentVariable PATH –>
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;c:\program
files\subversion\bin;c:\Program Files\Microsoft SQL
Server\90\Tools\binn\;c:\bin;C:\Program
Files\Microsoft Visual Studio\Common\Tools\WinNT;C:\Program Files\Microsoft
Visual
Studio\Common\MSDev98\Bin;C:\Program Files\Microsoft Visual
Studio\Common\Tools;C:\Program
Files\Microsoft Visual Studio\VC98\bin
001543 0a88 cmd.exe   :4ad01ba8<–USERENV.dll :769c03b9
GetEnvironmentVariableW –
>DWORD=173h (OUT LPWSTR lpBuffer=*4AD2BA20h–
–>L"C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;c:\Program
Files\su.. ")
001544 0a88 cmd.exe   :4ad01ba8–>USERENV.dll :769c03b9
GetEnvironmentVariableW (IN
LPCWSTR lpName=*4AD34608h–>L"PATHEXT," IN DWORD nSize=2000h)
001545 0a88                    GetEnvironmentVariable PATHEXT –>
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
001546 0a88 cmd.exe   :4ad01ba8<–USERENV.dll :769c03b9
GetEnvironmentVariableW –
–>DWORD=30h (OUT LPWSTR lpBuffer=*4AD2BA20h–
–> L."COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH")
001547 0a88 cmd.exe   :4ad02aaa–>kernel32.dll:7c80b2d0 GetDriveTypeW (IN
LPCWSTR lpRootPathName=*13D8C4h–>L"C:\")
001548 0a88 cmd.exe   :4ad02aaa<–kernel32.dll:7c80b2d0 GetDriveTypeW
–>UINT=3h ()
001549 0a88 cmd.exe   :4ad01b5f–>USERENV.dll :769c03fa FindFirstFileW (IN
LPCWSTR lpFileName=*1638C0h–>L"C:\test\cmd_test\bin\foobar.*")
001550 0a88                   FindFirstFileW 'C:\test\cmd_test\bin\foobar.*'
–>
INVALID_HANDLE_VALUE *** failed [system probe C:\test\cmd_test\bin\foobar.*
–> ffffffffh][no virtual or system matches]
```

Using Sandbox Merge (sbmerge)

Sandbox Merge (`sbmerge.exe`) is used to merge runtime changes recorded in the application sandbox back into a ThinApp project.

A typical workflow for using sbmerge looks like this:

1 Use Setup Capture to create a ThinApp project.

2 Build a captured application using `build.bat`.

3 Run a captured application, customize your user settings and virtual environment. The changes go to the sandbox.

4 Run sbmerge to merge registry and file system changes from your sandbox into your ThinApp project.

5 Build your updated captured application using `build.bat`.

6 Deploy your updated, captured application (including customizations from Step 3).

Usage

```
sbmerge.exe Print [OptionalParameters]
sbmerge.exe Apply [Optional Parameters]
```

Optional Parameters

```
[–ProjectDir PathToProject] [–SandboxDir PathToSandbox]
[–Quiet] [–Exclude ExcludeFile.ini]
```

Main Operation

`Print` displays sandbox changes. This is a non-destructive operation, no modifications are made to the sandbox or original project.

`Apply` merges changes from the sandbox into the specified project. The project registry and file system are updated to reflect changes from the sandbox. The sandbox directory is deleted when this operation completes.

Optional Parameters

The following are optional parameters:

- `ProjectDir` specifies the project directory. The default is the current directory.
- `SandboxDir` specifies the sandbox location. The default is the active sandbox.
- `Quiet` does not print progess messages.
- The `Exclude` file defaults to the `snapshot.ini` file.

Example Usage

The following prints changes that are recorded in the sandbox:

```
c:\...\ThinAppdir> sbmerge Print –ProjectDir c:\myproject
```

The following merges the changes from the sandbox into the project directory located at c:\myproject sbmerge uses the normal search order for sandbox probing:

```
c:\...\ThinAppdir> sbmerge Apply –ProjectDir c:\myproject
c:\somedir> sbmerge Apply –ProjectDir c:\myproject –SandboxDir c:\documents
and settings\username\application data\thinstall\myproject
```

If you are running from the project directory, you must supply the complete path to sbmerge.exe. For example, `c:\Program Files\VMware\VMware ThinApp\sbmerge.exe`.

Using ThinReg

ThinReg (`ThinReg.exe`) is used to register ThinApp packages on a PC, including:

- Creating Start Menu and desktop shortcuts
- Setting up file type associations
- Adding uninstall information that can be executed from the system control panel
- Unregistering previously registered packages

Usage

```
ThinReg.exe [Optional Parameters] PackageName.exe [Package2.exe]
[PackagesByWildcard]
```

Optional Parameters include:

`/a, /allusers`

Register a package for all users. This will register the package for all users but if an unauthorized user attempts to run the application, a message will appear that tells the user he or she cannot run the application.

`/q, /quiet`

Do not output any information.

`/u, /unregister, /uninstall`

Unregister a package. If needed, this also removes the software from the `Add/Remove Programs` control panel applet.

`/r, /reregister`

Re-register a package. Normally, ThinReg detects if a package is already registered and skips it if it is. By using the /r option, you can force ThinReg to re-register the package.

`/k, /keepunauthorized, /keep`

If ThinReg detects a registration for a package for that you're no longer authorized (you are no longer part of the PermittedGroups specified in the packages `Package.ini`) it removes the registration information for that package. By specifying this option, ThinReg won't remove the registration info.

`/noarp`

Do not create an entry in the Add/Remove Programs control panel applet

`/norelaunch`

This option has an effect on Microsoft Vista only. ThinReg is launched on Vista without elevated privileges. That means standard users are able to invoke ThinReg without a UAC pop-up window. When ThinReg detects that it does need more privileges (those privileges are required for the /allusers option) and relaunches itself as a privileged process, causing a UAC pop-up window to show. The `/norelaunch` option prevents the relaunch and instead the registration fails.

The standard usage of ThinReg is to register a number of packages stored in a directory (probably on a file server) during login processing: `ThinReg \\server\share\dir\*.exe`. This automatically takes care of `PermittedGroups` specifications, registering and unregistering packages as needed.

When you register a package for the current user (no /allusers switch is present), only shortcuts and filetype associations for that the current user is authorized for in the `PermittedGroups` setting will be created. If there is no `PermittedGroups` setting, the current user is authorized for all executables.

When registering a package for all users (the /allusers switch is included), all shortcuts and filetype associations are created, regardless of the `PermittedGroups` setting. When you double-click on a shortcut that you are not authorized for, you will still get an "Access Denied" error message at runtime.

Example Usage

The following registers Word for the currently logged in user:

```
ThinReg.exe "\\server\share\Microsoft Office 2003 Winword.exe"
```

The following registers Word for all users on the system (requires admin rights):

```
ThinReg.exe /a "\\server\share\Microsoft Office 2003 Winword.exe"
```

The following registers all Office applications in the specified directory for the currently logged in user:

```
ThinReg.exe "\\server\share\Microsoft Office *.exe"
```

The following unregisters just Word for the current user:

```
ThinReg.exe /u "\\server\share\Microsoft Office 2003 Winword.exe"
```

The following unregisters all Office applications for the current user and removes the Add/Remove Programs entry:

```
ThinReg.exe /u "\\server\share\Microsoft Office *.exe"
```

NOTE Command-line unregistration is supported in ThinApp 3.210 and higher.

Using Snapshot

Typically you do not need to use `snapshot.exe` directly, it is invoked by Setup Capture. This documentation is for advanced users and system integrators who are building ThinApp functionality into other platforms.

Command Line Usage

All parameters are case insensitive.

```
snapshot CaptureFile.snapshot
[BaseDirectory1][BaseDirectory2][BaseRegistry1][BaseRegistry2][–Config
ConfigFile.ini]
snapshot Original.snapshot –Diff NewEnvironment.snapshot OutputDirectory
[–Config
ConfigFile.ini][–Quiet]
snapshot Original.snapshot –DiffPrint NewEnvironment.snapshot [–Config
ConfigFile.ini]
```

Snapshot can be used for two purposes, saving a snapshot and creating a ThinApp project from two previously captured snapshots.

Saving a Snapshot

Snapshot can capture a snapshot of a computer's file system and registry, that is used later to create a ThinApp project. If there are no additional parameters, Snapshot scans and saves a copy of the following data:

- All file information for all local drives (directories, filenames, file attributes, file sizes, and file modification dates)
- A complete copy of the registry trees `HKEY_LOCAL_MACHINE` and `HKEY_USERS` (because HKCR and HKCU are subsets of HKLM and HKU, there is no need to scan them)

Example Usage

```
snapshot c:\Capture.snapshot
```

Captures a complete snapshot of local drives and registry to the file `c:\Capture.snapshot.`

```
snapshot c:\Capture.snapshot c:\ e:\
```

Captures a complete snapshot of the drives `c:\` and `e:\`, no registry information is captured.

```
snapshot c:\Capture.snapshot c:\ HKEY_LOCAL_MACHINE\Software\Classes
```

Captures a complete snapshot of the drive `c:\` and all of the `HKEY_CLASSES_ROOT` registry subtree.

Snapshot can be used to generate a ThinApp project directory by comparing two snapshots.

Example Usage

```
snapshot c:\Original.snapshot –Diff c:\NewEnvironment.snapshot c:\MyProject
```

This compares the two specified projects, and generates a resulting ThinApp Project in the directory `c:\MyProject.`

Displaying Differences Between Two Previously Captured Snapshots

```
snapshot Original.snapshot –DiffPrint NewEnvironment.snapshot
```

`DiffPrint` is similar to `–Diff` except that is does not have an output directory and all changes detected are printed to the console.

Configuration Files

The configuration file allows you to specify directories and subkeys to be excluded from a scan of created projects.

If no configuration file is specified Snapshot attempts to load its config file from these locations:

1 `Application Data\Thinapp\Snapshot.ini` (user's `AppData` directory)

2 `C:\Program Files\Thinapp\Snapshot.ini` (location from which `Snapshot.exe` is run)

See the default `Snapshot.ini` in the `C:\Program Files\Thinapp\Snapshot.ini` for more details about customizations that can be performed.

`Snapshot.exe` can be used for:

- Saving the state of a PC's file system and registry

  For example:

  ```
  snapshot C:\data.snapshot snapshot
  C:\data.snapshot C:\ HKEY_LOCAL_MACHINE
  ```

- Comparing two previously recorded states

  For example:

  ```
  snapshot C:\start.snapshot –diffprint C:\end.snapshot
  ```

- Printing contents of a saved state

  For example:

  ```
  snapshot C:\start.snapshot –print
  ```

- Generating a ThinApp project by comparing two previously saved states

  For example:

  ```
  snapshot C:\start.snapshot –SuggestProject C:\end.snapshot C:\project.ini
  snapshot C:\project.ini –GenerateProject
  ```

Usage

```
snapshot CaptureFile.snapshot
[BaseDirectory1][BaseDirectory2][BaseRegistry1][BaseRegistry2][–Config
ConfigFile.ini]
snapshot Original.snapshot –Diff NewEnvironment.snapshot OutputDirectory
[–Config
ConfigFile.ini][–Quiet]
snapshot Original.snapshot –DiffPrint NewEnvironment.snapshot [–Config
ConfigFile.ini]
```

The following is an example `.bat` file to generate a ThinApp project using a start and ending snapshot:

```
snapshot c:\start.snapshot
echo Please install application and press ENTER when installation is complete
pause
snapshot C:\end.snapshot s
napshot C:\start.snapshot –SuggestProject c:\end.snapshot
c:\SuggestedProject.ini
```

At this point a GUI application can look at `SuggestedProject` and ask the user which executables they want to be accessible from `Package.ini`.

```
snapshot C:\SuggestedProject.ini –GenerateProject c:\ProjectLocation
del C:\start.snapshot
del C:\end.snapshot
del C:\SuggestedProject.ini
```

Using dll_dump

Use dll_dump to list all captured applications that are currently running on a PC. Dll_dump lists both the processes that are running and all DLLs loaded within that process, whether the DLL is loaded virtually or by the system.

Dll_dump can be found in the following directory:

```
C:\Program Files\Thinapp.VS>dll_dump.exe
```

Usage

`dll_dump ADDRESS` (shows the DLL and process that has this address loaded)

`dll_dump SUBSTRING` (shows DLLs loaded by ThinApp processes where the name matches the `SUBSTRING`)

`dll_dump *` (shows DLLs loaded by all ThinApp processes)

`dll_dump –fp` (shows DLL full path for all ThinApp processes running, not just filenames)

`dll_dump ADDRESS SUBSTRING` (only shows processes matching `SUBSTRING` where `ADDRESS` is loaded).

One of the most useful purposes for `dll_dump` is to list all running captured applications on a PC.

If you use a spy program like Process Explorer on a captured application, you do not see DLLs that are loaded by ThinApp because they are virtualized and Windows does not recognize that they exist. Likewise, if you attach a debugger to a running ThinApp process, the debugger is not aware of virtual DLLs. If you are investigating code running at a specific address, you can use dll_dump to convert this address into a virtual DLL name and base address.

Using log monitor, you can generate a trace and convert this to text format.

This section lists all DLLs that were loaded by the application over the course of its execution history.

DLLs that are described as SYSTEM_LOADED are loaded by Windows from the host PC, including all the basic OS DLLs like `kernel32.dll`.

DLLs that are described as MEMORY_MAPPED_ANON are loaded by ThinApp and completely isolated from Windows.

For Adobe Reader, you should see something like this:

```
PRELOADED_BY_SYSTEM 00400000-00410000, C:\Program Files\Adobe\Acrobat
7.0\Reader\AcroRd32.exe PRELOADED_MAP 00400000-00410000, C:\Program
Files\Adobe\Acrobat 7.0\Reader\AcroRd32.exe SYSTEM_LOADED 00400000-00410000,
C:\thinstest\1072\Adobe Reader 7.0\bin\Adobe.exe
SYSTEM_LOADED 77dd0000-77e6b000, C:\WINDOWS\system32\ADVAPI32.dll
SYSTEM_LOADED 76fd0000-7704f000, C:\WINDOWS\system32\CLBCATQ.DLL
SYSTEM_LOADED 5d090000-5d127000, C:\WINDOWS\system32\comctl32.dll ...

MEMORY_MAPPED_ANON 05000000-05085000, C:\Program Files\Adobe\Acrobat
7.0\Reader\ACE.dll

MEMORY_MAPPED_ANON 03000000-038c9000, C:\Program Files\Adobe\Acrobat
7.0\Reader\AcroRd32.dll MEMORY_MAPPED_ANON 06000000-061aa000, C:\Program
Files\Adobe\Acrobat 7.0\Reader\AGM.dll MEMORY_MAPPED_ANON 07000000-0701b000,
C:\Program Files\Adobe\Acrobat 7.0\Reader\BIB.dll MEMORY_MAPPED_ANON
7c3a0000-7c41b000, C:\WINDOWS\system32\MSVCP71.dll
```

Using ConfigDump

The ConfigDump.dll extracts configuration information from the host machine into the support folder of a captured application.

ConfigDump.dlll can be found in the following directory:

```
C:\Program Files\VMware\ThinApp\Captures\<package name>\Support\Capture
Machine Overview.txt
```

The following are explanations of the text in the Capture Machine Overview.txt file:

```
Dump started on 5/6/2008 at 18:56:13.498
Dump ended on 56/2008 at 18:56:15.139
```

The date and time that the ConfigDump was run. The time is normally about one second.

```
Operating system:    Windows Server 2003 Service Pack 2
```

The platform operating system under which the captured application is running. If it is available, the service pack data is also shown.

```
User name:        <name>
```

This is the login name of the current user.

```
Member of: None, Everyone, Debugger Users, Administrators, Remote Desktop
Users, Users, REMOTE INTERACTIVE LOGON, INTERACTIVE, Authenticated Users,
This Organization, <name lookup failed>, LOCAL, NTLM Authentication
```

These are the various user groups to which the user belongs. Note the `<name lookup failed>` entry above, which indicates that a problem occurred when trying to get that particular group name.

```
Computer name:   THINAPP-1U4
```

This is the hardware name associated with the computer. This name can also be viewed using the control panel's system applet, under the Computer Name tab.

```
IE version:        7.0.5730.11
```

This is the version of Internet Explorer currently in use. This data can also be viewed in the `Help > About` menu item of IE.

```
Drives: Drive C: Fixed NTFS serial 3892890922 [xE808CD2A], 294,211 MB user
free, CASE_SENSITIVE_SEARCH CASE_PRESERVED_NAMES UNICODE_ON_DISK
PERSISTENT_ACLS FILE_COMPRESSION VOLUME_QUOTAS SUPPORTS_SPARSE_FILES
SUPPORTS_REPARSE_POINTS SUPPORTS_OBJECT_IDS SUPPORTS_ENCRYPTION NAMED_STREAMS

Drive D: CD-ROM CDFS serial 2681771136 [x9FD89480], 0 MB user free,
CASE_SENSITIVE_SEARCH UNICODE_ON_DISK READ_ONLY_VOLUME
```

This is information about each of the physical drives attached to the computer. The entries in capital letters show the various features supported by each drive, and are normally meaningful only to support personnel or experienced users.

```
Working dir:       C:\cmd_test\bin
```

This is the working directory and is also known as the "current directory." This is the directory that is reported whenever an application uses the `GetCurrentDirectory( )` system function.

```
ADO version:       2.82.3959.0
```

This is the version of Active Data Objects (ADO) that is currently installed. ADO is a relatively new (compared with ODBC) Microsoft-only technology. It is a set of COM objects used to access "data sources," providing a layer between a programming language and the OLE DB (whether they are databases or otherwise) in a uniform manner. ADO enables a developer to write programs that access data without knowing how the database is implemented.

```
Terminal services:    Not running
```

This specifies whether Terminal Services is currently active. Terminal Services enable a server to host multiple, simultaneous client sessions. For example, on WinXP Prof along with the Server 2003 family, Remote Desktop uses Terminal Services to allow a single session to run remotely. The Terminal Services Remote Desktop Web Connection extends Terminal Services technology to the Web.

```
Remote session:     Yes
```

This specifies whether a Remote Desktop session is currently being used.

```
LUA enabled:       No
```

This stands for Least-privileged User Account as opposed to an administrator account. This feature usually applies to Microsoft Vista. The principle of "least privilege" is based on the notion that if a low-privileged process is compromised, it will do a lot less damage to a system than a high-privileged process would be capable of doing.

```
LOCALE_SYSTEM_DEFAULT:  0409 ENU
LOCALE_USER_DEFAULT:   0409 ENU
```

Windows employs the concept of "locales" to specify international regions and languages. Each locale is specified by a unique 32-bit identifier that contains a "language id" and a "sort order" for that region (or country).

```
Installed apps
System-wide
    7-Zip 4.57, AddressBook, Branding, Capture Express, Connection Manager,
    DirectAnimation, DirectDrawEx, DXM_Runtime, Fontcore, getPlus(R)_ocx,
    ICW, IDA Pro Standard v4.9 SP, Microsoft Internationalized Domain Names
    Mitigation APIs, IE40, IE4Data, IE5BAKEX, Windows Internet Explorer 7,
    IEData, InstallShield for Microsoft Visual C++ 6, InstallShield Uninstall
    Information, VMware Lab Manager, Microsoft Base Smart Card Cryptographic
 . . .
Per-user
    FileZilla Client 3.0.9.2
```

This is an enumeration of the applications currently installed on the system. Applications appearing in this list were likely placed on the system by an "installer." As part of the installation process, the installer probably added particular keys to the system registry. Note that applications that a user places directly into local storage, such as applications copied from the Internet, will not appear in this list.

Many installers can install an application for use by anyone using the computer, or only for a particular user.

System-wide applications are mined from the registry key

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
```

while per-user applications are taken from registry key

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
Environment strings:   =::=::\
             ALLUSERSPROFILE=C:\Documents and Settings\All Users
             APPDATA=C:\Documents and Settings\ron\Application Data
             CLIENTNAME=LITHIUM
             ClusterLog=C:\WINDOWS\Cluster\cluster.log
             CommonProgramFiles=C:\Program Files\Common Files
             COMPUTERNAME=THINAPP-1U4
             ComSpec=C:\WINDOWS\system32\cmd.exe
             DEFLOGDIR=C:\Documents and Settings\All Users\Application
  . . .
```

This is a listing of the environment strings. These strings can also be viewed by using the Control Panel's `System` applet, and then clicking the **`Advanced > Environment Variables`** button.

```
Current process:
```

This is information about the currently executing application, usually a captured application.

```
Process ID: 4400
```

The "process identifier" of the currently executing application. Windows assigns a unique process ID to each active process. View process IDs by using the `Processes` tab of Task Manager. Note that the terms "process" and "application" are sometimes used interchangeably in Windows. Compare the `Applications` and `Processes` tabs of Task Manager.

```
Minimum req'd operating system ver: 4.0
```

Whenever a Windows application is built, a specification is made of the minimum opsys version needed to run the application. This version is normally made by default (not consciously by the developer) and usually defaults to Windows NT 4.0.

This value, obtained using the system API `GetProcessVersion( )` function, is placed into the application's executable file image header by the linker.

```
notepad.exe
```

The name of the currently executing application. Application names can be viewed using the `Applications` tab of Task Manager.

```
C:\cmd_test\bin\notepad.exe
```

This is the fully-qualified path from which the application was launched. Note that this does not have to be local storage. For example, an application can be run from a network share.

```
CompanyName: Microsoft Corporation
FileDescription: Notepad
FileVersion: 5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)
InternalName: Notepad
LegalCopyright: © Microsoft Corporation. All rights reserved.
OriginalFilename: NOTEPAD.EXE
ProductName: Microsoft® Windows® Operating System
ProductVersion: 5.1.2600.2180
```

This is data contained in the version resource block of an application's executable file. This information can also be viewed by right-clicking on an application's icon or executable file name, selecting the **`Properties`** submenu item, and then clicking the **`Version`** tab, if there is one. Note that many applications do not contain version information.

```
Priority class: Normal
```

Windows NT and later versions of Windows use priority levels to divide CPU time among processes. For example, if Word has a priority level of eight and Excel has a priority level of four, Word can be granted more CPU time (implemented using "time slices"). Changing the priority level can greatly aid an application's performance by instructing the CPU to run one program's code before another.

Windows allows four priority levels to be set: low, normal, high, and real-time. Low runs the application at a lower priority, specifically level four. This is useful for large tasks running in the background because with a low priority, applications do not degrade the responsiveness of other applications. Normal is level eight and is the default. High is level 13 and runs the application at a level equal to some services. This makes the program very responsive, but can slow other processes dramatically.

You can view a process priority level in Task Manager by selecting **`View > Select Columns > Base Priority`** and choosing the **`Processes`** tab.

`Process Affinity Mask: x3`

The processor affinity in Windows is a number which specifies the number of processors that will be used by a running process. 64-bit Windows supports a maximum of 64 processors. 32-bit Windows supports a maximum of 32 processors. An affinity mask value is specified in hexadecimal.

A process affinity mask is a bit vector in which each bit represents the processors that a process is allowed to run on. A process affinity mask is a proper subset of a system affinity mask. The processor affinity mask value can be retrieved by using the `GetProcessAffinityMask( )` API function.

`System affinity mask: x3`

A system affinity mask is a bit vector in which each bit represents the processors that are configured into a system. A process is only allowed to run on the processors configured in a system.

The system affinity mask is also returned by the `GetProcessAffinityMask( )` API function.

`Creation time: Tues 6 Jun 2008 18:56:13`

This is the date and time the current process was executed. This value can be retrieved using the `GetProcessTimes( )` API function.

`Priority boost: Normal behavior`

Each process thread has a dynamic priority. This is the priority the Windows scheduler uses to determine which thread to execute. Initially, a thread's dynamic priority is the same as its base priority. The system can raise and lower the dynamic priority to ensure that it is responsive and that no threads need processor time.

A thread's priority boost information can be retrieved by using the `GetProcessPriorityBoost( )` API function.

```
Modules:
Base     Size      Entry
x01000000 x00014000  x010020BC  notepad.exe   C:\cmd_test\bin\notepad.exe
x7C800000 x000C0000 x00000000   ntdll.dll      C:\WINDOWS\system32\ntdll.dll
x77E40000 x00102000  x77E646B6  kernel32.dll
C:\WINDOWS\system32\kernel32.dll
x77380000 x00091000  x7739D8F7  USER32.dll   C:\WINDOWS\system32\USER32.dll
   . . .
```

This is a list of the DLL modules currently loaded by an application. Note that any modules previously loaded and then unloaded will not show up in this list. Also note that modules in this list are not necessarily loaded directly by the application. Modules can in turn load other modules.

```
32-bit processes:
x0000 idle system process
x0004 system process
x0124 smss.exe            \SystemRoot\System32\smss.exe
x016C winlogon.exe        \??\C:\WINDOWS\system32\winlogon.exe
x019C services.exe         C:\WINDOWS\system32\services.exe
x01A8 lsass.exe            C:\WINDOWS\system32\lsass.exe
x025C svchost.exe          C:\WINDOWS\system32\svchost.exe
x032C svchost.exe          C:\WINDOWS\System32\svchost.exe
x03CC spoolsv.exe          C:\WINDOWS\system32\spoolsv.exe
. . .
```

A list of all the 32-bit processes currently running. Note that processes do not necessarily directly correspond to applications. Any 16-bit processes will not show up in this list.

```
Shell folders:
   ProgramFilesDir = C:\Program Files
   Common StartMenu = C:\Documents and Settings\All Users\Start Menu
   StartMenu = C:\Documents and Settings\ron\Start Menu
   Common AppData = C:\Documents and Settings\All Users\Application Data
   Local AppData = C:\Documents and Settings\ron\Local Settings\Application
               Data
   . . .
```

Shell folders are special folders that Windows uses to indicate the default location for many types of settings and data. Most users are familiar with folders such as Program Files, My Documents, and Documents and Settings.

Use the `SHGetFolderPath( )` AP function to determine where a particular shell folder is located. There are currently about 25 different shell folder specifications.

```
System services:
  AeLookupSvc – Application Experience Lookup Service
    Process ID: 812
    Shared process? Yes
State: Running
    Flags: x00000000 Not running in system process
Alerter – Alerter
    Process ID: 0
    Shared process? Yes
    State: Stopped
    Flags: x00000000 Not running in system process
  . . .
```

System services are relatively long-running executables that perform specific functions, and that normally do not require user intervention.

Windows services can be configured to start when the operating system is first loaded, or they can be loaded at some later point "on demand." Services run in the "background" as long as Windows is running, or until they are either stopped or abort. Windows services are similar in concept to a Unix "daemon."

System services can be viewed in the `Processes` tab of the Task Manager. They are usually identified by a user name such as system, local service, or network service, though not all processes with the system username are actually services.

System services are managed by a special system component known as the Service Control Manager, or SCM.

```
32–bit device drivers:
  Kernel mode drivers:
    Abiosdsk  Abiosdsk
    ACPI  Microsoft ACPI Driver
    ACPIEC  ACPIEC
    adpu160m  adpu160m
     . . .
```

Like system services, device drivers usually run without the awareness of the typical user. There are two classes of drivers: kernel mode and user mode.

Kernel mode drivers are very special types of software than run within ring 0 of the processor, alongside the operating system itself. Drivers are responsible for things such as "talking" to the platform's hardware, managing network connections, interacting with user mode processes, and implementing file systems.

User-mode drivers do many of the things that kernel mode drivers do, except that they do not have the special privileges of kernel drivers. As a result, they are far less likely to cause the kinds of catastrophic problems that wayward kernel mode drivers can cause.

User mode drivers do not appear in the above list.

```
File system and filter drivers:
    Cdfs  Cdfs
    DfsDrive  DfsDriver
    Fastfat  Fastfat
    FltMgr  FltMgr
    MRxDAV  WebDav Client Redirector
    MRxSmb  MRxSmb
     . . .
```

File system drivers and file system filter drivers are special types of kernel mode drivers that deal with the various file systems installed on the computer. Examples include the FAT, NTFS and CDFS file systems.

File system drivers stand between user applications and the hardware drivers that talk directly to the drives. They implement a file system “on top” of the bare hardware layer of a drive (tracks, sectors, cylinders, etc.). For example, a file system's directory and subdirectory structures are implemented by file system drivers.

Filter drivers normally sit “on top of” file system drivers, meaning that they see file system requests before the file system driver does. Filter drivers can filter a filesys operation to do something other than what was originally intended. Specifically, they can reject, modify, or pass an operation through unmodified. Common examples of filter driver types include compression, encryption, quota monitoring, etc.

```
Adapter drivers:
    0 adapter drivers
```

Adapter drivers are special types of kernel mode device drivers that communicate directly with hardware adapters. As such, they use direct I/O and/or DMA machine instructions. Only kernel mode drivers are allowed to execute these types of privileged machine instructions.

Generating MSI Files 10

ThinApp can optionally generate auto-registering MSI files during the build process.

Building an MSI Database

You might consider building an MSI database when you need to deliver ThinApp captured applications through traditional desktop management systems to remote locations and have shortcuts and file type associations automatically created. You can use MSI packages to do this. Most systems including plain Active Directory Group Policy provide simple methods to push and execute MSI packages.

An MSI database created by ThinApp contains the following:

- The captured executable files for a specific project if the database is less than 2GB. If the database exceeds 2GB, it only contains the installer logic and the ThinReg registration utility. For databases larger than 2GB, the captured executable files are stored in two or more cabinet files (named <inventoryname>_1.CAB, <inventoryname>_2.CAB etc.). These files must be distributed with the MSI file to have a complete installer.
- ThinReg registration utility.

When you execute a ThinApp MSI file, the MSI database extracts your captured executable files to `c:\Program Files`. Use the optional configuration option `MSIInstallDirectory` (see "MSIInstallDirectory" on page 63) to control the extraction destination.

If you specify `ALLUSERS=""` to msiexec, or you create an MSI package that installs per-user by default using the configuration option `MSIDefaultInstallAllUsers` (see "MSIDefaultInstallAllUsers" on page 62) then the captured executable files are extracted to the user's `%AppData%` directory.

The MSI installation executes ThinReg to register file types and create shortcuts.

To build an MSI database

1 Edit a package file generated by `Package.ini` to specify `MSIFilename=<filename>.msi` ("MSIFilename" on page 63).

2 Enter **`run build.bat`**.

NOTE If `MSIFilename` is not specified, MSI generation is skipped during the build process.

By default, the generated database installs machine-wide. You can change this by setting the following option:

```
MSIDefaultInstallAllUsers=0
```

(see "MSIDefaultInstallAllUsers" on page 62)

This creates a database with a default of per-user installation.msiexec command-line parameters.

No matter what you specified at package build time, you can still override the type of installation at deployment time. For example, if you created the database with "`MSIDefaultInstallAllUsers=1`", you can still force a per-user deployment by deploying using:

```
msiexec /i <database>.msi ALLUSERS=""
```

To force a per-machine deployment, use:

```
msiexec /i <database>.msi ALLUSERS=1
```

To default to per-machine installation for Administrators (belonging to the "Administrators" group) and per-user for non-Administrators, set the value of `MSIDefaultInstallAllUsers` to 2.

When doing a per-machine deployment, the default installation directory is the localized equivalent of `"%ProgramFilesDir%\<InventoryName> (captured)`. For a per-user deployment, the default is `%AppData%\<InventoryName> (captured)`. In both cases, you can override the installation directory by passing an INSTALLDIR property to msiexec:

```
msiexec /i <database>.msi INSTALLDIR=C:\Mydir\Mypackage
```

Microsoft Vista

For deployment on Vista, you must indicate whether an installer needs elevated privileges or not. Normally, when performing a per-user installation you do not need elevated privileges but when doing a per-machine installation you will. You can specify whether the package needs elevated privileges by using the `MSIRequireElevatedPrivileges` option (see "MSIRequireElevatedPrivileges" on page 65). If you're going to deploy the installer machine-wide, set the value to 1. This results in UAC prompts (if you have them enabled), even if you force a per-user installation by using the ALLUSERS="" command line option. By setting the option to 0, you do not receive a UAC prompt but the installation fails if you try to install it machine-wide.

Handling Upgrades to MSI Applications

A MSI database contains a number of codes, including a `ProductCode` and an `UpgradeCode`. When rebuilding a package, keep the `UpgradeCode` the same. This can be specified using the `MSIUpgradeCode` option. You can change the `ProductCode` (using the MSIProductCode option) if desired. See "MSIUpgradeCode" on page 65.

The advantage of changing the `ProductCode` is that it is easier to install a newer version. If the `ProductCode` of the new version is the same as the `ProductCode` of the old version, the installation prompts you to remove the old version using **`Add/Remove Programs`** first. If the `ProductCode` is different, the installation uninstalls the old version and then installs the new version.

If you do not specify an MSI`ProductCode` option, a different ProductCode is generated for each build automatically.

Using Scripts

11

This chapter contains information on scripting and an API reference and includes the following sections:

Scripting allows you to execute custom code before starting an application packaged with ThinApp or after an application exits.

Callback functions allow you to specify when blocks of code execute.

API functions allow you to execute ThinApp-specific functions and interact with the ThinApp runtime.

To add scripts to your application, create an ANSI text file with the `.vbs` file extension in the root project directory for an application (the same directory in which `Package.ini` is located). During the build process, ThinApp adds all of the script files to your executable file and then at runtime, it executes each of these script files.

Many applications create child processes, and scripts execute for child processes as well. To execute code only in the main parent process, use callback functions.

ThinApp uses VB Script to execute script files, so any valid VB Script loads under a ThinApp executable. Refer to Microsoft's VB Script User Guide for more information. VB script can be used to access COM controls registered on the host system, or registered within the virtual package.

Callback Functions

Callback functions with specific names execute only under certain conditions. For example, callback functions allow script code to execute only when an application starts or quits.

Currently defined callback function names include the following:

- `OnFirstSandboxOwner`—Called only when an application first locks the sandbox. This callback is not called if a second copy of the same application is launched using the same sandbox while the first copy is still running. If a first application spawns sub-process and then quits, the sandbox remains locked by the second sub-process so this callback does not execute again until all sub processes have quit and the application is run again.
- `OnFirstParentStart`—Called prior to executing a ThinApp executable file regardless of whether the sandbox is simutaneously owned by another captured executable.
- `OnFirstParentExit`—Called when the first parent process exits. If a parent process executes a child process and quits, this callback is called even if the child process continues to execute.
- `OnLastProcessExit`—Called when the last process owning the sandbox exits. If a parent process executes a child process and quits, this callback is called when the last child process exits.

```
------------------------example.vbs ----------------------------------
Function OnFirstSandboxOwner
msgbox "The sandbox owner is: " + GetCurrentProcessName
End Function

Function OnFirstParentExit
msgbox "Quiting application: " + GetCurrentProcessName
End Function

msgbox "This code will execute for all parent and child processes"
-------------------------------------------------------------------------
```

Example Scripts

The following are example scripts:

- Timeout an application on a specific date.
- Run a `.bat` file from a network share inside the virtual environment.
- Modifying the virtual registry.

- Import `.reg` file at runtime.
- Stop a virtual service when the main application quits.
- Copy an external system configuration file into the virtual environment on startup.

To use an example

1 Save contents to a plain text file with the `.vbs` extension in the same directory as your `Package.ini` file (produced by Setup Capture).

It doesn't matter what filename you use, all .vbs files are added to the package at build time and run.

2 Rebuild the application.

.bat example

This script executes an external `.bat` file from a network share inside of the virtual environment. The `.bat` file can make modifications to the virtual environment by copying files, deleting files, or apply registry changes using `regedit /s regfile.reg`. Make sure you only execute this for the first parent process otherwise each copy of `cmd.exe` executes the script and an infinite recursion will develop.

```
Function OnFirstParentStart
 Set Shell = CreateObject("Wscript.Shell")
 Shell.Run "\\jcdesk2\test\test.bat"
End Function
```

timeout example

Prevent an application from being used after a specified date.

This check is performed on the parent process startup as well as any child process startup.

```
ExpirationDate = CDate("03-20-07")

if Date >= ExpirationDate then
 msgbox "This application has expired, please contact Administrator"
 ExitProcess 0
end if
```

Registry Modify

This example modifies the virtual registry at runtime so that an external ODBC driver can be loaded from the same directory where the package executable is located.

Get path to package executable files.

```
Origin = GetEnvironmentVariable("TS_ORIGIN")
```

Find last slash in path and grab just the characters before this.

```
LastSlash = InStrRev(Origin, "\")
SourcePath = Left(Origin, LastSlash)
```

Form a new path to the ODBC DLL file located outside of the package.

```
DriverPath=SourcePath + "tsodbc32.dll"
```

Modify the virtual registry so that it points to this location. This causes the application to load the DLL from an external location.

```
Set WSHShell = CreateObject("Wscript.Shell")
WSHShell.RegWrite "HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\Transoft
ODBC Driver\Driver," DriverPath
```

.reg example

Import the registry values from an external `.reg` file into the virtual registry at runtime.

```
Function OnFirstParentStart
ExecuteVirtualProcess "regedit /s c:\tmp\somereg.reg"
End Function
```

Stopping Service

Stop a virtual or real service when the main application quits.

```
Function OnFirstParentExit
Set WshShell = CreateObject("WScript.Shell")
WshShell.Run "net stop ""iPod Service"""
End Function
```

Copyfile Example

The following scripts demonstrate how you can copy a configuration file located in the same directory as your captured executable into the virtual file system each time the application starts. One use for this is to enable you to have an external configuration file that is easy to edit after deployment. Because the copy occurs each time you run the application, any edits to the external version are reflected in the virtual version.

For example, if your captured executable is running from `\\server\share\myapp.exe`, this script looks for a configuration file located at `\\server\share\config.ini` and copies it to the virtual file system location: `c:\Program Files\my application\config.ini.`

By putting this code in the OnFirstParentStart function, it is only called once per execution. Otherwise it is executed for every child process as well.

```
Function OnFirstParentStart
```

`TS_ORIGIN` is set by ThinApp to indicate the full path to a captured executable package. A virtual application sets the TS_ORIGIN variable to the physical path of the main data container.

If you have a virtual application consisting of a main.exe and a shortcut.exe, both of them are placed in C:\VirtApp. When you run C:\VirtApp\main.exe the TS_ORIGIN var will be set to C:\VirtApp\main.exe. When you run C:\VirtApp\shortcut.exe, the TS_ORIGIN environment variable will be set to C:\VirtApp\main.exe (it is always set to the main data container, even if you create a shortcut). This happens early during process startup. When you run VBScripts that are included in the package, the variable has already been set and it is available to the VBScripts.

```
Origin = GetEnvironmentVariable("")
```

Separate the filename from TS_ORIGIN by finding the last backslash and remove everything after this.

```
LastSlash = InStrRev(Origin, "\")
SourcePath = Left(Origin, LastSlash)
```

The source file to copy into the virtual environment is the package path plus `config.ini.`

```
SourceFile = SourcePath + "Config.ini"
```

The location we want to copy to might be in a different location on different computers if the `Program Files` directory is mapped to a different location than `c:\` The following call lets ThinApp expand a macro to get the correct location for the local PC.

```
DestFile = ExpandPath("%ProgramFilesDir%\MyApplication\Config.ini")
```

Use the file systemObject to check to make sure the source file exists.

```
Set objFSO = CreateObject("Scripting.file systemObject")
If objFSO.FileExists(SourceFile) Then
```

If the source file exists, copy it into the virtual file system. The virtual directory `%ProgramFilesDir%\MyApplication` is in the package.

```
objFSO.CopyFile SourceFile, DestFile, TRUE
End if
End Function
```

System Registry Example

This script adds a value to the system registry.

Create a `.reg` file and execute "`regedit /s`" as an external process so it accesses the system registry instead of the virtual registry.

```
Function OnFirstParentStart
```

Create the `.reg` file in a place that has `IsolationMode` set to `Merged`, so it can be easily accessed from both the virtual environment (this script) and the system environment (regedit /s). One directory that has a `Merged IsolationMode` by default is the `%Personal%` directory, so create the .reg file there.

```
RegFileName = ExpandPath("%Personal%\thin.reg")
Set fso = CreateObject("Scripting.file systemObject")
Set RegFile = fso.CreateTextFile(RegFileName, true)
```

Construct the `.reg` file.

```
RegFile.WriteLine("Windows Registry Editor Version 5.00")
RegFile.WriteBlankLines(1)
RegFile.WriteLine("[HKEY_CURRENT_USER\Software\Thinapp\demo]")
RegFile.WriteLine(chr(34) and "InventoryName" and chr(34) and "=" and chr(34)
and GetBuildOption("InventoryName") and chr(34))
RegFile.Close
```

Enter the information in the system registry, wait until it is done.

```
RegEditPid = ExecuteExternalProcess("regedit /s " and chr(34) and RegFileName
and chr(34))
WaitForProcess RegEditPid, 0
```

Clean up.

```
fso.DeleteFile(RegFileName)
End Function
```

API Reference

This chapter contains API reference information for ThinApp.

AddForcedVirtualLoadPath

Function AddForcedVirtualLoadPath(Path)

This function instructs ThinApp to load all DLLs from the specified path as virtual DLLs even if they are not located in the package. Use this function if the application needs to load external DLLs that have dependencies on DLLs located inside the package.

Parameters

Path

[in] The filename or path for DLLs to load as virtual.

Example

Load any DLL located in the same directory as the executable as a virtual DLL.

TS_ORIGIN is the path from which the executable is running.

```
Origin = GetEnvironmentVariable("TS_ORIGIN")
```

To delete the filename from TS_ORIGIN, so find the last backslash and remove everything after it.

```
LastSlash = InStrRev(Origin, "\")
SourcePath = Left(Origin, LastSlash)
```

Tell ThinApp to load all DLLs in the same directory (or deeper) from where the source executable is.

This allows us to drop additional files in the SourcePath tree and have them resolve imports against virtual DLLs.

```
AddForcedVirtualLoadPath(SourcePath)
```

ExitProcess

Sub ExitProcess(ExitCode)

This function quits the current process and sets the specified Error Code.

Parameters

ExitCode

[in] The error code to set. This information might be available to a parent process. Typically a value of 0 is used to indicate no error.

Example

Exit the process and indicate success.

```
ExitProcess 0
```

NOTE As the process exits, the scripting system receives its `Function OnEndProcess` callback. Also, any DLLs that are loaded run their termination code so they can properly clean up.

ExpandPath

Function ExpandPath(InputPath)

This function converts a path from macro format to system format.

Parameters

InputPath

[in] A path in macro format.

Returns

The expanded macro path in system format.

Example

Path = `ExpandPath("%ProgramFilesDir%\Myapp.exe")`

Path = `c:\Program Files\myapp.exe`

NOTE All macro paths must escape the % and # characters by replacing these characters with `#25` and `#23.`

ExecuteExternalProcess

Function ExecuteExternalProcess(CommandLine)

This function executes a command outside of the virtual environment. It can be used if you need to make real system changes.

Parameters

Command Line

[in] CommandLine represents the application and commandline parameters to execute outside of the virtual environment.

Returns

Integer process ID, the process ID can be used with WaitForProcess ("WaitForProcess" on page 166).

Example

```
ExecuteExternalProcess("cmd.exe /c copy c:\systemfile.txt
c:\newsystemfile.txt")
```

Execute a command that requires quotes in the command-line.

```
ExecuteExternalProcess("regsvr32 /s " and chr(34) and "c:\Program
Files\my.ocx" and chr(34))
```

ExecuteVirtualProcess

Function ExecuteVirtualProcess(CommandLine)

This function executes a command inside of the virtual environment. It can be used if you need to make changes to the virtual environment.

Parameters

CommandLine

[in] CommandLine represents the application and commandline parameters to execute outside of the virtual environment.

Returns

Integer process ID, the process ID can be used with WaitForProcess ("WaitForProcess" on page 166).

Example

```
ExecuteVirtualProcess("cmd.exe /c copy c:\systemfile.txt c:\virtualfile.txt")
```

Execute a command that requires quotes in the command-line.

```
ExecuteVirtualProcess("regsvr32 /s " and chr(34) and "c:\Program
Files\my.ocx" and chr(34))
```

GetBuildOption

Function GetBuildOption(OptionName)

This function returns the value of a setting specified in the [BuildOptions] section of the `Package.ini` file used for capturing.

Parameters

Option Name

[in] Name of the setting.

Returns

This function returns a string value. If the requested Option Name does not exist an empty string "" is returned.

Example

```
Package.ini contains:
[BuildOptions]
DemoOption=DemoValue
```

The following line would appear in a VBS file:

```
Value = GetBuildOption("DemoOption")
```

Value = "DemoValue"

GetFileVersionValue

Function GetFileVersionValue(Filename, Value)

This function returns version information value from a specific DLL, executable, OCX, etc. This function can be used to determine the internal version number of a DLL or retrieve information about the copyright owner of a DLL, or product name associated with the DLL.

Parameters

Filename

[in] The name of the `filename` whose version information is being retrieved.

Value

[in] The name of the Value to retrieve from the version information section of the specified file

The following values can be retrieved from most DLLs:

Comments
InternalName
ProductName
CompanyName
LegalCopyright
ProductVersion
FileDescription
LegalTrademarks
PrivateBuild
FileVersion
OriginalFilename
SpecialBuild

Returns

This function returns a string value. If the requested Filename does not exist or the specified Value cannot be located in the file, the value of "" is returned.

Example

```
FileVersion = GetFileVersionValue("c:\windows\system32\kernel32.dll,"
               "FileVersion")

if FileVersion = "1.0.0.0" then
                MsgBox "This is Version 1.0!"

End if
```

GetCommandLine

Function GetCommandLine

This function allows you to access the command-line parameters passed to the current running program

Returns

This function returns a string that represents the command line arguments passed to the current running program, including the original executable.

Example

```
MsgBox "The command line for this EXE was " + GetCommandLine
```

GetCurrentProcessName

Function GetCurrentProcessName

This function allows you to access the full virtual path name of the current process.

Returns

This function returns a string that represents the full executable path name inside of the virtual environment. Typically this is `c:\Program Files\...` even though the package source might be running from a network share.

Example

```
MsgBox "Running EXE path is " + GetCurrentProcessName
```

GetOSVersion

Function GetOSVersion()

This function returns information about the current version of Windows.

Parameters

This function has no parameters.

Returns

This function returns a string in the following format:

```
MAJOR.MINOR.BUILD_NUMBER.PLATFORM_ID OS_STRING
```

`MAJOR` is one the following values

Windows Server 2003	5
Windows XP	5
Windows 2000	5
Windows NT 4.0	4
Windows Me	4
Windows 98	4
Windows 95	4
Windows NT 3.51	3

MINOR is one of the following values

Windows Server 2003	2
Windows XP	1
Windows 2000	0
Windows NT 4.0	0
Windows Me	90
Windows 98	10
Windows 95	0
Windows NT 3.51	51

BUILD_NUMBER is the build number of the OS.

Windows Me/98/95: The low-order word contains the build number of the operatingsystem. The high-order word contains the major and minor version numbers.

PLATFORM_ID is one of the following values.

Value = 1 for Windows Me, Windows 98, or Windows 95 (Windows 95 based OS)

Value = 2 for Windows Server 2003, Windows XP, Windows 2000, or Windows NT. (Windows NT based OS)

OS_STRING Represents additional information about the OS such as "Service Pack 2"

Example

```
if GetOSVersion() = "5.1.0.2 Service Pack 2"
                then MsgBox "You are running on Windows XP Service Pack 2!"
endif
```

GetEnvironmentVariable

Function GetEnvironmentVariable(Name)

This function returns the environment variable associated with variable name Name.

Parameters

Name

[in] The name of the environment variable for which the value is to be retrieved.

Returns

This function returns the string value associated with the environment variable "name."

Example

```
MsgBbox "The package source EXE is " + GetEnvironmentVariable("TS_ORIGIN")
```

RemoveSandboxOnExit

Sub RemoveSandboxOnExit(YesNo)

This function set toggles whether to delete the sandbox when the last child process exits.

If you set the `Package.ini` option `RemoveSandboxOnExit=1`, the default clean up for the package with be "Yes." In this case you, can change the clean up to "No" by calling RemoveSandboxOnExit with the value of 0. If you did not modify the `Package.ini` option `RemoveSandboxOnExit=1`, the default clean up for the package with be "No." In this case you, can change the clean up to "Yes" by calling RemoveSandboxOnExit with the value of 1.

Parameters

Yes No

[in] Do you want to clean up when the last process shuts down? 1=Yes, 0=No

Example

This turns on cleanup

```
RemoveSandboxOnExit 1
```

This turns off cleanup

```
RemoveSandboxOnExit 0
```

SetEnvironmentVariable

Sub SetEnvironmentVariable(Name, Value)

This function set the value of an environment variable.

Parameters

Name

[in] The name of environment variable where the value is to be stored.

Value

[in] The value to be stored.

Example

```
SetEnvironmentVariable "PATH", "C:\Windows\system32"
```

SetfileSystemIsolation

Sub Setfile systemIsolation(Directory, IsolationMode)

This function sets the isolation mode of a directory.

Parameters

Directory

[in] Full path of the directory whose isolation mode is to be set.

IsolationMode

[in] Isolation mode to be set
1 = WriteCopy
2 = Merged
3 = Full

Example

Set the isolation mode of the temp directory to Merged.

```
Setfile systemIsolation GetEnvironmentVariable("TEMP"), 2
```

Availability

This option requires ThinApp version 3.215 or higher.

SetRegistryIsolation

Sub SetRegistryIsolation(RegistryKey, IsolationMode)

This function sets the isolation mode of a registry key.

Parameters

RegistryKey

[in] The registry key whose isolation mode is to be set. Start with "HKLM" for HKEY_LOCAL_MACHINE, "HKCU" for HKEY_CURRENT_USER of "HKCR" for HKEY_CLASSES_ROOT

IsolationMode

[in] Isolation mode to be set
1 = WriteCopy
2 = Merged
3 = Full

Example

Set isolation of HKEY_CURRENT_USER\Software\Thinapp\Test to Full

```
SetRegistryIsolation "HKCU\Software\Thinapp\Test," 3
```

Availability

This option requires ThinApp version 3.215 or higher.

WaitForProcess

Function WaitForProcess(ProcessID, TimeOutInMilliSeconds)

This function waits until the specified ProcessID has completed execution.

Parameters

ProcessID

[in] The processID to wait for completion. The process ID can come from ExecuteExternalProcess or ExecuteVirtualProcess.

TimeOutInMilliSeconds

[in] The maximum amount of time to wait for the process to end before continuing. If 0 is specified, INFINITE is used.

Returns

This function returns an integer
0 = Timeout failed
1 = Process exited
2 = The process does not exists or security denied

Example

```
id = ExecuteExternalProcess("cmd.exe")
WaitForProcess(id, 0)
```

12 Troubleshooting

This chapter describes troubleshooting for ThinApp and contains the following sections:

Use volume license keys when they are available. This usually eliminates problems in which applications refuse to run on a machine different from the packaging machine.

Applications that use per-machine network activation might not work when they are run after being captured on one machine and run on a different machine.

When packaging applications that bind to a specific host, do not run your virtual application prior to snapshot finalization.

Some applications might store hard-coded path information in data files and might operate incorrectly when roaming to a new computer.

Contacting Technical Support

You can contact VMware Technical Support at the following URL:

http://www.vmware.com/support/

To make your contact with Technical Support as effective as possible, please be prepared to provide the following:

- Step-by-step reproduction of the procedure you were performing when you encountered a problem. Make sure to provide a clear, concise description of your procedure. Include all relevant details in your procedure so that it can be easily replicated.
- Your host configuration information. Include the Windows operating system you use, whether you use Terminal Services or Citrix, and any additional prerequisite programs that you installed natively.
- Copies of your trace files.

  NOTE Generate ASCII text files from your binary trace files and zip the files before sending them.

- A unmodified copy of your capture folder and all content. Do not include the compiled executables from the BIN subfolder.
- A description of the expected behavior, versus what actually happened.
- A copy of applications that you captured, if possible.
- Any non-virtual files or registry key settings that might be relevant.
- Any system services or device drivers required.
- Your server components configuration (Oracle Server, Active Directory, etc.).

In addition, you can provide the following optional components to assist Technical Support to help you:

- A copy of your sandbox folder.
- A virtual machine that you can upload to Technical Support containing the affected system. If your operating system is not on a virtual machine, use VMware VM Converter to convert your operating system into a virtual machine.

  VM Converter is a free application that you can download at the following URL:

  http://www.vmware.com/download/converter/

Troubleshooting Specific Applications

The following sections describe troubleshooting tips that you can use to create packages for certain applications.

Microsoft Office 2007

A potential conflict exists when you run a captured Microsoft Office 2007 on a system that has a full native installation of Microsoft Office 2003. In this configuration, the captured Microsoft Office 2007 applications detect that the "Handwriting" feature of Microsoft Office 2003 was installed but, because of isolation, the applications do not see some of the files/registry values belonging to that feature. This causes an automatic repair to be started that shows up as a Windows installer pop-up window during each launch of a captured Microsoft Office 2007 application.

To prevent the repair, make the files of the Microsoft Office 2003 Handwriting feature visible to the captured Microsoft Office 2007 applications. This can be done by changing the isolation mode on the following directories:

```
%ProgramFilesDir%\Common Files\Microsoft Shared\INK
%ProgramFilesDir%\Common Files\Microsoft Shared\Office11
%ProgramFilesDir%\Common Files\Microsoft Shared\Office11\1033
%ProgramFilesDir%\Microsoft Office
```

NOTE Depending on the language version of your installation, you might have a different directory instead of `%ProgramFilesDir%\Common Files\Microsoft Shared\Office11\1033`.

After a fresh capture, these directories contain a `##Attributes.ini` file with `DirectoryIsolationMode=Full`. Change this to `DirectoryIsolationMode=WriteCopy`. The isolation mode of a couple of registry files needs to be adjusted too. Edit `HKEY_LOCAL_MACHINE.txt` and search for these keys:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\UserData\S-1-5-18\Components\379E92CC2CB71D119A12000A9CE1A22A
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\UserData\S-1-5-18\Components\1650EACF3C291D11A92C0006794C4E25
```

Change the `isolationmode_full` at the beginning of the lines to `isolationmode_writecopy`.

Microsoft Outlook

Outlook stores its account settings and other data in a number of registry keys and files. When you start Microsoft Outlook for the first time, it checks if these keys exist and, if not, prompts you to create a new account.

This works fine in the virtual environment when Microsoft Outlook is not installed on the physical system. However, when the user already has Microsoft Outlook installed physically, the captured version finds the registry keys in the system registry and use those settings, which is probably not what is intended. Full Isolation Mode is required for the registry keys and files where Microsoft Outlook stores its settings, so the captured version pretends the system settings do not exist. To set this up, some changes need to be made to the capture. Add the following to the `HKEY_CURRENT_USER.txt` file:

```
isolation_full HKEY_CURRENT_USER\Identities
isolation_full HKEY_CURRENT_USER\Software\Microsoft\Windows
NT\CurrentVersion\Windows Messaging Subsystem\Profiles
```

Next, a file named `##Attributes.ini` with the following contents:

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

This setting must be created in each of the following subdirectories:

```
%AppData%\Microsoft\AddIns
%AppData%\Microsoft\Office
%AppData%\Microsoft\Outlook
%Local AppData%\Microsoft\FORMS
%Local AppData%\Microsoft\Outlook
```

Create the subdirectories as needed.

Attachments

By default, Microsoft Outlook creates a directory where it stores attachments when you open an attachment for viewing. This directory is typically `C:\Documents and Settings\<username>\Local Settings\Temp\Temporary Internet Files\OLKxxxx` where the last "xxxx" is replaced by something randomly chosen.

This works fine when the viewing application runs in the same virtual sandbox as Microsoft Outlook. However, external application might not be able to find the file it is supposed to show. This is because Microsoft Outlook stores the file in the sandbox. To solve this, the isolation mode of the directory where the attachments are stored needs to be set to `Merged`.

Next you need to find out the name of the directory (the last part of the name was randomly chosen). To solve this, add a value to `HKEY_CURRENT_USER.txt` that sets the name of attachment directory:

```
isolation_full
HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Outlook\Security
Value=OutlookSecureTempFolder
REG_SZ~%Profile%\Local Settings\OutlookTempxxxx#2300
```

NOTE The "11.0" in the key name is for Outlook 2003, use the appropriate version number for your version of Outlook.

For slightly increased security, replace the last four "xxxx" characters with something random. Next, create a directory `%Profile%\Local Settings\OutlookTempxxxx` in your project and create an `##Attributes.ini` file there containing:

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

The directory `%Profile%\Local Settings\OutlookTempxxxx` is just an example, you can use whatever you want, as long as you make sure that the directory is named in the OutlookSecureTempFolder registry key and is set to the correct isolation mode.

Explore.exe

The Windows Shell (`explorer.exe`) checks for an already running copy using `GetShellWindow()`. If `explorer.exe` detects a shell is already running, it sends a message to the running copy and exits. If you want to run `explorer.exe` in the virtual environment, you can do the following steps:

1 Kill the currently running process `explorer.exe` using Task Manager

2 Run the `explorer.exe` from a ThinApp virtual environment

Java Runtime Environment (JRE)

A conflict might occur if one version of Java is installed natively and another one is included in a captured executable. This is because newer versions of Java also install a plug-in DLL that gets loaded by `IE`. This plug-in DLL overwrites virtual registry keys when it is loaded and can conflict with a virtualized copy of older Java runtimes. You can prevent IE from loading plug-ins by adding the following line to beginning of `HKEY_LOCAL_MACHINE.txt`.

```
isolation_full
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser
Helper Objects
```

Index

T

U

V