

ThinApp User's Guide

ThinApp 4.7

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see <http://www.vmware.com/support/pubs>.

EN-000400-01

vmware®

You can find the most up-to-date technical documentation on the VMware 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

Copyright © 2010 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at <http://www.vmware.com/go/patents>.

VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

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

Contents

About This Book 7

1 Installing ThinApp 9

- ThinApp Requirements 9
 - Operating Systems, Applications, and Systems That ThinApp Supports 9
 - Applications That ThinApp Cannot Virtualize 10
- Recommendations for Installing ThinApp 10
 - Using a Clean Computer 10
 - Using the Earliest Operating System Required for Users 11
- Install ThinApp Software 11
- Checking ThinApp Installation Files 11

2 Capturing Applications 13

- Phases of the Capture Process 13
- Preparing to Capture Applications 13
- Capturing Applications with the Setup Capture Wizard 14
 - Create a System Image Before the Application Installation 14
 - Rescan the System with the Installed Application 14
 - Defining Entry Points as Shortcuts into the Virtual Environment 15
 - Set Entry Points 15
 - Manage with VMware Horizon Application Manager 15
 - Set User Groups 16
 - Defining Isolation Modes for the Physical File System 16
 - Set File System Isolation Modes 18
 - Storing Application Changes in the Sandbox 18
 - Customize the Sandbox Location 18
 - Send Anonymous Statistics to VMware 18
 - Customize ThinApp Project Settings 19
 - Defining Package Settings 19
 - Customize Package Settings 20
 - Opening Project and Parameter Files 20
 - Build Virtual Applications 21
- Advanced Package Configuration 21
 - Modifying Settings in the Package.ini File 21
 - Modifying Settings in the ##Attributes.ini File 22
- Capturing Internet Explorer 6 on Windows XP 22
 - Requirements for Capturing Internet Explorer 6 on Windows XP 23
 - Capture Internet Explorer 6 on Windows XP by Using the Setup Capture Wizard 23
 - Extracting and Registering ThinDirect 24
- Capturing Multiple Application Installers with ThinApp Converter 24
 - ThinApp Converter Process 24
 - System Requirements for Running ThinApp Converter 25
 - Preparing the Configuration File for ThinApp Converter 25
 - Predefined Environment Variables 31

3 Deploying Applications 33

- ThinApp Deployment Options 33
- Deploying ThinApp with Deployment Tools 33

Deploying ThinApp in the VMware View Environment	33
Deploying ThinApp on Network Shares	34
Deploying ThinApp Using Executable Files	34
Establishing File Type Associations with the thinreg.exe Utility	34
Application Sync Effect on the thinreg.exe Utility	34
Run the thinreg.exe Utility	35
Optional thinreg.exe Parameters	35
Building an MSI Database	37
Customizing MSI Files with Package.ini Parameters	37
Modify the Package.ini File to Create MSI Files	37
Controlling Application Access with Active Directory	39
Package.ini Entries for Active Directory Access Control	39
Starting and Stopping Virtual Services	40
Automatic Startup for Virtual Services	40
Using ThinApp Packages Streamed from the Network	41
How ThinApp Application Streaming Works	41
Requirements and Recommendations for Streaming Packages	42
Stream ThinApp Packages from the Network	43
Using Captured Applications with Other System Components	43
Performing Paste Operations	43
Accessing Printers	43
Accessing Drivers	43
Accessing the Local Disk, the Removable Disk, and Network Shares	43
Accessing the System Registry	44
Accessing Networking and Sockets	44
Using Shared Memory and Named Pipes	44
Using COM, DCOM, and Out-of-Process COM Components	44
Starting Services	44
Using File Type Associations	44
Sample Isolation Mode Configuration Depending on Deployment Context	45
View of Isolation Mode Effect on the Windows Registry	45
4 Updating and Linking Applications	47
Application Updates That the End User Triggers	47
Application Sync Updates	47
Application Link Updates	50
Application Updates That the Administrator Triggers	54
Forcing an Application Sync Update on Client Machines	55
Updating Applications with Runtime Changes	55
Automatic Application Updates	56
Dynamic Updates Without Administrator Rights	57
Upgrading Running Applications on a Network Share	57
File Locks	57
Upgrade a Running Application	57
Sandbox Considerations for Upgraded Applications	58
Updating the ThinApp Version of Packages	58
relink Examples	58
5 Locating the ThinApp Sandbox	59
Search Order for the Sandbox	59
Controlling the Sandbox Location	61
Store the Sandbox on the Network	61
Store the Sandbox on a Portable Device	61
Sandbox Structure	62
Making Changes to the Sandbox	62

Listing Virtual Registry Contents with vregtool	62
6 Creating ThinApp Snapshots and Projects from the Command Line	63
Methods of Using the snapshot.exe Utility	63
Creating Snapshots of Machine States	63
Creating the Template Package.ini file from Two Snapshot Files	64
Creating the ThinApp Project from the Template Package.ini File	64
Displaying the Contents of a Snapshot File	65
Sample snapshot.exe Commands	65
Create a Project Without the Setup Capture Wizard	65
Customizing the snapshot.ini File	66
7 ThinApp File System Formats and Macros	67
Virtual File System Formats	67
ThinApp Folder Macros	67
List of ThinApp Macros	68
Processing %SystemRoot% in a Terminal Services Environment	69
8 Creating ThinApp Scripts	71
Callback Functions	71
Implement Scripts in a ThinApp Environment	72
.bat Example	72
Timeout Example	72
Modify the Virtual Registry	73
.reg Example	73
Stopping a Service Example	73
Copying a File Example	73
Add a Value to the System Registry	74
API Functions	75
AddForcedVirtualLoadPath	75
ExitProcess	75
ExpandPath	76
ExecuteExternalProcess	76
ExecuteVirtualProcess	77
GetBuildOption	77
GetFileVersionValue	77
GetCommandLine	78
GetCurrentProcessName	78
GetOSVersion	79
GetEnvironmentVariable	80
RemoveSandboxOnExit	80
SetEnvironmentVariable	80
SetfileSystemIsolation	81
SetRegistryIsolation	81
WaitForProcess	81
9 Monitoring and Troubleshooting ThinApp	83
Providing Information to Technical Support	83
Log Monitor Operations	83
Troubleshoot Activity with Log Monitor	84
Perform Advanced Log Monitor Operations	84
Log Format	86
Troubleshooting Specific Applications	90
Troubleshoot Registry Setup for Microsoft Outlook	90
Viewing Attachments in Microsoft Outlook	90

Starting Explorer.exe in the Virtual Environment	91
Troubleshooting Java Runtime Environment Version Conflict	91

Glossary	93
----------	----

Index	97
-------	----

About This Book

The *ThinApp User's Guide* provides information about how to install ThinApp™, capture applications, deploy applications, and upgrade applications. You can refer to this guide to customize parameters and perform scripting.

Intended Audience

This book is intended for anyone who installs ThinApp and deploys captured applications. Typical users are system administrators responsible for the distribution and maintenance of corporate software packages.

VMware ThinApp Documentation

The complete documentation set for VMware ThinApp consists of the following documents.

- ThinApp User's Guide. Conceptual and procedural information to help you complete a task.
- ThinApp 4.6 Release Notes. Late-breaking news and descriptions of known issues and workarounds.
- Migrating Applications with ThinApp During an Upgrade from Microsoft Windows XP to Windows 7. Procedural information for using ThinApp to migrate applications from Windows XP to Windows 7.

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. To access the current version of this book and other books, go to <http://www.vmware.com/support/pubs>.

Online and Telephone Support

To 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.

Support Offerings

To 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>.

Legal Notice

ThinApp uses the regular expression library originally written by Henry Spencer.

Copyright (c) 1986, 1993, 1995 by University of Toronto.

Written by Henry Spencer. Not derived from licensed software.

Permission is granted to anyone to use this software for any purpose on any computer system, and to redistribute it in any way, subject to the following restrictions:

- 1 The author is not responsible for the consequences of use of this software, no matter how awful, even if they arise from defects in it.
- 2 The origin of this software must not be misrepresented, either by explicit claim or by omission.
- 3 Altered versions must be plainly marked as such, and must not be misrepresented (by explicit claim or omission) as being the original software.
- 4 This notice must not be removed or altered.

Installing ThinApp

You can install ThinApp to isolate applications, simplify application customization, deploy applications to different operating systems, and eliminate application conflict.

This information includes the following topics:

- [“ThinApp Requirements”](#) on page 9
- [“Recommendations for Installing ThinApp”](#) on page 10
- [“Install ThinApp Software”](#) on page 11
- [“Checking ThinApp Installation Files”](#) on page 11

ThinApp Requirements

Review the requirements for operating systems and captured applications before installing ThinApp.

Operating Systems, Applications, and Systems That ThinApp Supports

ThinApp supports various operating systems, applications, and systems.

- 32-bit platforms include Windows NT, Windows 2000, Windows XP, Windows XPE, Windows 2003 Server, Windows Vista, Windows Server 2008, Windows 7
- 64-bit platforms include Windows XP 64 bit, Windows 2003 64 bit, Windows Vista 64 bit, Windows Server 2008 64 bit, Windows Server 2008 R2 64 bit, Windows 7 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 Xenapp

ThinApp supports Japanese applications captured and run on Japanese operating systems.

Certain operating systems and applications are not supported by ThinApp.

- 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 Virtualize

ThinApp cannot convert some applications into virtual applications and might block certain application functions.

You must use traditional installation technologies to deploy some application types.

- Applications that do not natively support the deployment operating system.

If an operating system does not support the native installation of an application, that operating system is not a supported ThinApp deployment platform for that application.

- Applications requiring installation of kernel-mode device drivers

ODBC drivers work because they are user mode drivers.

- Antivirus and personal firewalls
- Scanner drivers and printer drivers
- Some VPN clients

Device Drivers

Applications that require device drivers do not work when packaged with ThinApp. You must install those device drivers in their original format on the host computer. Because ThinApp does not support virtualized device drivers, you cannot use ThinApp to virtualize antivirus, VPN clients, personal firewalls, and disk and volume mounting-related utilities.

If you capture Adobe Acrobat, you can modify and save PDF files, but you cannot use the PDF printer driver that enables you to save documents to PDF format.

Shell Integration

Some applications that provide shell integration have reduced functions when they exist in a ThinApp package. For example, a virtual application that integrates with Windows Explorer cannot add specific entries to the Windows Explorer context menus.

DCOM Services that are Accessible on a Network

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

Global Hook Dynamic Link Libraries

Some applications use the `SetWindowsHookEx` API function to add a DLL file to all processes on the host computer. The DLL intercepts Windows messages to capture keyboard and mouse input from other applications. ThinApp ignores requests from applications that use the `SetWindowsHookEx` function to try to install global hook DLLs. ThinApp might reduce the application functions.

Recommendations for Installing ThinApp

When you install ThinApp, consider the recommendations and best practices for the software.

Using a Clean Computer

VMware recommends using a clean computer to install ThinApp because the environment affects the application capture process. A clean computer is a physical or virtual machine with only a Windows operating system installed. In a corporate environment where you have a base desktop image, the base desktop image is your clean computer. The desktop computer might already have some components and libraries installed.

Application installers skip files that already exist on the computer. If the installer skips files, the ThinApp package does not include them during the application capture process. The application might fail to run on other computers where the files do not exist. A clean computer enables the capture process to scan the computer file system and registry quickly.

If you install ThinApp and capture an application on a computer that has Microsoft .NET 2.0 already installed, .NET 2.0 is not included in the ThinApp package. The captured application runs only on computers that have .NET 2.0 already installed.

Using Virtual Machines for Clean Computers

The easiest way to set up a clean computer 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 capture an application, you can restore the snapshot and revert it to a clean virtual machine state that is ready for the next application capture.

You can use VMware Workstation or other VMware products to create virtual machines. For information about VMware products, see the VMware Web site.

Using the Earliest Operating System Required for Users

Install ThinApp on a clean computer with the earliest version of the operating system you plan to support. In most cases, the earliest platform 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 lacks. ThinApp excludes these DLLs from the captured application package if the application typically installs these DLLs.

After you create a ThinApp application package, you can overwrite files in the package with updated versions and rebuild the application without the capture process.

Install ThinApp Software

Use the ThinApp executable file to install ThinApp.

Install ThinApp software

- 1 Download ThinApp to a clean physical or virtual Windows machine.
- 2 Double-click the ThinApp executable file.
- 3 In the **Patent Lists** dialog box, click **Next**.
- 4 Accept the license, type the serial number, and type a license display name that appears when you open applications that ThinApp captures.
- 5 Click **Install**.

ThinApp is installed.

Checking ThinApp Installation Files

The ThinApp installation generates the VMware ThinApp directory in C:\Program Files\VMware. You might check the files in this directory to perform operations such as starting the Log Monitor utility to view recent activity.

The following key files in the VMware ThinApp directory affect ThinApp operations:

- **AppSync.exe** – Keeps captured applications up to date with the latest available version.
- **logging.dll** – Generates .trace files.
- **dll_dump.exe** – Lists all captured applications that are currently running on a system.
- **log_monitor.exe** – Displays the execution history and errors of an application.
- **relink.exe** – Updates existing packages to the latest ThinApp version installed on the system.

- **sbmerge.exe** – Merges runtime changes recorded in the application sandbox with the ThinApp project and updates the captured application.
- **Setup Capture.exe** – Captures and configures applications through a wizard.
- **snapshot.exe** – Compares the preinstallation environment and postinstallation environment during the application capture process.

ThinApp starts this utility during the setup capture process.

- **snapshot.ini** – Stores entries for the virtual registry and virtual file system that ThinApp ignores during the process of capturing an application.

The `snapshot.exe` file references the `snapshot.ini` file. Advanced users might modify the `snapshot.ini` file to ensure that ThinApp does not capture certain entries when creating an application package.

- **template.msi** – Builds the MSI files.

You can customize this template to ensure that the `.msi` files generated by ThinApp adhere to company deployment procedures and standards. For example, you can add registry settings that you want ThinApp to add to client computers as part of the installation.

- **thinreg.exe** – Registers captured applications on a computer.

This registration includes setting up shortcuts and the **Start** menu and setting up file type associations that allow you to open applications.

- **tlink.exe** – Links key modules during the build process of the captured application.
- **vftool.exe** – Compiles the virtual file system during the build process of the captured application.
- **vregtool.exe** – Compiles the virtual registry during the build process of the captured application.

Capturing Applications

You can capture applications to package an application into a virtual environment.

The Setup Capture wizard is the main method to capture applications and set initial application parameters. Advanced users who must capture applications from the command line can use the `snapshot.exe` utility instead of the Setup Capture wizard.

This section includes the following topics:

- [“Phases of the Capture Process”](#) on page 13
- [“Preparing to Capture Applications”](#) on page 13
- [“Capturing Applications with the Setup Capture Wizard”](#) on page 14
- [“Advanced Package Configuration”](#) on page 21
- [“Capturing Internet Explorer 6 on Windows XP”](#) on page 22
- [“Capturing Multiple Application Installers with ThinApp Converter”](#) on page 24

Phases of the Capture Process

Capturing an application involves system scans, application configuration, package configuration, and generation of the virtual application for distribution.

The Setup Capture wizard sets initial parameters for the application. You can customize the full set of parameters outside of the wizard.

Preparing to Capture Applications

Preparing for the capture process involves understanding the needs and dependencies of the application.

For target applications that have dependencies on other applications, libraries, or frameworks, you can capture the dependencies or use the Application Link utility to link separate virtual applications at runtime. For information about the Application Link utility, see [“Application Link Updates”](#) on page 50.

For target applications that require locale formats, such as a specific date format, you can capture them in an environment with the required locale setting. ThinApp runs virtual applications according to the regional and language settings on the capture system rather than the settings on the system that runs the application. Although you can modify the default locale setting by commenting out the `LocaleIdentifier` parameter in the `Package.ini` file and rebuilding the application, you can avoid complications in the capture environment. For information about the `LocaleIdentifier` parameter, see [“LocaleIdentifier”](#) on page 83.

Capturing Applications with the Setup Capture Wizard

The capture process packages an application and sets initial application parameters. If you use a virtual machine, consider taking a snapshot before you run the wizard. A snapshot of the original clean state enables you to revert to the snapshot when you want to capture another application.

This information uses Mozilla Firefox as a key example for application capture.

Create a System Image Before the Application Installation

The Setup Capture wizard starts the capture process by scanning the system to assess the environment and create a baseline system image.

Create a system image before the application installation

- 1 Download the applications to capture.
For example, download `Firefox Setup 2.0.0.3.exe` and copy it to the clean computer you are working with.
- 2 Close any applications, such as virus scans, that might change the file system during the capture process.
- 3 From the desktop, select **Start > Programs > VMware > ThinApp Setup Capture**.
- 4 (Optional) In the **Ready to Prescan** dialog box, click **Advanced Scan Locations** to select the drives and registry hives to scan.

You might want to scan a particular location other than the `C:\` drive if you install applications to a different drive. In rare cases, you might want to avoid scanning a registry hive if you know that the application installer does not modify the registry.

- 5 Click **Prescan** to establish a baseline system image of the hard drive and registry files.

The scanning process takes about 10 seconds for Windows XP.

Rescan the System with the Installed Application

You can install the application to virtualize before the Setup Capture wizard rescans the system and assess changes from the initial system image.

Install the application and rescan the system

- 1 When the **Install Application** page appears, minimize the Setup Capture wizard and install the applications to capture.
For example, double-click `Firefox Setup 2.0.0.3.exe` to install Firefox. If the application needs to restart after the installation, restart the system. The process restarts the Setup Capture wizard.
- 2 (Optional) If you are capturing Internet Explorer, in the **Install Application** page, click **Internet Explorer**, to complete additional steps before installing the browser.

If you are capturing Internet Explorer 6 on Windows XP, see [“Capturing Internet Explorer 6 on Windows XP”](#) on page 22.

For more information about entry points, see [“Defining Entry Points as Shortcuts into the Virtual Environment”](#) on page 15.

- 3 (Optional) Make any necessary configuration changes to comply with your company policies, such as using specific security settings or a particular home page.

If you do not make configuration changes at this time, each user must make changes.

- 4 (Optional) Start the application and respond to any messages for information before you continue with the Setup Capture wizard.

If you do not respond to any messages at this time, each user who uses the application must do so during the initial start.

- 5 Close the application.
- 6 Maximize the Setup Capture wizard, click **Postscan** to proceed with another scan of the computer, and click **OK** to confirm the postscan operation.

ThinApp stores the differences between the first baseline image and this image in a virtual file system and virtual registry.

Defining Entry Points as Shortcuts into the Virtual Environment

Entry points are the executable files that act as shortcuts into the virtual environment and start the virtual application. The entry points you can choose from depend on the executable files that your captured application creates during installation.

For example, if you install Microsoft Office, you can select entry points for Microsoft Word, Microsoft Excel, and other applications that are installed during a Microsoft Office installation. If you install Firefox, you might select `Mozilla Firefox.exe` and `Mozilla Firefox (SafeMode).exe` if users require safe mode access.

During the build process that occurs at the end of the Setup Capture wizard, ThinApp generates one executable file for each selected entry point. If you deploy the application as an MSI file or use the `thinreg.exe` utility, the desktop and **Start** menu shortcuts created on user desktops point to these entry points.

Entry Points for Troubleshooting

ThinApp provides entry points to troubleshoot your environment.

Debugging an application might involve the following entry points:

- `cmd.exe` – Starts a command prompt in a virtual context that enables you to view the virtual file system.
- `regedit.exe` – Starts the registry editor in a virtual context that enables you to view the virtual registry.
- `iexplore.exe` – Starts `iexplore.exe` in a virtual context that enables you to test virtualized ActiveX controls.

Entry points start native executable files in a virtual context. Entry points do not create virtual packages of `cmd.exe`, `regedit.exe`, or `iexplore.exe`.

If you cannot predict the need for debugging or troubleshooting the environment, you can use the `Disabled` parameter in the `Package.ini` file at a later time to activate these entry points.

Set Entry Points

You can designate the executable files that make up the list of entry points. ThinApp installs the executable files during the capture process.

Set entry points in the Setup Capture wizard

- 1 On the **Entry Points** page, select the check boxes for user-accessible entry points.
The wizard displays the executable files that were directly accessible through the desktop or **Start** menu shortcuts.
- 2 (Optional) To debug your environment, select the **Show entry points used for debugging** check box to display the `iexplore.exe`, `regedit.exe`, and `cmd.exe` troubleshooting options.

Manage with VMware Horizon Application Manager

You can use VMware Horizon Application Manager to manage the deployment and entitlement of ThinApp packages. See *Using VMware Horizon Application Manager to Manage the Deployment and Entitlement of ThinApp Packages*, available from the ThinApp download site.

Set User Groups

ThinApp can use Active Directory groups to authorize access to the virtual application. You can restrict access to an application to ensure that users do not pass it to unauthorized users.

Active Directory Domain Services define security groups and distribution groups. ThinApp can only support nested security groups.

Set user groups in the Setup Capture wizard

- 1 On the **Groups** page, limit the user access to the application.
 - a Select **Only the following Active Directory groups**.
 - b Click **Add** to specify Active Directory object and location information.

Option	Description
Object Types	Specifies objects.
Locations	Specifies a location in the forest.
Check Names	Verify object names.
Advanced	Locates user names in the Active Directory forest.
Common Queries (under Advanced)	Searches for groups according to names, descriptions, disabled accounts, passwords, and days since last login.

- 2 (Optional) Change the message that appears for users that ThinApp cannot authorize.

Defining Isolation Modes for the Physical File System

Isolation modes determine the level of read and write access to the native file system outside of the virtual environment. You might adjust isolation mode settings depending on the application and the requirements to protect the physical system from changes.

The selection of isolation modes in the capture process determines the value of the `DirectoryIsolationMode` parameter in the `Package.ini` file. This parameter controls the default isolation mode for the files created by the virtual application except when you specify a different isolation mode in the `##Attributes.ini` file for an individual directory.

The selection of a directory isolation mode does not affect the following areas:

- ThinApp treats write operations to network drives according to the `SandboxNetworkDrives` parameter in the `Package.ini` file. This parameter has a default value that directs write operations to the physical drive. ThinApp treats write operations to removable disks according to the `SandboxRemovableDisk` parameter in the `Package.ini` file. This parameter has a default value that directs write operations to the physical drive.
- If you save documents to the desktop or My Documents folder, ThinApp saves the documents to the physical system. ThinApp sets the isolation mode in the `##Attributes.ini` files in `%Personal%` and `%Desktop%` to `Merged` even when you select `WriteCopy` isolation mode.

Applying Merged Isolation Mode for Modifications Outside the Package

With Merged isolation mode, applications can read and modify elements on the physical file system outside of the virtual package. Some applications rely on reading DLLs and registry information in the local system image.

The advantage of using Merged mode is that documents that users save appear on the physical system in the location that users expect, instead of in the sandbox. The disadvantage is that this mode might clutter the system image. An example of the clutter might be first-execution markers by shareware applications written to random computer locations as part of the licensing process.

When you select Merged isolation, ThinApp completes the following operations:

- Sets the `DirectoryIsolationMode` parameter in the `Package.ini` file to `Merged`.
- Sets up exceptions that apply WriteCopy isolation to the following directories and their subdirectories:
 - `%AppData%`
 - `%Common AppData%`
 - `%Local AppData%`
 - `%Program Files Common%`
 - `%ProgramFilesDir%`
 - `%SystemRoot%`
 - `%SystemSystem%`

ThinApp retains Merged isolation mode for the `%SystemSystem%\spool` subdirectory by creating an exception to the `%SystemSystem%` parent directory isolation mode.

- Between the prescan and postscan capture operations, assigns Full isolation mode to any directories that the application creates during the installation. This process is unrelated to the isolation mode of any new directories that the running virtual application creates.

Merged isolation mode in the Setup Capture wizard has the same effect as Merged isolation mode in the `Package.ini` file, including the directory exceptions that specify WriteCopy isolation mode. The Setup Capture wizard and manual capture process with the `snapshot.exe` utility configure the directory exceptions for you with the `##Attributes.ini` files within the directories.

Applying WriteCopy Isolation Mode to Prevent Modifications Outside of the Package

With WriteCopy isolation mode, ThinApp can intercept write operations and redirect them to the sandbox.

You can use WriteCopy isolation mode for legacy or untrusted applications. Although this mode might make it difficult to find user data files that reside in the sandbox instead of the physical system, this mode is useful for locked down desktops where you want to prevent users from affecting the local file system.

When you select WriteCopy isolation in the Setup Capture wizard, ThinApp completes a number of operations.

- Sets the `DirectoryIsolationMode` parameter in the `Package.ini` file to `WriteCopy`.
- Sets up exceptions that apply Merged isolation to these directories
 - `%Personal%`
 - `%Desktop%`
 - `%SystemSystem%\spool`
- Between the prescan and postscan capture operations, assigns Full isolation mode to any directories that the application creates during the installation. This process is unrelated to the isolation mode of any new directories that the running virtual application creates.

WriteCopy isolation mode in the Setup Capture wizard has the same effect as WriteCopy isolation mode in the `Package.ini` file, including the directory exceptions that specify Merged isolation mode. The Setup Capture wizard and `snapshot.exe` utility configure the directory exceptions for you with the `##Attributes.ini` files within the directories.

Set File System Isolation Modes

The capture process sets the level of read and write access to the physical file system to determine which directories are visible and writable by the virtual application.

For information about Full isolation and registry isolation that are available only outside of the Setup Capture wizard, see [“DirectoryIsolationMode”](#) on page 64 and [“RegistryIsolationMode”](#) on page 65.

Set file system isolation modes in the Setup Capture wizard

On the **Isolation** page, select the isolation mode for the physical file system.

Option	Description
Full write access to non system directories (Merged isolation mode)	Allows the application to read resources on and write to the local machine.
Restricted write access (WriteCopy isolation mode)	Allows the application to read resources on the local machine and to restrict most modifications to the sandbox. ThinApp copies file system changes to the sandbox to ensure that ThinApp only modifies copies of files instead of the actual physical files.

Storing Application Changes in the Sandbox

The sandbox is the directory where all changes that the captured application makes are stored. The sandbox is runtime modification storage and is not a cache. The next time you open the application, those changes are incorporated from the sandbox.

When you delete the sandbox directory, the application reverts to its captured state. You might delete a sandbox when an application has a problem and you want to revert the application back to the working original state.

Customize the Sandbox Location

You can deploy the sandbox to a local user machine, carry it on a mobile USB device, or store it in a network location.

If you deploy the sandbox to a local machine, use the user's profile as the sandbox location. The default location of the sandbox for Firefox might be %AppData%\Thinstall\Mozilla Firefox 3.0. The typical %AppData% location is C:\Documents and Settings\<user_name>\Application Data. The user's profile is the default location because of the write access.

A network location is useful for backing up the sandbox and for users who log in to any computer and keep their application settings. Use the absolute path to the location, such as \\thinapp\sandbox\Firefox. You can select a network location even if an application is installed on a local machine.

A portable device location is useful to keep the sandbox data on the device where the application resides.

Customize the sandbox location in the Setup Capture wizard

On the **Sandbox** page, select the user's profile, application directory, or custom location for the sandbox.

Send Anonymous Statistics to VMware

To improve ThinApp support for applications, VMware uses the capture process to confirm whether to collect anonymous data about deployed ThinApp packages. The data includes the application start time, total running time, and number of runs for the application.

Send anonymous statistics to VMware

On the **Usage Statistics** page, click the **Yes - Send anonymous usage statistics to VMware** option button to confirm the data collection status.

Customize ThinApp Project Settings

A project is the data that the capture process creates. You cannot run or deploy the captured application until you build a package from the project files.

Setting up the project involves determining the inventory name and the project location. The inventory name facilitates internal tracking of the application and determines the default project directory name.

Customize project settings in the Setup Capture wizard

- 1 On the **Project Settings** page, change the inventory name.

Using the `thinreg.exe` utility or deploying the captured application as an MSI file causes the inventory name to appear in the **Add or Remove Programs** dialog box for Windows.

- 2 Change the directory where you want to save the ThinApp project.

If you keep the default directory and capture Firefox 2.0.0.3, the path might appear as `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox (2.0.0.3)`.

Defining Package Settings

A package is the executable file or MSI file with executable files that you use to run or deploy a captured application. You build a package from the project files.

Setting up the package during the capture process involves specifying information about the main virtual application file that serves as the primary data container, MSI generation, and compression.

Defining the Primary Data Container

The primary data container is the main virtual application file that includes the ThinApp runtime and the read-only virtual file system and virtual registry. The primary data container file is a `.exe` or a `.dat` file that resides in the same `/bin` directory with any subordinate application executable files. Entry points reference the information in the primary data container.

To identify the primary data container after you capture an application, check the `ReadOnlyData` parameter in the `Package.ini` file.

Generating MSI Packages in the Capture Process

You can capture an application and deploy it as an MSI Windows installation package. The MSI installation places the application in the `C:\Program Files` directory.

A typical Firefox application does not require an MSI installation. Other applications, such as Microsoft Office, that integrate with application delivery tools, work well as an MSI package. MSI generation requires you to install the MSI on the target device before you can use the application package.

MSI packages automate the process of registering file-type associations, registering desktop and **Start** menu shortcuts, and displaying control panel extensions. If you plan to deploy ThinApp executable files directly on each computer, you can accomplish the same registration by using the `thinreg.exe` utility.

Compressing Packages in the Capture Process

Compressing a package in the capture process decreases the size of an executable package but does not affect MSI packages.

Compression can reduce the on-disk storage requirement by 50 percent but slows the application performance when ThinApp uncompresses initial blocks that start the application. VMware does not recommend compression for test builds because compression increases the build time.

Customize Package Settings

The capture process includes initial settings for the primary data container, MSI packages, and executable package compression.

Customize package settings in the Setup Capture wizard

- 1 On the **Package Settings** page, select the primary data container from the list that is based on your executable file entry points.
 - If the size of the primary container is smaller than 200MB, ThinApp creates a .exe file as the primary container. For a small application such as Firefox, any .exe file can serve as the main data container.
 - If the size of the primary container is larger than 200MB, ThinApp creates a separate .dat file as the primary container because Windows XP and Windows 2000 cannot show shortcut icons for large .exe files. Generating separate small .exe files together with the .dat file fixes the problem.
 - If the size of the primary container is between 200MB and 1.5GB, ThinApp creates the default .dat file unless you select a .exe file to override the default .dat file.
- 2 (Optional) If you select a .exe file to override the default .dat file when the size of the primary container is between 200MB and 1.5GB, ignore the generated warning.

Selecting a .exe file enables all applications to work properly but might prevent the proper display of icons.
- 3 (Optional) If you cannot select a primary data container, type a primary data container name to generate a .dat file.

If you plan to use the Application Sync utility to update a captured application, ThinApp uses the primary data container name during the process. You must use the same name across multiple versions of the application. You might not be able to select the same primary data container name from the list. For example, Microsoft Office 2003 and Microsoft Office 2007 do not have common entry point names.
- 4 (Optional) Select the **Generate MSI package** check box and change the MSI filename.
- 5 (Optional) To create a smaller executable package for locations such as a USB device, select the **Compress virtual package** check box.
- 6 Click **Save**.

Opening Project and Parameter Files

The capture process provides an opportunity to review the project files to update settings before building the executable package or MSI package.

For example, if you capture Firefox 2.0.0.3, you might browse the C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3 directory to update a setting, such as an Active Directory specification, in the Package.ini file that contains the parameters set during the capture process. For information about updating settings, see [“Advanced Package Configuration”](#) on page 21.

The project includes folders, such as %AppData%, that represent file system paths that might change locations when running on different operating systems or computers. Most folders have ##Attributes.ini files that specify the isolation mode at the folder level.

Build Virtual Applications

You can adjust project files and build the application for deployment.

Build virtual applications in the Setup Capture wizard

- 1 (Optional) On the **Ready to Build** page, scan or change the project files.

Option	Description
Edit Package.ini	Modify application parameters for the entire package.
Open project folder	Browse ThinApp project files in Windows Explorer.

- 2 (Optional) To prevent a build from taking place, select the **Skip the build process** check box.
You can build the package at a later time with the `build.bat` file in the virtual application folder. For example, a Firefox 2.0.0.3 path to the `build.bat` file might be `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3\build.bat`.
- 3 Click **Build** to build an executable package or MSI package containing the files you installed during the capture process.
- 4 (Optional) Deselect the **Open folder containing project executables after clicking Finish** check box to view the executable files and MSI files at a later time.
- 5 Click **Finish**.

You can rebuild the package at any time after you click **Finish** to make changes.

Advanced Package Configuration

Advanced users might modify configuration files, such as the `Package.ini` or `##Attributes.ini` files, before building the package during the capture or after the initial build of the package.

Modifying Settings in the Package.ini File

You can modify the `Package.ini` file to update the overall package.

The file resides in the captured application folder. A Firefox 2.0.0.3 path might be `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3\Package.ini`.

The following parameters are a few examples of settings that you might modify:

- `DirectoryIsolationMode` – Sets the isolation mode to `Merged`, `WriteCopy`, or `Full`.
- `PermittedGroups` – Restricts use of an application package to a specific set of Active Directory users.
- `SandboxName` – Identifies the sandbox.
You might keep the name for incremental application updates and change the name for major updates.
- `SandboxPath` – Sets the sandbox location.
- `SandboxNetworkDrives` – Specifies whether to direct write operations on the network share to the sandbox.
- `RequiredAppLinks` – Specifies a list of external ThinApp packages to import to the current package at runtime.
- `OptionalAppLinks` – Specifies a list of external ThinApp packages to import to the current package at runtime.

For information about all `Package.ini` parameters, download a copy of the *Thinapp Package.ini Reference* from the ThinApp download site

Modify the Package.ini File

Use a text editor to modify the `Package.ini` file.

Modify the Package.ini file

- 1 Open the `Package.ini` file located in the captured application folder.

For example, a Firefox 2.0.0.3 path might be `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3\Package.ini`.

- 2 Activate the parameter to edit by removing the semicolon at the beginning of the line.

For example, activate the `RemoveSandboxOnExit` parameter for Firefox.

```
RemoveSandboxOnExit=1
```

- 3 Delete or change the value of the parameter and save the file.

- 4 Double-click the `build.bat` file in the captured application folder to rebuild the application package.

For example, a Firefox 2.0.0.3 path to the `build.bat` file might be `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3\build.bat`.

Modifying Settings in the ##Attributes.ini File

The `##Attributes.ini` file exists in the folder macros of the project folder and applies configuration settings at the directory level. The `Package.ini` file applies settings at the overall application level. You can use the `DirectoryIsolationMode`, `CompressionType`, and `ExcludePattern` parameters in an `##Attributes.ini` file to override the `Package.ini` settings at the directory level.

For example, you can set the isolation mode at the directory or application level to determine which files and registry keys are visible and written by the virtual application you create. The detailed setting in the `##Attributes.ini` file overrides the overall `Package.ini` setting. The `Package.ini` setting determines the isolation mode only when ThinApp does not have `##Attributes.ini` information.

The `##Attributes.ini` file appears in most folders for the captured application. For example, the `Attributes.ini` file might be located in `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3\%AppData%\##Attributes.ini`.

Modify the ##Attributes.ini File

Use a text editor to modify the `##Attributes.ini` file.

Modify the ##Attributes.ini file

- 1 In the `##Attributes.ini` file, uncomment, update, or delete the parameter.
- 2 Double-click the `build.bat` file in the captured application folder to rebuild the application package.

Capturing Internet Explorer 6 on Windows XP

After you use the Setup Capture wizard to capture Internet Explorer 6 running on Windows XP, on a test machine you can use the ThinApp ThinDirect plug-in to redirect Web sites or specific pages to automatically open in a virtual Internet Explorer 6 browser. You can view Web pages that are incompatible in the native version of Internet Explorer in the virtual Internet Explorer 6. A list is maintained that facilitates a redirection process for specified incompatible domains and pages.

You can also install Internet Explorer 6 plug-ins such as Java runtime plug-ins. The plug-ins are treated as any other file during Setup Capture. The plug-ins are embedded in the Internet Explorer 6 capture.

After the ThinDirect plug-in is successfully installed in your native browser, when a user requests a URL that is included in the redirect list, a message appears in the native browser to alert the user that the page is being redirected to a virtual Internet Explorer 6 browser. The virtual browser opens and the requested URL appears.

Requirements for Capturing Internet Explorer 6 on Windows XP

Before you start the Setup Capture wizard the following requirements must be met:

- You must have a clean virtual machine with Windows XP installed.
Ensure that Windows XP includes all the service packs and Microsoft updates, so that Internet Explorer 6 is captured with the latest security fixes from Microsoft.
- ThinApp must be installed on the same machine.

Capture Internet Explorer 6 on Windows XP by Using the Setup Capture Wizard

Capturing Internet Explorer 6 using the Setup Capture wizard is similar to that of capturing other applications. There are two key differences. When you use the Setup Capture wizard to capture Internet Explorer 6 on Windows XP, you define an entry point to Internet Explorer. You also use ThinDirect to specify URLs that will be redirected to the virtualized Internet Explorer 6 browser.

See [“Capturing Applications with the Setup Capture Wizard”](#) on page 14 for a full overview of the standard Setup Capture process.

Run setup capture on a machine running Windows XP with Service Pack 3, and with the .NET framework installed.

Capture Internet Explorer 6 on Windows XP

- 1 Create a system image using the Prescan process of the Setup Capture wizard.
- 2 In the Install Application dialog box, click **Internet Explorer**.
- 3 Select **Include entry point for virtualized Internet Explorer 6 in the virtual package** and click **OK**.
This option captures both the files that changed during setup capture and other required files and registry settings.
- 4 Install any plug-ins for Internet Explorer that you want to be included in the package.
- 5 Rescan the system using the Postscan process of the Setup Capture wizard.
- 6 In the Setup Capture – Entry Points dialog box, select the default, **VirtIE6.exe**.
- 7 Follow the prompts until the Native Browser Redirect dialog box appears.
- 8 Create a list of the Web sites and pages that you want to redirect to the virtual Internet Explorer 6 package.
Each entry must be on a separate line.
 - You can use wildcards, for example `*.example.com`.
 - You can specify a site so that all pages on that site are redirected, for example, `www.example.com`.
 - You can specify a site name followed by a page name, so that the specific page is redirected, for example `javatester.org/version.html`.
- 9 (Optional) When you have saved the package, open the `ThinDirect.txt` file, which contains the entry point to Internet Explorer 6 and the list of redirect addresses, and edit the file.
This file only exists after you create entries in the Native Browser Redirect dialog box.
The redirection list is located in `%appdata%\roaming\VMware\VMware Thinapp\Thindirect`.
- 10 Follow the prompts to build the project.
The `ThinDirect.exe` file is embedded in the package, with the plug-in `ThinDirect.dll` and plug-in launcher `ThinDirectLauncher.exe` files.

Extracting and Registering ThinDirect

After you have built the Internet Explorer 6 package, you need to extract and register the ThinDirect plug-in on the test machine. The ThinDirect plug-in must be installed as part of the virtual package. The plug-in is installed in your native browser during the registration process.

Extract and register ThinDirect

In the console, run the **thinreg /a VirtIE6.exe** command to extract the ThinDirect application, and extract and register the ThinDirect library.

The ThinDirect application is installed in the `Program Files/VMware/VMware ThinApp/ThinDirect` directory.

You can have multiple ThinDirect text files in the ThinDirect directory, if they all have unique names. The ThinDirect plug-in then reads all files.

In addition to individual machine registration, you can register Web page redirects on a individual user basis by omitting the `/a` switch. To achieve individual-user redirects requires that the ThinDirect plug-in be installed as a separate step from an Administrator account. If you do not install the ThinDirect plug-in as a separate step, Thinreg displays an error.

You can push additional Web page redirect to end-user computers by copying files with a specific format to specific individual-machine or individual-user locations.

Capturing Multiple Application Installers with ThinApp Converter

On virtual machines running a Windows operating system, you can use ThinApp Converter to convert multiple application installers into ThinApp packages. After you provide a configuration file with specific settings that the converter accesses, ThinApp Converter runs applications in silent mode. Silent mode means that the process occurs without requiring user input, after initial configuration settings are specified. ThinApp Converter transparently captures installation content, generates ThinApp projects, and build the projects into a ThinApp package in virtual machines you specify in the configuration file. This process is fully automated, from when ThinApp Converter starts to run until the ThinApp package is built.

The ThinApp executable file and the application installers can run on virtual machines.

ThinApp Converter Process

Before you run ThinApp Converter, you must use the `ThinAppConverter.ini` configuration file as a template to specify the virtual machine environment on which the applications to be converted reside, the network share paths, and various other mandatory and optional parameters. You then use the `-f` command line switch to specify the configuration file that you created, which ThinApp Converter will use. For example, `ThinAppConverter.exe -f myConfig.ini`.

ThinApp Converter reads the configuration file to identify which installers are to be converted and the virtual machines on which the conversion is to occur.

ThinApp Converter then powers on each virtual machine and takes a snapshot that is used after the conversion process is complete.

After the snapshot is taken, ThinApp Converter pushes a silent capture agent to virtual machines. The silent capture agent runs transparently on the virtual machines, capturing the application installation process in a similar way to that of the Setup Capture wizard when a single application is being captured. The silent capture agent performs the following actions:

- Runs a ThinApp prescan
- Installs an application from the network share specified in the configuration file
- Runs a postscan
- Generates a ThinApp project on the network share specified in the configuration file
- Performs project post-processing tasks

- Builds the ThinApp project on the network share into a package

The silent capture agent then returns control to the ThinApp Converter, which reverts the virtual machines to their precapture state, using their original snapshot.

The process is then repeated for the next application installation process that needs to be converted. When multiple virtual machines are specified, the capture agent runs on the machines simultaneously. As a virtual machine becomes available, it is once again used for converting the next application

ThinApp Converter Limitations

- Not all application installation processes support silent installation mode. ThinApp Converter does not support automatic capture for an installation process that does not support silent installation.
- The installer directory name must not contain the equals symbol (=).

System Requirements for Running ThinApp Converter

ThinApp Converter requires one of the following virtual machine environments:

- VMware ESX Server 4.0, or later
- VMware vCenter Server 4.0, or later
- VMware Workstation 7.0, or later

The virtual machines that are used in the conversion process must have the following items installed:

- Windows XP with Service Pack 3, Windows Vista, or Windows 7
- The latest version of VMware Tools

ThinApp Converter includes a private copy of the VMware VIX API library. If a more recent version of the library already exists on the host machine, ThinApp Converter tries to use the newest version.

VMware recommends that you use Windows 2003 or Windows 2008 as a file server for network share. The file server needs to have sufficient system resources to handle a large quantity of file operations. Do not use the host machine that runs the ThinApp Converter executable file as the file server for the network share.

When using a VMware Workstation environment, ensure that the network settings are in bridged mode.

Preparing the Configuration File for ThinApp Converter

A sample configuration file, `ThinAppConverter.ini`, is included in the ThinApp installation. The file is generally located in `C:\Program Files\VMware\VMware ThinApp`.

Modify or create a copy of this file to suit your requirements. Use UTF-8 encoding when you specify parameter values.

The `ThinAppConverter.ini` configuration file includes the following section headings:

- `[HostEnvironment]` contains virtual machine hosting parameters.
- `[VirtualMachineN]` contains virtual machine-specific parameters.
- `[Settings]` contains parameters that provide global control of the capture process.
- `[AppSettings:AppName]` contains optional application-specific parameters.

HostEnvironment

The `HostEnvironment` section of the configuration file contains the connection parameters for connecting to VMware ESX Server, VMware vCenter Server, or VMware Workstation on a local machine.

`[HostEnvironment]` parameters are mandatory.

- You can only specify a single endpoint at a time in the configuration file. For example, if you plan to use a single VMware ESX Server, you can have `ThinAppConverter.exe` directly connect to that server.

- You cannot specify more than one ESX server. To use more than one ESX server, configure `ThinAppConverter.exe` to connect to VMware vCenter Server, which manages multiple ESX servers.
- You can use a locally installed VMware Workstation.

VirtualMachineHost

The name of the virtual machine to which ThinApp Converter is to connect.

- To connect a single VMware ESX Server, use the IP address or host name of the ESX server.
- To connect to VMware vCenter Server, use the IP address or name of the vCenter server.
- To connect to a local VMware Workstation instance, use **localhost**.
- For VMware ESX Server or VMware vCenter Server, if you are not using standard HTTPS with port 443, you can specify the entire URL.

Examples

The following example shows a virtual machine specified by ESX server hostname.

```
[HostEnvironment]
VirtualMachineHost=MyEsx.vmware.com
```

The following example shows a virtual machine specified by IP address.

```
[HostEnvironment]
VirtualMachineHost=10.13.11.23
```

The following example shows a local machine specified as **localhost**.

```
[HostEnvironment]
VirtualMachineHost=localhost
```

HostLoginUserName

The login user name for the host machine.

Use the same login user name for connecting to the server as you use for logging in to the VMware vSphere Client. You must have sufficient privileges to turn on and off virtual machines, take virtual machine snapshots, and so on.

You can use UPN format when you specify a user name for vCenter. For example, `user@domain.com`.

`HostLoginUserName` is ignored when logging into VMware Workstation.

HostLoginPassword or HostLoginPasswordBase64

The login password for the host machine. You have the following options when you specify passwords:

- You can enter clear text.
- You can specify a base64 encoded password for the `HostLoginPasswordBase64` parameter. Specifying an encoded password does not increase security. You need to protect the actual INI file.

All passwords are handled in the same way.

HostLoginPasswordPrompt

Specifies that the user be prompted to enter a password.

If you do not want to store the vSphere Server password in the configuration file, specify the value as **true**. When set to **true**, a prompt always appears, even if a `HostLoginPassword` is specified in the configuration file.

Example

The following example shows a typical host environment specification. The virtual machine name is specified as the ESX server hostname. A password has been specified, however the user will still be prompted to enter as password, as specified in `HostLoginPasswordPrompt`.

```
[HostEnvironment]
VirtualMachineHost=MyEsx.vmware.com
```

```
HostLoginUserName=root
HostLoginPassword=secret
HostLoginPasswordPrompt=true
```

VirtualMachineN

The `VirtualMachineN` section of the configuration file contains a list of the Windows-based virtual machines that will be utilized in the conversion process.

Create a `VirtualMachineX` section for each virtual machine that you want to include, and specify their parameters. *X* is 1, and subsequent virtual machine sections are numbered sequentially.

[`VirtualMachineM`] parameters are mandatory.

VmxPath

Specify the configuration path of the virtual machine.

For ESX Server or vCenter Server, you can identify the virtual machine configuration file path using the vSphere Client.

Identify the virtual machine configuration path using the vSphere Client

- 1 Right-click the virtual machine and select **Edit Settings**.
- 2 Click the **Options** tab, and copy the string from the **Virtual Machine Configuration File** field.
- 3 Use this string as the virtual machine configuration file path.

For Workstation, specify the entire file path on the host on which the VMX configuration file resides. For example, `C:\MyVMs\Windows XP\Windows XP.vmx`. Do not place the path in quotation marks, even if the path contains a space.

UserName

A valid user name for the virtual machine guest operating system. The user must have administrator privileges for the virtual machine guest operating system.

You can use UPN format when you specify a user name. For example, `user@domain.com`.

Password or PasswordBase64

A valid password for the virtual machine guest operating system. You have the following options when you specify passwords:

- You can enter clear text.
- You can specify a base64 encoded password for the `PasswordBase64` parameter.

Specifying an encoded password does not increase security strength. You need to protect the actual INI file.

All passwords are handled in the same way.

If the `Password` setting is not used, the password for the guest is assumed to be blank. Most Windows virtual machines do not support automation with empty passwords, so you should specify a guest password.

PasswordPrompt

Specifies that the user be prompted to enter a password.

If you do not want to store the virtual machine password in the configuration file, specify the value as `true`. When set to `true`, a prompt always appears, even if a password is specified in the configuration file.

Examples

Following is an example for an ESX server-based environment. A password has been specified and, as `PasswordPrompt` is set to `false`, the user will not be prompted to enter a password.

```
[VirtualMachine1]
VmxPath=[Storage] WinXP_Converter/WinXP_Converter.vmx
```

```

UserName=administrator
Password=secret
PasswordPrompt=false

```

Following is an example for a VMware Workstation-based virtual machine. On virtual machine 1, PasswordPrompt has been set to true. The user will be prompted for a password even though a password has been specified in the configuration.

```

[VirtualMachine1]
VmxPath=C:\MyVMs\Windows XP\Windows XP.vmx
UserName=administrator
Password=secret
PasswordPrompt=true
[VirtualMachine2]
VmxPath=C:\MyVMs\Windows 7\Windows 7.vmx
UserName=adminuser@mydomain.com
Password=
PasswordPrompt=true

```

NOTE Do not place the path in quotation marks, even if the path contains a space.

Settings

The Settings section of the configuration file contains the parameters for the application installation directory and ThinApp project output directory, in the form of UNC. It also contains several parameters controlling the conversion process behavior.

ThinApp Converter only requires read-only permissions for the network share that contains the application installers. It requires read/write permissions for the network share that contains the ThinApp projects.

If input and output directories are on the same file server, you must use the same user account to connect them.

InputUncPath

Specify the network share UNC path for the application installers. For example: \\fileserver\sharename, or \\fileserver\sharename\dirname.

InputMountUserName

Specify the user name used for connecting to that network share. UPN format can be used when you specify a domain user, for example user@domain.com

InputMountPassword or InputMountPasswordBase64

Specify the password for connecting to the network share. You have the following options when you specify passwords:

- You can enter clear text.
- You can specify a base64 encoded password for the PasswordBase64 parameter.

InputMountPasswordPrompt

Specifies that the user be prompted to enter a password.

If you do not want to store the network share password in the configuration file, specify the value as true. When set to true, a prompt always appears, even if a password is specified in the configuration file.

OutputUncPath

Specify the network share UNC path to the location of the generated ThinApp projects.

For example: \\fileserver\sharename, or \\fileserver\sharename\dirname

OutputMountUserName

Specify the user name used for connecting to the OutputUncPath network share. UPN format can be used to specify a domain user, for example, user@domain.com.

OutputMountPassword or OutputMountPasswordBase64

Specify the password for connecting to the network share. You have the following options when you specify passwords:

- You can enter clear text.
- You can specify a base64 encoded password for the `PasswordBase64` parameter.

`OutputMountPasswordPrompt`

Specifies that the user be prompted to enter a password.

If you do not want to store the network share password in the configuration file, specify the value as `true`. When set to `true`, a prompt always appears, even if a password is specified in the configuration file.

Example

Following is an example of network share specifications. The user for the application installation directory has only read permissions. For both the input and output network shares, a prompt will display, requiring a user to enter a password.

```
[Settings]
InputUncPath=\\AppInstallerServer\AppInstallers\ThinAppMigration
InputMountUserName=readonlyUser
InputMountPassword=secret
InputMountPasswordPrompt=true
OutputUncPath=\\DeploymentServer\ThinAppProj
OutputMountUserName=readwriteUser
OutputMountPassword=secret
OutputMountPasswordPrompt=true
```

ThinApp Converter Logic for Detecting the Application Installation Processes

For the application installer's network share, ThinApp Converter examines all subdirectories under the specified UNC path recursively, including their subdirectories. For each subdirectory, it determines which command to run for silent application installation using the following logic:

- 1 Attempts to find a value for `InstallationCommand` in the `[AppSettings:AppName]` section of the configuration file. If successful ThinApp Converter uses that value.
- 2 Attempts to find a file named `install.cmd` or `install.bat`. If successful, ThinApp Converter runs that file.
- 3 If ThinApp Converter finds a single `.cmd` or `.bat` file, it runs that file.
- 4 If ThinApp Converter finds a single `.exe` file, it runs that file.
- 5 If ThinApp Converter finds a single `.mst` file, it runs that file and adds the necessary silent installation switches.
- 6 If ThinApp Converter finds a single `.msi` file, it runs that file and adds the necessary silent installation switches.

If none of the steps enable ThinApp Converter to find a correct installation command, the subdirectory is skipped. A warning is logged in the log file.

You must remove all network connections to the file server reference in the INI file from the host on which you run ThinApp Converter, to prevent conflict between user credentials.

`PackageIniOverrideFile`

Specify the file path to the global `Package.ini` override file.

This optional parameter enables you to specify a global override file for `Package.ini` that is generated for each ThinApp project. The values in the override file are merged into `Package.ini` in the ThinApp project that is generated for each application.

Global overrides are useful when you have a global policy setting, for example, `PermittedGroup` in `Package.ini`.

A Package.ini override file is formatted like a standard Windows INI file. You can add INI parameters and values that are relevant to the Package.ini file.

The path is relative to the application installer's network share. Using the example for specifying the network shares for the application installers and ThinApp projects, if you specify `PackageIniOverrideFile=override.ini`, ThinApp Converter will try to find the file under `\\AppInstallerServer\AppInstaller`. You can provide a more explicit value by using predefined variables. For more information, see [“Predefined Environment Variables”](#) on page 31.

You can specify a `Package.ini` file for each application. This process is described as part of the `[AppSettings:AppName]` section.

ExclusionList

Specify a comma- or semicolon-separated list of application directories that ThinApp will skip when searching for application installers.

The list is case insensitive.

You can specify wildcards for DOS-style file names. For example, `Microsoft*. ?` and `*` are supported.

Example

Following is an example of an exclusion specification using a wildcard.

```
[Settings]
ExclusionList=App?.old;FireFox1.0
```

ProjectPostProcessingCommand

Specify the file path to the project post processing command.

The file path is relative to the application installer's network share. Using the example for specifying the network shares for the application installers and ThinApp projects, if you specify `ProjectPostProcessingCommand=addscript.bat`, ThinApp Converter will try to find the file under `\\AppInstallerServer\AppInstaller`. You can provide a more explicit value by using predefined variables. For more information, see [“Predefined Environment Variables”](#) on page 31.

StopOnError

Specify whether ThinApp Converter should stop converting an application if it encounters an error, or continue with the other applications. The default value is `false`.

BuildAfterCapture

Specify whether the ThinApp Converter should build the ThinApp Projects into packages following capture.

The default value is `true`.

DetectIdle

Specify whether ThinApp Converter should try to detect if an application installer is stalled, for example when the application is waiting for user input on the guest virtual machine because incorrect silent installation switches were specified.

The default value is `true`. ThinApp Converter might not be able to detect all situations in which the installer is idle.

InstallerTimeout

Specify how long ThinApp Converter should wait for an application installer to finish before it quits.

By default, the value is 7200 seconds.

AppSettings:AppName

This optional section provides parameters that you can use to add settings that are specific to an application. *AppName* is the actual name of the subdirectory that contains the application installer. These parameters can be added to each AppSettings section. In most circumstances, you will not need to configure this section.

InstallationCommand

Specify how ThinApp Converter should start the application installer. If there is no value, ThinApp Converter attempts to select one installation command using the logic described in [“ThinApp Converter Logic for Detecting the Application Installation Processes”](#) on page 29.

PackageIniOverrideFile

The Package.ini override file that is applied to a single application installer. When this parameter has a value, the global override file is processed first, followed by this application-specific override file.

The file path is relative to the application installer subdirectory. Using the example at the bottom of this section, if you specify `PackageIniOverrideFile=override.ini`, ThinApp Converter will try to find the file under `\\AppInstallerServer\\AppInstaller\\Adobe`. You can provide a more explicit value by using predefined variables. For more information, see [“Predefined Environment Variables”](#) on page 31.

ProjectPostProcessingCommand

Specify the project post processing command for the specific application.

When this parameter has a value, the global override file is processed first, followed by this application-specific post processing command.

Example

Following is an example of how to apply an application-specific override during post processing.

```
[AppSettings:Adobe]
InstallationCommand=AdbeRdr920_en-US.exe /sAll
PackageIniOverrideFile=override.ini
[AppSettings:TextPad]
InstallationCommand=silent_install.bat
ProjectPostProcessingCommand=%AppInstallerDir%\addscript.bat
```

Predefined Environment Variables

The values for `PackageIniOverrideFile` (global and per application), `ProjectPostProcessingCommand` (global and per application), and `InstallationCommand` can contain environment variables. ThinApp Converter expands the value before using it.

ThinApp Converter adds these variables as predefined environment variables:

- `%AppInstallersRootDir%` - The UNC path of the application installers that is specified in `InputUncPath` in the `[Settings]` section.
- `%AppInstallerDir%` - The subdirectory under `%AppInstallersRootDir%` for the relevant application.
- `%ThinAppProjectsRootDir%` - The UNC path for the generated ThinApp projects that is specified in `OutputUncPath` in the `[Settings]` section.
- `%ThinAppProjectDir%` - The subdirectory under `%ThinAppProjectsRootDir%` for the relevant application.

Example

Following is an example of how predefined variables can be used in the `PackageIniOverrideFile`, `ProjectPostProcessingCommand`, and `InstallationCommand` parameters.

```
[Settings]
PackageIniOverrideFile=%AppInstallersRootDir%\AppSyncSettings.ini
;will resolve to \\AppInstallerServer\\AppInstaller\\AppSyncSettings.ini
[AppSettings:Adobe]
InstallationCommand=AdbeRdr920_en-US.exe /sAll
PackageIniOverrideFile=%AppInstallerDir%\override.ini
;will resolve to \\AppInstallerServer\\AppInstaller\\Adobe\\AppSyncSettings.ini
```


Deploying Applications

Deploying captured applications involves working with deployment tools, the `thinreg.exe` utility, MSI files, and Active Directory.

This information includes the following topics:

- [“ThinApp Deployment Options”](#) on page 33
- [“Establishing File Type Associations with the thinreg.exe Utility”](#) on page 34
- [“Building an MSI Database”](#) on page 37
- [“Controlling Application Access with Active Directory”](#) on page 39
- [“Starting and Stopping Virtual Services”](#) on page 40
- [“Using ThinApp Packages Streamed from the Network”](#) on page 41
- [“Using Captured Applications with Other System Components”](#) on page 43
- [“Sample Isolation Mode Configuration Depending on Deployment Context”](#) on page 45

ThinApp Deployment Options

You can deploy captured applications with deployment tools, in a VMware View™ environment, on a network share, or as basic executable files.

Deploying ThinApp with Deployment Tools

Medium and large enterprises often use major deployment tools, such as Symantec, BMC, and SMS tools. ThinApp works with all major deployment tools.

When you use any of these tools, you can create MSI files for the captured applications and follow the same process you use to deploy native MSI files. See deployment instructions from the tool vendors. For information about MSI files, see [“Building an MSI Database”](#) on page 37.

Deploying ThinApp in the VMware View Environment

You can use VMware View to distribute ThinApp packages.

The workflow for deploying packages might involve the following tasks:

- Creating executable files for the captured applications.
- Storing the executable files on a network share.

- Creating a login script that queries applications entitled to the user and runs the `thinreg.exe` utility with the option that registers the applications on the local machine. Login scripts are useful for nonpersistent desktops. See [“Establishing File Type Associations with the thinreg.exe Utility”](#) on page 34.
- Controlling user access to fileshares. IT administrators might control access by organizing network shares based on function and associating permissions with network shares based on those functional boundaries.

Deploying ThinApp on Network Shares

Small and medium enterprises tend to use a network share. You can create executable files for the captured application and store them on a network share. Each time you deploy a new application or an update to an existing package, you can notify client users to run the `thinreg.exe` utility with an appropriate option.

IT administrators can control user access to fileshares by organizing network shares based on function and associating permissions with network shares based on those functional boundaries.

The differences between the network share option and the VMware View option are that the network share option assumes a mix of physical and virtual (persistent) desktops and involves users running the `thinreg.exe` utility directly instead of relying on login scripts.

Deploying ThinApp Using Executable Files

You can use a basic deployment option with executable files when disk use is limited.

You can create executable files for the captured applications, copy them from a central repository, and run the `thinreg.exe` utility manually to register file type associations, desktop shortcuts, and the application package on the system.

Establishing File Type Associations with the thinreg.exe Utility

If you create executable files instead of MSI files during the capture process, you must run the `thinreg.exe` utility to open files, such as a `.doc` document or an `.html` page. For example, if you click a URL in an email message, ThinApp must be set to start Firefox. You do not have to run the `thinreg.exe` utility for MSI files because MSI files start the utility during the application installation.

The `thinreg.exe` utility creates the **Start** menu and desktop shortcuts, sets up file type associations, adds deinstallation information to the system control panel, and unregisters previously registered packages. The utility enables you to see the control panel extensions for applications, such as Quicktime or the mail control panel applet for Microsoft Outlook 2007. When you right-click a file, such as a `.doc` file, the `thinreg.exe` utility enables you to see the same menu options for a `.doc` file in a native environment.

If an application runs SMTP or HTTP protocols, such as an email link on a Web page that needs to open Microsoft Outlook 2007, the `thinreg.exe` utility starts available virtual applications that can handle those protocols. If virtual applications are not available, the `thinreg.exe` utility starts native applications that can handle those protocols.

The default location of the utility is `C:\Program Files\VMware\VMware ThinApp`.

Application Sync Effect on the thinreg.exe Utility

The Application Sync utility affects the `thinreg.exe` utility during the update process.

If you add, modify, or remove executable files, the `thinreg.exe` utility reregisters the file type associations, shortcuts, and icons.

If you install protocols, MIME types, control panel applets, and templates other than executable files, the `thinreg.exe` utility reregisters these elements.

Run the thinreg.exe Utility

This example of running the `thinreg.exe` utility provides some sample commands.

The package name in the `thinreg.exe` commands can appear in the following ways:

- `C:\<absolute_path_to_.exe>`
- Relative path to `.exe` file
- `\\<server>\<share>\<path_to_.exe>`

As a variation, you can use a wildcard specification, such as `*.exe`.

If the path or filename contains spaces, enclose the path in double quotation marks. The following command shows the use of double quotation marks.

```
thinreg.exe "\\DEPLOYSERVER\ThinApps\Microsoft Office Word 2007.exe"
```

For information about `thinreg.exe` parameters, see [“Optional thinreg.exe Parameters”](#) on page 35.

Run the thinreg.exe utility

- 1 Determine the executable files that ThinApp must register with the local environment.
- 2 From the command line, type the `thinreg.exe` command.

```
thinreg.exe [<optional_parameters>] [<package1.exe>][<package2.exe>][<packages_by_wildcard>]
```

If the server name is `DEPLOYSERVER` and the share is `ThinApps`, use the following example to register Microsoft Word for the logged-in user.

```
ThinReg.exe "\\DEPLOYSERVER\ThinApps\Microsoft Office 2007 Word.exe"
```

Use the following example to register all Microsoft Office applications in the specified directory for the logged-in user.

```
ThinReg.exe "\\DEPLOYSERVER\ThinApps\Microsoft Office *.exe"
```

Optional thinreg.exe Parameters

The `thinreg.exe` utility monitors the `PermittedGroups` setting in the `Package.ini` file, registering and unregistering packages as needed. When the `thinreg.exe` utility registers a package for the current user, the utility creates only the shortcuts and file type associations that the current user is authorized for in the `PermittedGroups` setting. If this setting does not exist, the current user is authorized for all executable files.

When the `thinreg.exe` utility registers a package for all users with the `/allusers` parameter, ThinApp creates all shortcuts and file type associations regardless of the `PermittedGroups` setting. When you double-click a shortcut icon that you are not authorized for, you cannot run the application.

If the package name you want to register or unregister contains spaces, you must enclose it in double quotation marks.

For information about the `PermittedGroups` setting and support for Active Directory groups, see [“PermittedGroups”](#) on page 69.

[Table 3-1](#) lists optional parameters for the `thinreg.exe` utility. Any command that uses the `/a` parameter requires administrator rights.

Table 3-1. Optional thinreg.exe Parameters

Parameter	Purpose	Sample Usage
/a, /allusers	Registers a package for all users. If an unauthorized user attempts to run the application, a message informs the user that he or she cannot run the application.	thinreg.exe /a "\\<server>\<share>\Microsoft Office 2007 Word.exe"
/q, /quiet	Prevents the display of an error message for an unrecognized command-line parameter.	thinreg.exe /q <unknown_option>
/u, /unregister, /uninstall	Unregisters a package. This command removes the software from the Add/Remove Programs control panel applet.	Unregister Microsoft Word for the current user. thinreg.exe /u "\\<server>\<share>\Microsoft Office 2007 Word.exe" Unregister all Microsoft Office applications for the current user and remove the Add/Remove Programs entry. thinreg.exe /u "\\server\share\Microsoft Office *.exe" If a user registers the package with the /a parameter, you must use the /a parameter when unregistering the package. thinreg.exe /u /a *.exe
/r, /reregister	Reregisters a package. Under typical circumstances, the thinreg.exe utility can detect whether a package is already registered and skips it. The /r option forces the thinreg.exe utility to reregister the package.	thinreg.exe /r "\\<server>\<share>\Microsoft Office 2007 Word.exe" If a user registers the package with the /a parameter, you must use the /a when reregistering the package. thinreg.exe /r /a *.exe
/k, /keepunauthorized, /keep	Prevents the removal of registration information even if you are no longer authorized to access an application package. Without this option, the thinreg.exe utility removes the registration information for that package if it detects you are no longer authorized to access the package. ThinApp stores authorization information in the PermittedGroups parameter of the Package.ini file.	thinreg.exe /k "\\<server>\<share>\Microsoft Office 2007 Word.exe"
/noarp	Prevents the creation of an entry in the Add/Remove Programs control panel applet.	thinreg.exe /q /noarp "\\<server>\<share>\Microsoft Office 2007 Word.exe"
/norelaunch	Starts the thinreg.exe utility on Microsoft Vista without elevated privileges. Standard users can start the utility without a user account control (UAC) pop-up window. When the thinreg.exe utility detects a need for more privileges, such as the privileges required for the /allusers parameter, the utility restarts itself as a privileged process and generates a UAC pop-up window. The /norelaunch option blocks this restart process and causes the registration to fail.	thinreg.exe /q /norelaunch "\\<server>\<share>\Microsoft Office 2007 Word.exe"

Building an MSI Database

If you do not create MSI files during the capture process, you can still create these files after building an application. An MSI database is useful for delivering captured applications through traditional desktop management systems to remote locations and automatically creating shortcuts and file type associations. Basic Active Directory group policies provide ways to distribute and start MSI packages.

ThinApp creates an MSI database that contains captured executable files, installer logic, and the `thinreg.exe` utility.

Customizing MSI Files with Package.ini Parameters

You can customize the behavior of MSI files by modifying `Package.ini` parameters and rebuilding the application package.

The following parameters can affect MSI configuration:

- The `MSIInstallDirectory` parameter sets the installation directory for the package.
For example, include `MSIInstallDirectory=C:\Program Files\` in the `Package.ini` file.
- The `MSIDefaultInstallAllUsers` parameter sets the installation of the package for individual users. ThinApp installs the package in the `%AppData%` user directory.
For example, include `MSIDefaultInstallAllUsers=0` in the `Package.ini` file.
For more information about this parameter, see [“Specifying a Database Installation for Individual Users and Machines”](#) on page 38.
- The `MSIFileName` parameter names the package.
For example, include `MSIFilename=Firefox30.msi` in the `Package.ini` file.
- The `MSIRequireElevatedPrivileges` parameter indicates whether an installer needs elevated privileges for deployment on Microsoft Vista. Installations for individual users do not usually need elevated privileges but per-machine installations require such privileges.
For example, include `MSIRequireElevatedPrivileges=1` in the `Package.ini` file.
- The `MSIProductCode` parameter makes it easier to install a new version of the application. An MSI database contains a product code and an upgrade code. When you update a package, keep the original value of the `MSIUpgradeCode` parameter.

If the parameter value of the new version is the same as the value of the old version, the installation prompts you to remove the old version. If the values for the parameter are different, the installation uninstalls the old version and installs the new version.

VMware recommends that you avoid specifying an `MSIProductCode` value and allow ThinApp to generate a different product code for each build.

Regardless of the parameter values specified at build time, you can override the settings at deployment time. See [“Force MSI Deployments for Each User or Each Machine”](#) on page 38. For more information about MSI parameters, see [“Configuring MSI Files”](#) on page 92.

Modify the Package.ini File to Create MSI Files

For more information about MSI parameters, see [“Customizing MSI Files with Package.ini Parameters”](#) on page 37 and [“Configuring MSI Files”](#) on page 92.

Before you can modify MSI parameters, you must add an entry for the `MSIFileName` parameter to generate MSI files.

Modify the MSI parameters

- 1 In the `Package.ini` file, type the MSI filename.

MSIFilename=<filename>.msi

For example, the filename for Firefox might be `Mozilla Firefox 2.0.0.3.msi`.

- 2 (Optional) Update other MSI parameters.
- 3 Double-click the `build.bat` file in the captured application folder to rebuild the application package.

Specifying a Database Installation for Individual Users and Machines

You can modify the installation of the MSI database for users and machines.

ThinApp installs the MSI database across all machines. You can change the default installation with the following parameter values:

- To create a database installation for individual users, use a value of 0 for the `MSIDefaultInstallAllUsers` parameter in the `Package.ini` file. This value creates `msiexec` parameters for each user.
- To allow administrators to create a database installation for all users on a machine, or to allow an individual user without administrator rights to create an installation only for that user, use a value of 2 for the `MSIDefaultInstallAllUsers` parameter. Administrators belong to the Administrators Active Directory group.

For more information about the `MSIDefaultInstallAllUsers` parameter, see [“MSIDefaultInstallAllUsers”](#) on page 92.

Force MSI Deployments for Each User or Each Machine

Regardless of the parameter values specified at build time, you can override MSI settings at deployment time.

For example, if you created the database with a value of 1 for the `MSIDefaultInstallAllUsers` parameter, you can still force individual user deployments for Firefox 3.0 with the `msiexec /i Firefox30.msi ALLUSERS=""` command.

If you use the `ALLUSERS=""` argument for the `msiexec` command, ThinApp extracts the captured executable files to the `%AppData%` user directory.

Force MSI deployments for individual users or for all users on a machine

- (Optional) From the command line, type the `msiexec /i <database>.msi ALLUSERS=""` command to force deployments for individual users.
- (Optional) From the command line, type the `msiexec /i <database>.msi ALLUSERS=1` command to force deployments for all users on a machine.

Override the MSI Installation Directory

You can use the `msiexec` command to override the default MSI installation directory.

When ThinApp performs an individual machine MSI deployment, the default installation directory is the localized equivalent of `%ProgramFilesDir%\<inventory_name>` (VMware ThinApp). If you install a Firefox package for each machine, the package resides in `%ProgramFilesDir%\Mozilla Firefox` (VMware ThinApp).

When ThinApp performs an MSI deployment for individual users, the default installation directory is `%AppData%\<inventory_name>` (VMware ThinApp).

In both cases, you can override the installation directory by passing an `INSTALLDIR` property to the `msiexec` command.

Override the MSI installation directory

From the command line, type the `msiexec /i <database>.msi INSTALLDIR=C:\<my_directory>\<my_package>` command.

Deploying MSI Files on Microsoft Vista

When you deploy MSI files on Vista, you must indicate whether an installer needs elevated privileges. Typical individual user installations do not require elevated privileges but individual machine installations require such privileges.

ThinApp provides the `MSIRequireElevatedPrivileges` parameter in the `Package.ini` file that specifies the need for elevated privileges when the value is set to 1. Specifying a value of 1 for this parameter or forcing an individual user installation from the command line can generate UAC prompts. Specifying a value of 0 for this parameter prevents UAC prompts but the deployment fails for machine-wide installations.

Controlling Application Access with Active Directory

You can control access to applications using Active Directory groups.

When you build a package, ThinApp converts Active Directory group names into Security Identifier (SID) values. A SID is a small binary value that uniquely identifies an object. SID values are not unique for a few groups, such as the administrator group. Because ThinApp stores SID values in packages for future validation, the following considerations apply to Active Directory use:

- You must be connected to your Active Directory domain during the build process and the groups you specify must exist. ThinApp looks up the SID value during the build.
- If you delete a group and re-create it, the SID might change. In this case, rebuild the package to authenticate against the new group.
- When users are offline, ThinApp can authenticate them using cached credentials. If the users can log into their machines, authentication still works. Use a group policy to set the period when cached credentials are valid.
- 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, group policy, and security settings stored in Active Directory. You might log out before Active Directory credentials are recached.
- Certain groups, such as the Administrators group and 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.
- Active Directory Domain Services define security groups and distribution groups. If you use nested groups, ThinApp can only support nested security groups.

Package.ini Entries for Active Directory Access Control

ThinApp provides the `PermittedGroups` parameter in the `Package.ini` file to control Active Directory access.

When you start a captured application, the `PermittedGroups` parameter checks whether a user is a member of a specified Active Directory group. If the user is not a member of the Active Directory group, ThinApp does not start the application. For information about restricting packages to Active Directory groups, see [“PermittedGroups”](#) on page 69.

In the following `Package.ini` entry, App1 and App2 inherit `PermittedGroups` values.

```
[BuildOptions]
PermittedGroups=Administrators;OfficeUsers
[App1.exe]
    ...
    ..
[App2.exe]
    ...
    ...
```

In the following entry, only users belonging to the App1users group can use the App1.exe file, and members of the Everyone group can use the App2.exe file. The default message for denied users changes for App1.

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

Starting and Stopping Virtual Services

When you capture and deploy a package that contains a Windows service, such as the SQL Server service, any user can run the package and start and stop the service. Unlike native applications, virtual applications do not require administrator rights for these operations.

Automatic Startup for Virtual Services

You can install a virtual service as a physical service, so that it starts when the physical machine is started. The virtual service remains in its ThinApp project package, but is registered on the physical machine and controlled using the natively installed service management tools.

After you package your service, for example Apache Server, you register it on the physical machine, using the ThinReg.exe application. The service is created as a native service, using information from the virtual registry. The service is available to all users using the virtual application. The service is not user specific.

The process is composed of the following tasks:

- Capturing the service by using ThinApp
- Registering the service on the physical machine by using ThinReg

Create a virtual service for automatic startup

- 1 On a clean local machine, use ThinApp to capture the service.
- 2 After the postscan process is complete, in the Setup Capture - Ready to Build dialog, click **Edit Package.ini**.

The Package.ini file opens in a text editor.

- 3 Search for the **services** entry.

The entry is followed by the name of the service that you captured.

By default, the entry is commented out.

- 4 Remove the semicolon (;) from the start of the line.
- 5 Save the Package.ini file.
- 6 Build the ThinApp project.

You can now register your virtual service so that it can be managed by using the native services management tools.

Register the virtual service on a physical machine

- 1 Run the ThinReg.exe application.
- 2 At the command line, type **C:\Program Files\VMware\VMware ThinApp\ThinReg /a *.exe**.

You must use **/a** to register services. If you run ThinApp without this option, the service is not registered.

You can change the path, if required for your system.

- 3 From the **Start** menu, select **Programs > Administrative Tools > Services**.

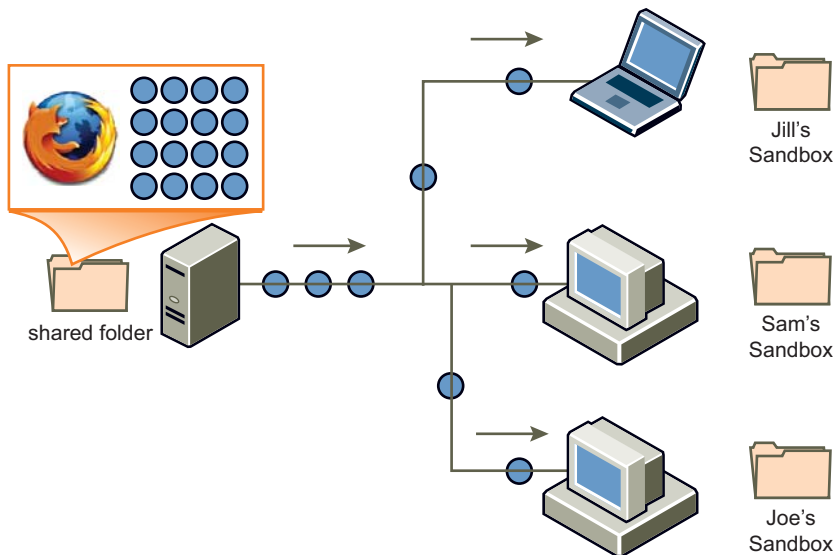
Your virtual service application appears in the list of services.

You can manage the service in the same way as any natively installed service.

Using ThinApp Packages Streamed from the Network

Any network storage device can serve as a streaming server for hundreds or thousands of client computers. See [Figure 3-1](#).

Figure 3-1. Data Block Streaming over a Network Share



On the end-user desktop, you can create shortcuts that point to the centrally hosted executable file packages. When the user clicks the shortcut, the application begins streaming to the client computer. During the initial streaming startup process, the ThinApp status bar informs the user of the progress.

How ThinApp Application Streaming Works

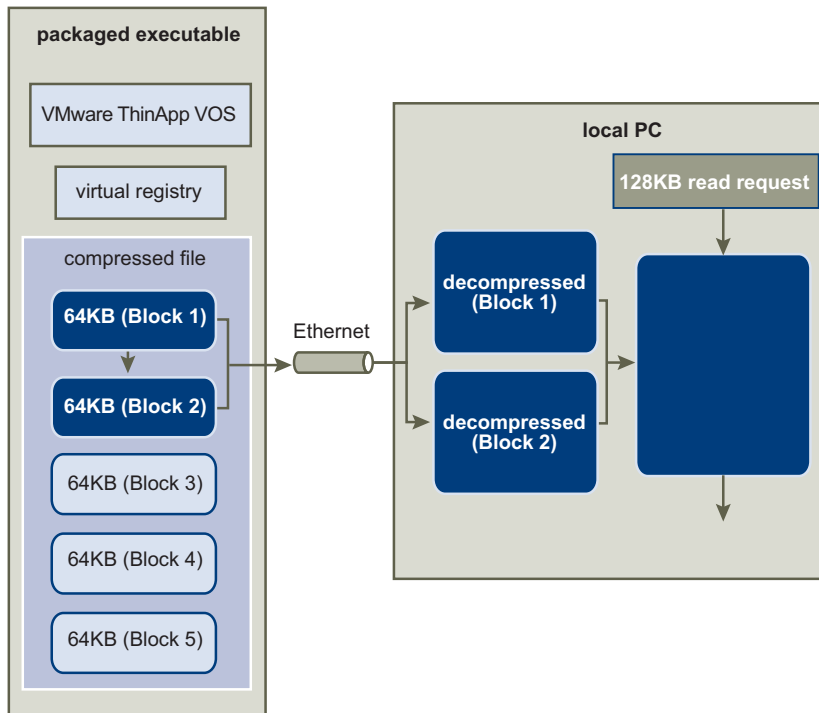
When you place compressed ThinApp executable files on a network share or USB flash drive, the contents from the executable file stream to client computers in a block-based fashion. As an application requests specific parts of data files, ThinApp reads this information in the compressed format over the network using standard Windows file sharing protocol. For a view of the process, see [Figure 3-2](#).

After a client computer receives data, ThinApp decompresses the data directly to memory. Because ThinApp does not write data to the disk, the process is fast. A large package does not necessarily take a long time to load over the network and the package size does not affect the startup time of an application. If you add an extra 20GB file to a package that is not in use at runtime, the package loads at the same speed. If the application opens and reads 32KB of data from the 20GB file, ThinApp only requests 32KB of data.

The ThinApp runtime client is a small part of the executable file package. When ThinApp loads the runtime client, it sets up the environment and starts the target executable file. The target executable file accesses other parts of the application stored in the virtual operating system. The runtime client intercepts such requests and serves them by loading DLLs from the virtual operating system.

The load time of the runtime client across a network is a few milliseconds. After ThinApp loads the runtime client to memory on the client computer, the end-user computer calculates which blocks of data are required from the server and reads them based on application activity.

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 computer runs low on memory, Windows discards some of its disk cache and provides the memory resource to other applications.

Figure 3-2. Application Streaming

Requirements and Recommendations for Streaming Packages

ThinApp does not require specific 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 for each application. Microsoft Office requires that only a fraction of the package contents stream before an application can run.

VMware recommends that you use ThinApp streaming in a LAN-based environment with a minimum of 100MB networks. For WAN and Internet deployments that involve frequent or unexpected disconnections, VMware recommends one of the following solutions:

- Use a URL to deploy the applications.
- Use a desktop deployment solution to push the package to the background. Allow the application 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 client machines located at each branch office.

Security Recommendations for Streaming Packages

VMware recommends that you make a central shared directory for the package read-only. Users can read the package contents but not change the executable file contents. When a package streams from a shared location, ThinApp stores application changes in the user sandbox. The default sandbox location is %AppData%\Thinstall\<application_name>. You can configure the sandbox location at runtime or at package time.

A common configuration is to place the user sandbox on another central storage device. The user can use any computer and keep individual application settings at a central share. When packages stream from a central share, they remain locked until all users exit the application.

Stream ThinApp Packages from the Network

Users can access packaged applications through the network.

Stream packages from the network

- 1 Place the ThinApp package in a location accessible to client computers.
- 2 Send a link to users to run the application directly.

Using Captured Applications with Other System Components

Captured applications can interact with other components installed on the desktop.

Performing Paste Operations

Review the following paste operations and limitations with ThinApp:

- **Pasting content from system installed applications to captured applications** – This paste operation is unlimited. The virtual application can receive any standard clipboard formats, such as text, graphics, and HTML. The virtual application can receive OLE objects.
- **Pasting from captured applications to system applications** – ThinApp converts OLE objects created in virtual applications to system native objects when you paste them into native applications.

Accessing Printers

A captured application has access to any printer installed on the computer that it is running on. Captured applications and applications installed on the physical system have the same printing ability.

You cannot use ThinApp to virtualize printer drivers. You must manually install printer drivers on a computer.

Accessing Drivers

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

Sometimes, an application without an associated driver might function with some limitations. For example, Adobe Acrobat installs a printer driver that enables applications system wide to render PDF files using a print mechanism. When you use a captured version of Adobe Acrobat, you can use it to load, edit, and save PDF files without the printer driver installation. Other applications do not detect a new printer driver unless the driver is installed.

Accessing the Local Disk, the 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 computer.

Review the default configuration options described in [Table 3-2](#).

Table 3-2. Default Configuration Options

Component	Description
Hard disk	An example of a hard disk is C:\. Isolation modes selected during the capture process affect access. Users can write to their Desktop and My Documents folders. Other modifications that the application makes go into the user sandbox. The default location of the sandbox is in the Application Data directory.
Removable disk	By default, any user who has access rights can read or write to any location on a removable disk.

Table 3-2. Default Configuration Options (Continued)

Component	Description
Network mapped drives	By default, any user who has access rights can read or write to any location on a network mapped disk.
UNC network paths	By default, any user who has access rights can read or write to any location on a UNC network path.

Accessing 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, ThinApp saves all registry modifications from captured applications in an isolated sandbox and the system remains unchanged.

Accessing Networking and Sockets

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

Using Shared Memory and 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 invisible to a captured application.

Using COM, DCOM, and Out-of-Process COM Components

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.

Starting Services

Captured applications can start and run system-installed services and virtual services. System services run in the virtual environment that controls the modifications that the services can make.

Using File Type Associations

Captured applications can run system-installed applications by using file type associations. You can add file type associations to the local computer registry to point to captured executable files for individual users and machines.

Sample Isolation Mode Configuration Depending on Deployment Context

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

You can adjust isolation modes to resolve the problems in [Table 3-3](#).

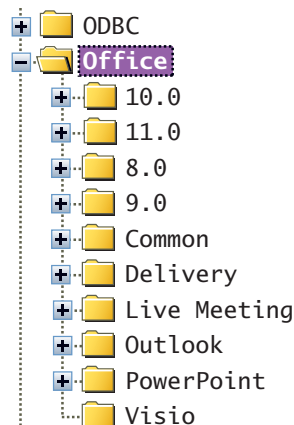
Table 3-3. Sample Problems and Solutions That Use Isolation Modes

Problem	Solution
An application fails to run because previous or future versions exist simultaneously or fail to uninstall properly.	<p>Use the Full isolation mode.</p> <p>ThinApp hides host computer files and registry keys from the application when the host computer files are located in the same directories and subkeys that the application installer creates.</p> <p>For directories and subkeys that have Full isolation, the applications only detect virtual files and subkeys. Any system values that exist in the same location are invisible to the application.</p>
An application fails because users did not design or test it for a multiuser environment. The application fails to modify files and keys without affecting other users.	<p>Use the WriteCopy isolation mode.</p> <p>ThinApp makes copies of registry keys and files that the application writes and performs all the modifications in a user-specific sandbox.</p> <p>For directories and subkeys that have WriteCopy isolation, the application recognizes the host computer files and virtual files. All write operations convert host computer files into virtual files in the sandbox.</p>
An application fails because it has write permission to global locations and is not designed for a locked-down desktop environment found in a corporate setting or on Windows Vista.	<p>Use the WriteCopy isolation mode.</p> <p>ThinApp makes copies of registry keys and files that the application writes and performs all the modifications in a user-specific sandbox.</p> <p>For directories and subkeys that have WriteCopy isolation, the application recognizes the host computer files and virtual files. All write operations convert host computer files into virtual files in the sandbox.</p>

View of Isolation Mode Effect on the Windows Registry

[Figure 3-3](#) shows a section of the Windows registry for a computer that has older Microsoft Office applications installed. Microsoft Office 2003 creates the HKEY_LOCAL_MACHINE\Software\Microsoft\Office\11.0 registry subtree.

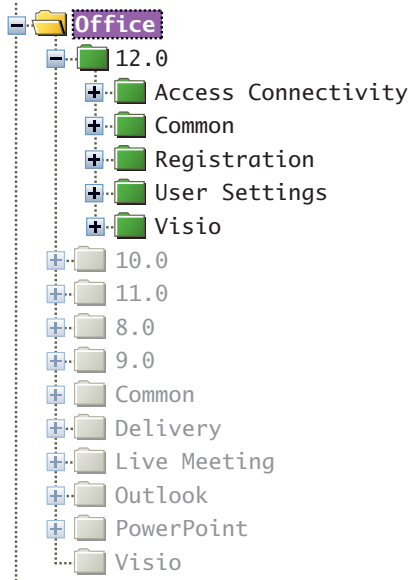
Figure 3-3. Windows Registry as Seen by Windows Regedit



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

Figure 3-4 shows the registry from the perspective of the captured Microsoft Visio 2007.

Figure 3-4. Windows Registry as Seen by the Captured Microsoft Visio 2007



Updating and Linking Applications

You can update virtual applications with different utilities depending on the extent of change and dependencies on other applications.

This information includes the following topics:

- [“Application Updates That the End User Triggers”](#) on page 47
- [“Application Updates That the Administrator Triggers”](#) on page 54
- [“Automatic Application Updates”](#) on page 56
- [“Upgrading Running Applications on a Network Share”](#) on page 57
- [“Sandbox Considerations for Upgraded Applications”](#) on page 58
- [“Updating the ThinApp Version of Packages”](#) on page 58

Application Updates That the End User Triggers

ThinApp provides the Application Sync and Application Link utilities to update applications with new versions or new components. The Application Sync utility updates an entire application package. The Application Link utility keeps shared components or dependent applications in separate packages.

Application Sync Updates

The Application Sync utility keeps deployed virtual applications up to date. When an application starts with this utility enabled, the application queries a Web server to determine if an updated version of the executable file is available. If an update is available, the differences between the existing package and the new package are downloaded and used to construct an updated version of the package. The updated package is used for future launches.

The Application Sync utility is useful for major configuration updates to the application. For example, you might update Firefox to the next major version. Remote users or users who are not connected to the corporate network can make use of the Application Sync utility by embedding update settings within the package and using any Web server to store the updated version of the package.

Using Application Sync in a Managed or Unmanaged Environment

If you use virtual applications that update automatically in a managed computer environment, do not use the Application Sync utility because it might clash with other update capabilities.

If an automatic update feature updates an application, the update exists in the sandbox. If the Application Sync utility attempts to update the application after an automatic application update, the version update stored in the sandbox take precedence over the files contained in the Application Sync version. The order of precedence for updating files is the files in the sandbox, the virtual operating system, and the physical machine.

If you have an unmanaged environment that does not update applications automatically, use the Application Sync utility to update applications.

Update Firefox 2.0.0.3 to Firefox 3 with Application Sync

This example shows the Application Sync update process for Firefox.

The update process involves modifying the `Package.ini` file. The `AppSyncURL` parameter requires a URL path. ThinApp supports HTTP, HTTPS, and file protocols. For information about all Application Sync parameters, see [“Configuring Application Updates with Application Sync”](#) on page 89.

Update Firefox 2.0.0.3 to Firefox 3

- 1 Capture Firefox 2.0.0.3 and Firefox 3 into separate packages.
- 2 Verify that the primary data container name is the same for both packages.

The primary data container, determined during the setup capture process, is the file that contains the virtual file system and virtual registry. If you have a Firefox 2.0.0.3 package that has `Mozilla Firefox 2.0.0.3.exe` as the name of the primary data container, and you have a Firefox 3 package that has `Mozilla Firefox 3.dat` as the name of the primary data container, change the name in the `Shortcut` parameter to a common name. For example, you can use `Firefox.exe` as a name.

- 3 Modify the `Package.ini` file in each package.

- a Open the `Package.ini` file located in the captured application folder.

For example, a Firefox 2.0.0.3 path to the `Package.ini` file might be `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3\Package.ini`.

- b Uncomment the Application Sync parameters you want to edit by removing the semicolon at the beginning of the line.

You must uncomment the `AppSyncURL` parameter to enable the utility.

- c Change the value of the parameters and save the file.

For example, you can copy an executable file of the latest Firefox version to a mapped network drive and type a path to that location as the value of the `AppSyncURL` parameter. If `Z:` is the mapped drive and `Firefox` is the name of the directory that stores the executable file, a sample path is `file:///Z:/Firefox/Firefox.exe`.

Make sure that the `AppSyncURL` path is the same in both `Package.ini` files and points to the updated version.

- 4 In the captured application folder, double-click the `build.bat` file to rebuild the application package.

For example, a Firefox 2.0.0.3 path to the `build.bat` file might be `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3\build.bat`.

- 5 To update Firefox 2.0.0.3 to Firefox 3, start the executable file, such as `Mozilla Firefox 2.0.0.3.exe`, in the `\bin` directory.

When you start the application before the expiration time set in the `AppSyncExpirePeriod` parameter of the `Package.ini` file, ThinApp downloads the update in the background as you work with the application. The next time you start the application, you can see the updated version.

When you start the application after the package expires, ThinApp downloads the update in the foreground and prevents you from working with the application. When the download is ready, ThinApp restarts the application with the new version.

Fix an Incorrect Update with Application Sync

If you have multiple Application Sync download updates, such as multiple Microsoft Office updates, and a certain update has an adverse affect or needs to be withdrawn, you can address the problem.

Fix an incorrect update

Place the correct update on the server that ThinApp can access.

The update is applied the next time the application is started on a client machine.

Application Sync Effect on Entry Point Executable Files

The Application Sync utility updates entry point executable files. For example, assume you deploy a Microsoft Office 2007 package that does not include Microsoft PowerPoint. The `Microsoft Office PowerPoint 2007.exe` entry point does not exist for the original package. If you rebuild the Microsoft Office 2007 package to include Microsoft PowerPoint, and you use the Application Sync utility to update client machines, the end users can access an entry point executable file for Microsoft PowerPoint.

Updating `thinreg.exe` Registrations with Application Sync

If you register virtual applications on the system using `thinreg.exe` and update applications with the Application Sync utility, you can update registrations by placing a copy of `thinreg.exe`, located in `C:\Program Files\VMware\VMware ThinApp`, alongside the updated package on the server.

Maintaining the Primary Data Container Name with Application Sync

The Application Sync utility requires that the name of the primary data container, the file that stores virtual files and registry information, is the same for the old and new versions of an application. For example, you cannot have an old version with `Microsoft Office Excel 2003.exe` as the primary data container name while the new version has `Microsoft Office 2007.dat` as the primary data container name. To verify the name of the primary data container, see the `ReadOnlyData` parameter in the `Package.ini` file. For more information about the primary data container, see [“Defining Entry Points as Shortcuts into the Virtual Environment”](#) on page 15.

Completing the Application Sync Process When Applications Create Child Processes

When a captured application creates child processes, ThinApp cannot complete the Application Sync process.

For example, you might create Microsoft Office 2003 and Microsoft Office 2007 packages, modify the `AppSyncURL` parameter in the `Package.ini` file for both packages, and copy the Microsoft Office 2007 package to a Web server and the Microsoft Office 2003 package to a client machine.

If you start the Microsoft Office 2003 package before the expiration time set in the `AppSyncExpirePeriod` parameter of the `Package.ini` file, ThinApp can download the update in the background as you work with the application but unable to show the updated version the next time you start the application. If you start the application after the package expires, ThinApp is unable to download the update in the foreground and restart the application when the download is ready.

Microsoft Office 2003 and Microsoft Office 2007 are examples of applications that create child processes. ThinApp cannot complete Application Sync updates until all child processes stop. You can perform one of the following tasks to resolve the issue:

- Log out and log in to the machine to stop the child processes.
- Create a script to end the child processes.
For example, you can create a script to end the `ctfmon.exe` and `mdm.exe` child processes associated with Microsoft Office 2003 and Microsoft Office 2007.
- Prevent the startup of the child process, such as the `ctfmon.exe` process associated with Microsoft Office and Internet Explorer applications.

Prevent the Startup of the `ctfmon.exe` Process for Microsoft Office and Internet Explorer

Preventing the startup of the `ctfmon.exe` process requires knowledge of the ThinApp sandbox and `sbmerge.exe` utility. For information about the `sbmerge.exe` utility, see [“Updating Applications with Runtime Changes”](#) on page 55.

Prevent the startup of the ctfmon.exe process

- 1 If you did not activate the `cmd.exe` entry point during the capture process, set the `Disabled` parameter for the `cmd.exe` entry in the `Package.ini` file to 0 and rebuild the package with the `build.bat` utility.
This generates an executable file for the `cmd.exe` entry point in the `/bin` directory.
- 2 Copy the `/bin` directory in the captured application directory to a clean virtual machine or delete the sandbox for the Microsoft Office package.
- 3 Double-click the `cmd.exe` entry point.
- 4 In the Windows command processor, run the `INTL.CPL` command.
- 5 In the **Languages** tab of the **Regional and Languages** dialog box, click **Details**.
- 6 In the **Advanced** tab of the **Text Services and Input Languages** dialog box, select the **Turn off advanced text services** check box.
- 7 Click **OK** in all the open dialog boxes and leave the Windows command processor open.
- 8 Unregister the `MSIMTF.dll` and `MSCTF.dll` files with the `REGSVR32.EXE/U <DLL_file>` command.
See knowledge base article 282599 in the Microsoft Web site.
- 9 Close the Windows command processor.
- 10 If the virtual machine does not reside on the same machine where ThinApp is installed, copy the sandbox from the package to the packaging system.
The default sandbox location is `%APPDATA%\Thinstall`.
- 11 From the standard command prompt on the packaging system, use the `sbmerge.exe` utility to merge the updated sandbox with the package.
A sample command is `SBMERGE APPLY -ProjectDir "C:\Program Files\VMware\VMware ThinApp\Captures\Microsoft Office Professional 2007" -SandboxDir "%APPDATA%\Thinstall\Microsoft Office Pro 2007"`.
- 12 Rebuild the package and test the package on a clean virtual machine to confirm that the `ctfmon.exe` process no longer exists.

Application Link Updates

The Application Link utility connects dependent applications at runtime. You can package, deploy, and update component pieces separately rather than capture all components in the same package.

ThinApp can link up to 250 packages at a time. Each package can be any size.

The Application Link utility is useful for the following objects:

- **Large shared libraries and frameworks** – Link runtime components, such as .NET, JRE, or ODBC drivers, with dependent applications.

For example, you can link .NET to an application even if the local machine for the application prevents the installation of .NET or already has a different version of .NET.

If you have multiple applications that require .NET, you can save space and make a single .NET package and point the multiple applications to the .NET package. When you update .NET with a security fix, you can update a single package rather than multiple packages.

- **Add-on components and plug-ins** – Package and deploy application-specific components and plug-ins separately from the base application.

For example, you might separate Adobe Flash Player or Adobe Reader from a base Firefox application and link the components.

You can deploy a single Microsoft Office package to all users and deploy individual add-on components for each user.

If you capture Microsoft Office and try to access a PDF attachment in the virtual Microsoft Outlook environment, you can set up Microsoft Office to detect a linked Adobe Reader package on the network when Adobe Reader is not available within the immediate virtual or physical environment.

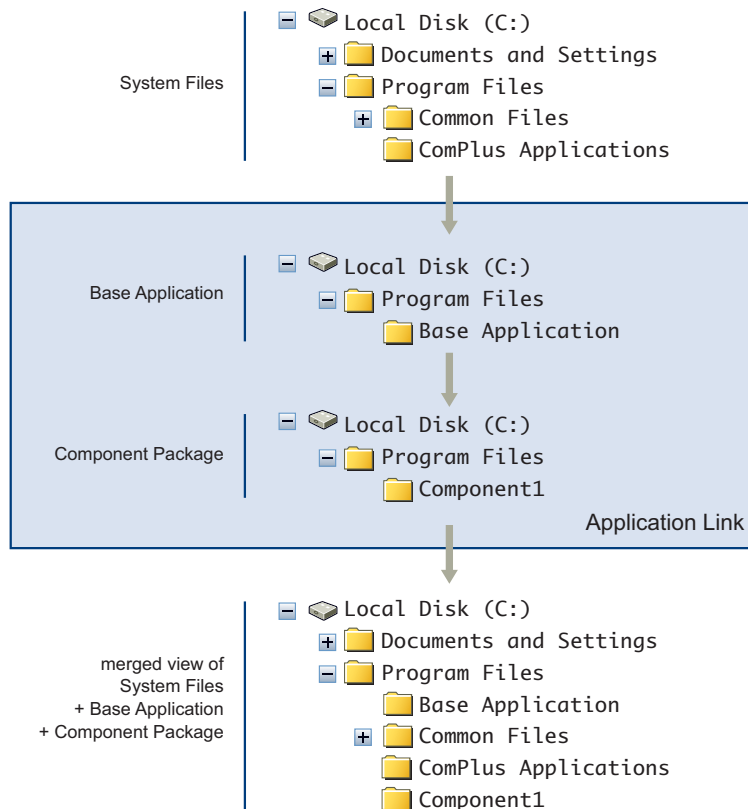
- **Hot fixes and service packs** – Link updates to an application and roll back to a previous version if users experience significant issues with the new version. You can deploy minor patches to applications as a single file and reduce the need for rollbacks.

The Application Link utility provides bandwidth savings. For example, if you have Microsoft Office 2007 Service Pack 1 and you want to update to Service Pack 2 without Application Link, you would transfer 1.5Gb of data per computer with the deployment of a new Office 2007 Service Pack 2 package. The Application Link utility transfers just the updates and not the whole package to the computers.

View of the Application using Application Link

Figure 4-1 shows the running application with a merged view of the system, the base application, and all linked components. Files, registry keys, services, COM objects, and environment variables from dependency packages are visible to the base application.

Figure 4-1. View of the System, Base Application, and Linked Components Using Application Link



Link a Base Application to the Microsoft .NET Framework

Review this sample workflow to link a base application, `MyApp.exe`, to a separate package that contains the Microsoft .NET 2.0 Framework. Make sure that the base application capture process does not include the Microsoft .NET 2.0 Framework. For information about the process of capturing an application, see [Chapter 2, "Capturing Applications,"](#) on page 13.

For information about required and optional Application Link parameters and formats in the `Package.ini` file, see ["Configuring Dependent Applications with Application Link"](#) on page 87.

Link an application to Microsoft .NET

- 1 Capture the installation of the .NET 2.0 Framework.
During the capture process, you must select at least one user-accessible entry point.
- 2 Rename the .exe file that ThinApp produces to a .dat file.
This renaming prevents users from accidentally running the application.
The name of the .dat file you select does not matter because users do not run the file directly.
For example, use `dotnet.dat`.
- 3 Save the .NET project to `C:\Captures\dotnet`.
- 4 Capture the base application by using the same physical system or virtual machine with the .NET framework already installed.
- 5 Save the project to `C:\Captures\MyApp`.
- 6 Open the `Package.ini` file in the captured application folder for the base application.
- 7 Enable the `RequiredAppLinks` parameter for the base application by adding the following line after the `[BuildOptions]` entry.
`RequiredAppLinks=dotnet.dat`
Application Link parameters must reference the primary data container of the application you want to link to. You cannot reference shortcut .exe files because these files do not contain any applications, files, or registry keys.
- 8 Rebuild the .NET 2.0 and base application packages.
 - a Double-click the `build.bat` file in `C:\Captures\MyApp`.
 - b Double-click the `build.bat` file in `C:\Captures\dotnet`.
 Running these batch files builds separate ThinApp packages.
- 9 Deploy the applications to an end-user desktop in `C:\Program Files\MyApp`.
 - a Copy `C:\Captures\MyApp\bin\MyApp.exe` to
`\\<end_user_desktop>\<Program_Files_share>\MyApp\MyApp.exe`.
 - b Copy `C:\Captures\dotnet\bin\cmd.exe` to
`\\<end_user_desktop>\<Program_Files_share>\MyApp\dotnet.dat`.

Set Up Nested Links with Application Link

ThinApp supports nested links with the Application Link utility. For example, if Microsoft Office links to a service pack, and the service pack links to a hot fix, ThinApp supports all these dependencies.

This procedure refers to AppA, which requires AppB; and AppB, which requires AppC. Assume the following folder layout for the procedure:

- `C:\AppFolder\AppA\AppA.exe`
- `C:\AppFolder\AppB\AppB.exe`
- `C:\AppFolder\AppC\AppC.exe`

For information about setting up required and optional Application Link parameters in this procedure, see [“Configuring Dependent Applications with Application Link”](#) on page 87.

Set up nested links

- 1 Capture Application A.
- 2 In the `Package.ini` file, specify Application B as a required or optional application link.
For example, add `RequiredLinks=\AppFolder\AppB\AppB.exe` to the file.

- 3 Capture Application B.
 - 4 In the `Package.ini` file for Application B, specify Application C as a required or optional application link. For example, add `RequiredLinks=\AppFolder\AppC\AppC.exe` to the file.
 - 5 Capture Application C.
- If you start Application A, it can access the files and registry keys of Application B and Application B can access the files and registry keys of Application C.

Affecting Isolation Modes with Application Link

ThinApp loads an Application Link layer during application startup and merges registry entries and file system directories. If ThinApp finds a registry subkey or file system directory that did not previously exist in the main package or layer that is already merged, ThinApp uses the isolation mode specified in the layer being loaded. If the registry subkey or file system directory exists in the main package and a layer that is already merged, ThinApp uses the most restrictive isolation mode specified in any of the layers or main package. The order of most restrictive to least restrictive isolation modes is Full, WriteCopy, and Merged.

PermittedGroups Effect on Linked Packages

If you link two applications and you specify a value for the `PermittedGroups` parameter, the user account used for starting the application must be a member of at least one of the Active Directory groups for this parameter in the `Package.ini` files of both applications. For information about the `PermittedGroups` parameter, see [“Configuring Permissions”](#) on page 68.

Sandbox Changes for Standalone and Linked Packages

Sandbox changes from linked packages are not visible to the base executable file. For example, you can install Acrobat Reader as a standalone virtual package and as a linked package to the base Firefox application. When you start Acrobat Reader as a standalone application by running the virtual package and you change the preferences, ThinApp stores the changes in the sandbox for Acrobat Reader. When you start Firefox, Firefox cannot detect those changes because Firefox has its own sandbox. Opening a `.pdf` file with Firefox does not reflect the preference changes that exist in the standalone Acrobat Reader application.

Import Order for Linked Packages

ThinApp imports linked applications according to the order of applications in the `RequiredAppLinks` or `OptionalAppLinks` parameter. If either parameter specifies a wildcard character that triggers the import of more than one file, alphabetical order determines which package is imported first.

The `OptionalAppLinks` parameter might appear as `OptionalAppLinks=a.exe;b.exe;plugins*.exe`.

Using `a.exe` and `b.exe` as sample executable files, ThinApp imports linked packages in the order described in [Table 4-1](#).

Table 4-1. Imported Linked Packages

Import Order	Linked Package
1	Base application
2	<code>a.exe</code>
3	<code>b.exe</code>
4	Plug-ins loaded in alphabetical order
5	Nested plug-ins for <code>a.exe</code>
6	Nested plug-ins for <code>b.exe</code>
7	Nested plug-ins for the first set of plug-ins in this list

For information about nested links, see [“Set Up Nested Links with Application Link”](#) on page 52.

File and Registry Collisions in Linked Packages

If the base application and a dependent package linked to the base application contain file or registry entries at the same location, a collision occurs. When this happens, the order of import operations determines which package has priority. The last package imported has priority in such cases and the file or registry contents from that package are visible to the running applications.

VBScript Collisions in Linked Packages

VBScript name collisions might prevent scripts in other imported packages from running. If you link packages with Application Link and those packages have scripts with the same name, ThinApp places the VBScripts from the linked packages into a single pool. For scripts with the same name, ThinApp runs the script from the last imported package and disregards the other script.

For example, a base package might contain the `a.vbs` and `b.vbs` files and a dependent package might contain the `b.vbs` and `c.vbs` files. Because a filename collision exists between the `b.vbs` files, the VBScript in the last imported package specified in a `RequiredAppLinks` or `OptionalAppLinks` parameter overrides any previously imported scripts with the same name. In this case, ThinApp condenses the pool of four `.vbs` files into a single pool with the `a.vbs` file from the base package and `b.vbs` and `c.vbs` files from the dependent package.

VBScript Function Order in Linked Packages

In a pool of VBScripts for packages linked with Application Link, functions in the main bodies of the scripts run in alphabetical order of the script names. ThinApp callback functions in the scripts run in reverse alphabetical order of the script names in the pool.

Storing Multiple Versions of a Linked Application in the Same Directory

If the directory holds a linked package, and you add an updated version of the linked package in the same directory, the Application Link utility detects and uses the updated version.

Using Application Sync for a Base Application and Linked Packages

If you use Application Link to link packages to a base package, and you start the base package, Application Sync can update only the base package. For example, if you build a Microsoft Office 2007 package with Application Sync entries in the `Package.ini` file, build an Adobe Reader package with Application Sync entries in the `Package.ini` file, use Application Link to link the two packages, and start Microsoft Office 2007, Application Sync only updates Microsoft Office 2007. You can update both Microsoft Office 2007 and Adobe Reader by starting each application separately.

If you do not update all the applications and link a base application to an expired plug-in, the base application can still load and use the plug-in.

Application Updates That the Administrator Triggers

ThinApp provides the `AppSync.exe` and `sbmerge.exe` utilities for administrators.

The `AppSync.exe` utility forces an Application Sync update on a client machine.

The `sbmerge.exe` utility make incremental updates to applications. For example, an administrator might use the utility to incorporate a plug-in for Firefox or to change the home page of a Web site to point to a different default site.

Forcing an Application Sync Update on Client Machines

You can use the AppSync command to force an Application Sync update on a client machine. You might want to update a package stored in a location where standard users do not have write access. In this situation, you cannot use Application Sync parameters to check for updates when an application starts because users do not have the required rights to update the package. You can schedule a daily AppSync.exe run under an account with sufficient privileges. The Application Sync parameters, such as AppSyncUpdateFrequency, in the Package.ini file do not affect the AppSync command.

To force an Application Sync update, use the AppSync <Application_Sync_URL> <executable_file_path> command. The value of the URL is the same as the Application Sync URL in the Package.ini file and the executable file path is the path to the executable file that requires the update.

Updating Applications with Runtime Changes

The sbmerge.exe utility merges runtime changes recorded in the application sandbox back into a ThinApp project. A typical workflow for this utility involves the following tasks:

- Capturing an application.
- Building the application with the build.bat file.
- Running a captured application and customizing the settings and virtual environment. ThinApp stores the changes in the sandbox.
- Running the sbmerge.exe utility to merge registry and file system changes from the sandbox into the ThinApp project.
- Rebuilding the captured application with the build.bat file
- Deploying the updated application.

Merge Sandbox Changes with Firefox

This procedure for the sbmerge.exe utility uses Firefox 2.0.0.3 as an example of a captured application.

Merge sandbox changes with Firefox 2.0.0.3

- 1 Capture Firefox 2.0.0.3.
- 2 Double-click the build.bat file in the captured application folder to rebuild the application package.

For example, a Firefox 2.0.0.3 path to the build.bat file might be C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3\build.bat.
- 3 Create a Thinstall directory in the bin directory for the sandbox location.
- 4 Start Firefox and make a change to the settings.

For example, change the home page.
- 5 From the command line, navigate to the directory where the ThinApp project folder resides.

For example, navigate to C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3.
- 6 From the command line, type the "C:\Program Files\VMware\VMware ThinApp\sbmerge" Print command.

ThinApp prints the changes that affected the sandbox folder when using the captured application.
- 7 From the command line, type the "C:\Program Files\VMware\VMware ThinApp\sbmerge" Apply command.

ThinApp empties the Thinstall folder and merges the sandbox changes with the application.

sbmerge.exe Commands

The `sbmerge.exe Print` command displays sandbox changes and does not make modifications to the sandbox or original project.

The `sbmerge.exe Apply` command merges changes from the sandbox with the original project. This command updates the project registry and file system to reflect changes and deletes the sandbox directory.

Usage

```
"C:\Program Files\VMware\VMware ThinApp\sbmerge" Print [<optional_parameters>]
"C:\Program Files\VMware\VMware ThinApp\sbmerge" Apply [<optional_parameters>]
```

Optional Parameters

The optional `sbmerge.exe` parameters specify project and sandbox paths and block progress messages and merging of sandbox files.

Table 4-2. Optional `sbmerge.exe` Parameters

Parameter	Description
<code>-ProjectDir <project_path></code>	If you start the <code>sbmerge.exe</code> command from a location other than the application project folder, use the absolute or relative path to the project directory using the <code>-ProjectDir <project_path></code> parameter. A sample command is <code>"C:\Program Files\VMware\VMware ThinApp\sbmerge" Print -ProjectDir "C:\<project_folder_path>"</code> .
<code>-SandboxDir <sandbox_path></code>	When you start a captured application, it searches for the sandbox in a particular order. See “Search Order for the Sandbox” on page 59. If you use a custom location for the sandbox, use the <code>-SandboxDir <sandbox_path></code> parameter to specify the location.
<code>-Quiet</code>	Blocks the printing of progress messages.
<code>-Exclude <excluded_file>.ini</code>	Prevents the merging of specific files or registry entries from the sandbox. You can specify a <code>.ini</code> file to determine the content for exclusion. This file contains separate sections to specify files, such as the <code>FileSystemIgnoreList</code> and the <code>RegistryIgnoreList</code> . The <code>sbmerge.exe</code> utility uses the <code>snapshot.ini</code> file in the ThinApp installation folder by default to exclude certain content from the merge process. This option enables you to specify another <code>.ini</code> file to ensure the additional exclusion of content.

Automatic Application Updates

If an application can update automatically, its update mechanism functions with ThinApp. If the application downloads the update and runs an installer or patching program, this activity occurs inside the virtual environment and ThinApp stores the changes from the update software in the sandbox. When the application restarts, it uses the version of the executable file in the sandbox and not the executable file from the original package.

For example, if you capture Firefox 1.5, your autoupdate mechanism might prompt you to upgrade to Firefox 2.0. If you proceed with the upgrade, the application downloads the updates, writes the updates to the sandbox, and prompts you to restart the application. When you run the captured application again, Firefox 2.0 starts. If you delete the sandbox, Firefox reverts back to version 1.5.

To merge changes that an auto update mechanism makes with the original package to build an updated executable file, use the `sbmerge.exe` utility. See [“Application Updates That the Administrator Triggers”](#) on page 54.

NOTE If you use the Application Sync utility to perform application updates, disable the auto-update capabilities of the application. See [“Using Application Sync in a Managed or Unmanaged Environment”](#) on page 47.

Dynamic Updates Without Administrator Rights

You can update 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 must run the `ngen.exe` file to generate native image assemblies for startup performance. In typical circumstances, the `ngen.exe` file writes to HKLM and `C:\WINDOWS`, both of which are only accessible with administrator accounts. With ThinApp, the `ngen.exe` file can install native image assemblies on guest user accounts but stores changes in a user-specific directory.

You can update the package on a central computer and push the changes to client machines or to central network shares as a new captured executable file. Use one of the following options for applying updates:

- During the setup capture process.
- Inside the virtual environment.

Applications with auto-update capabilities can undergo updates. If the update is a `patch.exe` file, the patch program can run in the virtual environment and run from a `cmd.exe` file entry point. Changes occur in the sandbox during automatic updates or manual updates to allow you to revert to the original version by deleting the sandbox.

If you apply patches in the virtual environment on a central packaging machine, you can use the `sbmerge.exe` utility to merge sandbox changes made by the update with the application. See [“Application Updates That the Administrator Triggers”](#) on page 54.

- In the captured project.

If you must update a small set of files or registry keys, replace the files in the captured project. This approach is useful for software developers who integrate ThinApp builds with their workflow.

Upgrading Running Applications on a Network Share

ThinApp allows you to upgrade or roll back an application that is running on a network share for multiple users. The upgrade process occurs when the user quits the application and starts it a second time. In Terminal Server environments, you can have multiple users executing different versions at the same time during the transition period.

File Locks

Starting an application locks the executable file package. You cannot replace, delete, or move the application. 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 the application processes and subprocesses are running.

If you store an application in a central location for many users, this file lock prevents administrators from replacing a packaged executable file with a new version until all users exit the application and release their locks.

Upgrade a Running Application

You can copy a new version of an application into an existing deployment directory with a higher filename extension, such as `.1` or `.2`. This procedure uses Firefox as a sample application.

You do not have to update shortcuts.

Upgrade a running application

- 1 Deploy the original version of the application, such as `Firefox.exe`.
- 2 Copy the application to a central share at `\\<server>\<share>\Firefox.exe`.

A sample location is `C:\Program Files\Firefox\Firefox.exe`.

- 3 Create a desktop or **Start** menu shortcut to the user's desktop that points to a shared executable file location at `\\<server>\<share>\Firefox.exe`.

Assume two users start `Firefox.exe` and lock the application.

- 4 Copy the updated version of `Firefox.exe` to the central share at `\\<server>\<share>\Firefox.1`.

If you are a new user, ThinApp starts the application with the new package data in `Firefox.1`. If you are a user working with the original version, you can see the new version after you exit the application and restart the application.

- 5 If you must deploy a more current update of Firefox, place it in the same directory with a higher number at the end.
- 6 Copy Version 2.0 of `Firefox.exe` to central share at `\\<server>\<share>\Firefox.2`

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

Sandbox Considerations for Upgraded Applications

When you upgrade an application, you can control whether users continue to use their previous settings by keeping the sandbox name consistent in the `Package.ini` file. You can prevent users from using an older sandbox with an upgraded application by packaging the upgraded application with a new name for the sandbox. Starting the upgraded application the first time creates the sandbox with the new name.

Updating the ThinApp Version of Packages

You can use the `relink.exe` utility to update an existing package or tree of packages to the latest version of ThinApp. Although you can install the latest version of ThinApp and run the `build.bat` utility to rebuild each target package with the latest ThinApp version, the `relink.exe` utility is a faster method to upgrade the ThinApp version of existing packages. You might want to update your package to benefit from the latest ThinApp features or support enhancements.

relink Examples

The `relink.exe` utility has an optional `-Recursive` flag and can target a single package or multiple packages.

```
relink [-Recursive] <target> [<target> ...]
```

For example, you can update an Adobe Reader package to the latest installed ThinApp version.

```
relink AdobeReader.exe
```

The `relink.exe` utility can use a wildcard pattern.

```
relink *.exe *.dat
```

The `relink.exe` utility can use directory names to process all ThinApp files in that directory.

```
relink C:\MyPackages
```

If you specify the `-Recursive` flag, the `relink.exe` utility processes all ThinApp files in the directory and all subdirectories. This flag is intended for use only with directory names.

If the target name contains spaces, you must use double quotes.

```
relink "Microsoft Office Professional 2007.dat"
```

Locating the ThinApp Sandbox

The sandbox is the directory where all changes that the captured application makes are stored. The next time you start the application, those changes are incorporated from the sandbox. When you delete the sandbox directory, the application reverts to its captured state.

This information includes the following topics:

- [“Search Order for the Sandbox”](#) on page 59
- [“Controlling the Sandbox Location”](#) on page 61
- [“Sandbox Structure”](#) on page 62

Search Order for the Sandbox

During startup of the captured application, ThinApp searches for an existing sandbox in specific locations and in a specific order. ThinApp uses the first sandbox it detects. If ThinApp cannot find an existing sandbox, ThinApp creates one according to certain environment variable and parameter settings. Review the search order and sandbox creation logic before changing the placement of the sandbox.

The search order uses Mozilla Firefox 3.0 as an example with the following variables:

- `<sandbox_name>` is Mozilla Firefox 3.0

The `SandboxName` parameter in the `Package.ini` file determines the name. See [“SandboxName”](#) on page 97.

- `<sandbox_path>` is `Z:\sandboxes`

The `SandboxPath` parameter in the `Package.ini` file determines the path. See [“SandboxPath”](#) on page 97.

- `<exe_directory>` is `C:\Program Files\Firefox`

The application runs from this location.

- `<computer_name>` is `JOHNDOE-COMPUTER`

- `%AppData%` is `C:\Documents and Settings\JohnDoe\Application Data`

ThinApp requests the `Application Data` folder location from the operating system. The location depends on the operating system or configuration.

ThinApp starts the sandbox search by trying to find the following environment variables in this order:

- **%<sandbox_name>_SANDBOX_DIR%**

This environment variable changes the sandbox location for specific applications on the computer. For example, if the `Mozilla Firefox 3.0_SANDBOX_DIR` environment variable exists, its value determines the parent directory sandbox location. If the value is `z:\FirefoxSandbox` before you run the application, ThinApp stores the sandbox in `z:\FirefoxSandbox.JOHNDOE-COMPUTER` if the directory already exists. If the directory does not exist, ThinApp creates a sandbox in `z:\FirefoxSandbox`.

- **%THINSTALL_SANDBOX_DIR%**

This environment variable changes the location of all sandboxes on a computer. For example, if the `THINSTALL_SANDBOX_DIR` environment variable exists, its value determines the parent directory sandbox location. If the value is `z:\MySandboxes` before you run the application, ThinApp creates a sandbox in `z:\MySandboxes`.

If ThinApp does not detect the `%<sandbox_name>_SANDBOX_DIR%` or `%THINSTALL_SANDBOX_DIR%` environment variable, ThinApp checks for the following file system directories and creates a sandbox in the first directory it detects:

- **<exe_directory>\<sandbox_name>.<computer_name>**

For example, `C:\Program Files\Firefox\Mozilla Firefox 3.0.JOHNDOE-COMPUTER`

- **<exe_directory>\<sandbox_name>**

For example, `C:\Program Files\Firefox\Mozilla Firefox 3.0`

- **<exe_directory>\Thinstall\<sandbox_name>.<computer_name>**

For example, `C:\Program Files\Firefox\Thinstall\Mozilla Firefox 3.0.JOHNDOE-COMPUTER`

- **<exe_directory>\Thinstall\<sandbox_name>**

For example, `C:\Program Files\Firefox\Thinstall\Mozilla Firefox 3.0`

- **<sandbox_path>\<sandbox_name>.<computer_name>**

For example, `Z:\sandboxes\Mozilla Firefox 3.0.JOHNDOE-COMPUTER`

- **<sandbox_path>\<sandbox_name>**

For example, `Z:\sandboxes\Mozilla Firefox 3.0`

- **%AppData%\Thinstall\<sandbox_name>.<computer_name>**

For example, `C:\Documents and Settings\JohnDoe\Application Data\Thinstall\Mozilla Firefox 3.0.JOHNDOE-COMPUTER`

- **%AppData%\Thinstall\<sandbox_name>**

For example, `C:\Documents and Settings\JohnDoe\Application Data\Thinstall\Mozilla Firefox 3.0`

If ThinApp does not detect the `%<sandbox_name>_SANDBOX_DIR%` or `%THINSTALL_SANDBOX_DIR%` environment variable, and does not detect the specified file system directories, ThinApp creates a sandbox using the following guidelines in this order:

- If the `SANDBOXPATH Package.ini` parameter is set, the value determines the sandbox location.
- If ThinApp completes the sandbox search without any results, ThinApp creates a sandbox in the default `%AppData%\Thinstall` directory of the user.

NOTE Only one computer at a time can use a shared sandbox. If a computer is already using a sandbox, ThinApp creates a new sandbox to allow you to continue working until the previous copy of the sandbox closes.

Controlling the Sandbox Location

The setup capture process adds the `SandboxName` parameter to the `Package.ini` file. If you capture Firefox and Mozilla Firefox 3.0 is the value of this parameter, the default location of the sandbox for the application is `%AppData%\Thinstall\Mozilla Firefox 3.0`. The typical `%AppData%` location is `C:\Documents and Settings\<user_name>\Application Data`. `%AppData%` is often mapped to a shared network drive.

Store the Sandbox on the Network

You can use the `SandboxPath` parameter to store the sandbox on a mapped drive. A network location is useful for backing up the sandbox and for users who log in to any machine and keep their application settings. For more information about the `SandboxPath` parameter, see [“SandboxPath”](#) on page 97.

Store the sandbox on a mapped drive

- 1 Open the `Package.ini` file.
- 2 Under the `SandboxName` parameter, set the `SandboxPath` parameter to the network location.

```
SandboxName=Mozilla Firefox 3.0
SandboxPath=Z:\Sandbox
```

For example, if Mozilla Firefox 3.0 is the value of the `SandboxName` parameter, the captured Firefox application creates the sandbox in `Z:\Sandbox\Mozilla Firefox 3.0`.

Store the Sandbox on a Portable Device

You can use the `SandboxPath` parameter to set a portable device location for the sandbox. You can use any portable device, such as a USB drive, that appears as a disk drive in the `My Computer` system folder. A portable device location is useful to keep the sandbox data on the device where the application resides.

For more information about the `SandboxPath` parameter, see [“SandboxPath”](#) on page 97.

Store the sandbox in the same directory on a USB drive where the executable file resides

- 1 Open the `Package.ini` file.
- 2 Under the `SandboxName` parameter, set the `SandboxPath` parameter to this value.

```
SandboxName=Mozilla Firefox 3.0
SandboxPath=.
```

For example, if Mozilla Firefox 3.0 is the value of the `SandboxName` parameter, the captured Firefox application creates the Mozilla Firefox 3.0 sandbox in the same directory that Firefox runs from.

Store the Sandbox in a Thinstall Directory on a USB Drive at the Same Level as the Executable File

The sandbox in a Thinstall directory located on a USB drive must be stored at the same level at which the executable file is stored.

Store the sandbox in a Thinstall directory on a USB drive at the same level as the executable file

- 1 If the `%THINSTALL_SANDBOX_DIR%` or `%<sandbox_name>_SANDBOX_DIR%` environment variables are set, unset the variables.
- 2 On the portable device, create a `Thinstall` directory in the same directory as your captured application.
The next time the packaged application starts from the portable device, the application creates a sandbox in the `Thinstall` directory.
- 3 If the application and sandbox originally ran from another location, such as a computer, and you need the same sandbox on a portable device, copy the `Thinstall` directory from `%AppData%` to the directory where the executable file resides on the device.

ThinApp no longer uses the sandbox in the original location.

Sandbox Structure

ThinApp stores the sandbox using a file structure almost identical to the build project structure. ThinApp uses macro names for shell folder locations, such as %AppData%, instead of hard coded paths. This structure enables the sandbox to migrate to different computers dynamically when the application runs from new locations.

The sandbox contains the following registry files:

- **Registry.rw.tvr** – Contains all registry modifications that the application makes.
- **Registry.rw.lck** – Prevents other computers from simultaneously using a registry located on a network share.
- **Registry.tvr.backup** – Contains a backup of the .tvr file that ThinApp uses when the original .tvr file is corrupted.

In addition to these registry files, the sandbox contains directories that include %AppData%, %ProgramFilesDir%, and %SystemRoot%. Each of these folders contains modifications to respective folders in the captured application.

Making Changes to the Sandbox

ThinApp stores file system information in the virtual registry. The virtual registry enables ThinApp to optimize file system access in the virtual environment. For example, when an application tries to open a file, ThinApp does not have to consult the real file system for the real system location and again for the sandbox location. Instead, ThinApp can check for the existence of the file by consulting only the virtual registry. This ability increases the ThinApp runtime performance.

VMware does not support modifying or adding files directly to the sandbox. If you copy files to the sandbox directory, 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 that you perform all modifications from the application itself.

Listing Virtual Registry Contents with vregtool

Because the sandbox contains the modifications to the registry, you might need the **vregtool** utility to view modified virtual registry changes. You must have access to the **vregtool** utility in C:\Program Files\VMware\VMware ThinApp.

A sample command to list the contents of a virtual registry file is **vregtool registry.rw.tvr printkeys**.

Creating ThinApp Snapshots and Projects from the Command Line

6

The `snapshot.exe` utility creates a snapshot of a computer file system and registry and creates a ThinApp project from two previously captured snapshots. You do not have to start the `snapshot.exe` utility directly because the Setup Capture wizard starts it. Only advanced users and system integrators who are building ThinApp capability into other platforms might make direct use of this utility.

Creating a snapshot of a computer file system and registry involves scanning and saving a copy of the following data:

- File information for all local drives

This information includes directories, filenames, file attributes, file sizes, and file modification dates.

- HKEY_LOCAL_MACHINE and HKEY_USERS registry trees

ThinApp does not scan HKEY_CLASSES_ROOT and HKEY_CURRENT_USER registry entries because those entries are subsets of HKEY_LOCAL_MACHINE and HKEY_USERS entries.

The `snapshot.ini` configuration file specifies what directories and subkeys to exclude from a ThinApp project when you capture an application. You might customize this file for certain applications.

This information includes the following topics:

- [“Methods of Using the snapshot.exe Utility”](#) on page 63
- [“Sample snapshot.exe Commands”](#) on page 65
- [“Create a Project Without the Setup Capture Wizard”](#) on page 65
- [“Customizing the snapshot.ini File”](#) on page 66

Methods of Using the snapshot.exe Utility

You can use the `snapshot.exe` utility to create snapshot files of machine states, create the template file for the `Package.ini` file, create a ThinApp project, and display the contents of a snapshot file.

For information about the full procedure to create a ThinApp project from the command line, see [“Create a Project Without the Setup Capture Wizard”](#) on page 65.

Creating Snapshots of Machine States

The `snapshot.exe` utility creates a snapshot file of a machine state. ThinApp captures the machine state and saves it to a single file to create a project. The `snapshot.exe` utility saves a copy of registry data and file system metadata that includes paths, filenames, sizes, attributes, and timestamps.

Usage

```
snapshot.exe SnapshotFileName.snapshot [-Config ConfigFile.ini] [BaseDir1] [BaseDir2] [BaseReg1]
```

Examples

```
Snapshot My.snapshot
Snapshot My.snapshot -Config MyExclusions.ini
Snapshot My.snapshot C:\MyAppDirectory HKEY_LOCAL_MACHINE\Software\MyApp
```

Options

The options specify the directories or subkeys in the snapshot.

Table 6-1. Snapshot Directories and Subkeys

Option	Description
-Config ConfigFile.ini	Specifies directories or registry subkeys to exclude during snapshot creation. If you do not specify a configuration file, ThinApp uses the <code>snapshot.ini</code> file from the ThinApp installation directory.
BaseDir1	Specifies one or more base directories to include in the scan. If you do not specify base directories, the <code>snapshot.exe</code> utility scans <code>C:\</code> and all subdirectories. If you scan a machine where Windows or program files are installed on different disks, include these drives in the scan. If you know that your application installation creates or modifies files in fixed locations, specify these directories to reduce the total time required to scan a machine.
BaseReg1	Species one or more base registry subkeys to include in the scan. If you do not specify registry subkeys, the <code>snapshot.exe</code> utility scans the <code>HKEY_LOCAL_MACHINE</code> and <code>HKEY_USERS</code> keys.

Creating the Template Package.ini file from Two Snapshot Files

The `snapshot.exe` utility generates a template `Package.ini` file. The utility scans the two snapshot files for all applications that are created and referenced from shortcut links or the **Start** menu. The template `Package.ini` file becomes the basis of the `Package.ini` file in a ThinApp project.

Usage

```
snapshot.exe Snap1.snapshot -SuggestProject Snap2.snapshot OutputTemplate.ini
```

Examples

```
Snapshot Start.snapshot -SuggestProject End.snapshot Template.ini
```

ThinApp requires all of the parameters.

Creating the ThinApp Project from the Template Package.ini File

The `snapshot.exe` utility creates the ThinApp project file from the template `Package.ini` file.

Usage

```
snapshot.exe Template.ini -GenerateProject OutDir [-Config ConfigFile.ini]
```

Examples

```
Snapshot Template.ini -GenerateProject C:\MyProject
Snapshot Template.ini -GenerateProject C:\MyProject -Config MyExclusions.ini
```

-Config ConfigFile.ini is optional. The configuration file specifies directories or registry subkeys for exclusion from the project. If you do not specify a configuration file, ThinApp uses the `snapshot.ini` file.

Displaying the Contents of a Snapshot File

The `snapshot.exe` utility lists the contents of the snapshot file.

Usage

```
snapshot.exe SnapshotFileName.snapshot -Print
```

Examples

```
Snapshot Start.snapshot -Print
```

ThinApp requires all of the parameters.

Sample snapshot.exe Commands

Table 6-2 describes sample commands for the `snapshot.exe` utility. The parameters are not case-sensitive. The commands are wrapped in the Command column because of space restraints.

Table 6-2. snapshot.exe Sample Commands

Command	Description
<code>snapshot C:\Capture.snapshot</code>	Captures a complete snapshot of local drives and registry to the file <code>C:\Capture.snapshot</code> .
<code>snapshot C:\Capture.snapshot C:\ E:\</code>	Captures a complete snapshot of the <code>C:\</code> and <code>E:\</code> drives. ThinApp does not capture registry information.
<code>snapshot C:\Capture.snapshot C:\data.snapshot C:\ HKEY_LOCAL_MACHINE</code>	Captures a complete snapshot of the <code>C:\</code> drive and all of the <code>HKEY_CLASSES_ROOT</code> registry subtree.
<code>snapshot C:\Original.snapshot -Diff C:\NewEnvironment.snapshot C:\MyProject</code>	Generates a ThinApp project directory by comparing two snapshots.
<code>snapshot Original.snapshot -DiffPrint NewEnvironment.snapshot</code>	Displays differences between two captured snapshots.
<code>snapshot C:\data.snapshot C:\ HKEY_LOCAL_MACHINE</code>	Saves the state of the computer file system and registry.
<code>snapshot C:\start.snapshot -diffprint C:\end.snapshot</code>	Compares two recorded states.
<code>snapshot C:\start.snapshot -print</code>	Prints the contents of a saved state.
<code>snapshot C:\start.snapshot -SuggestProject C:\end.snapshot C:\project.ini</code>	Generates a ThinApp project by comparing two saved states.

Create a Project Without the Setup Capture Wizard

You can use the `snapshot.exe` utility from the command line instead of using the Setup Capture wizard that runs the `snapshot.exe` utility in the background. The command-line utility is useful to package a large number of applications or automate ThinApp project creation. The typical location of the `snapshot.exe` utility is `C:\Program Files\VMware\VMware ThinApp\snapshot.exe`.

The snapshot process makes a copy of the all registry entries on the system and file system metadata. File system metadata includes path, filename, attribute, size, and time stamp information but excludes actual file data.

Create a project with the snapshot.exe command-line utility

- 1 Save an initial snapshot of the current machine configuration to disk.
`snapshot.exe C:\Start.snapshot`
- 2 Install the application and make any necessary manual system changes.

- 3 Save to disk a snapshot of the new machine configuration.

snapshot.exe C:\End.snapshot

- 4 Generate a template Package.ini file.

snapshot.exe C:\Start.snapshot -SuggestProject C:\End.snapshot C:\Template.ini

ThinApp uses the template file to generate the final Package.ini file. The template file contains a list of all detected executable file entry points and Package.ini parameters. If you write your own script to replace the Setup Capture wizard, use the template Package.ini file to select the entry points to keep or customize Package.ini parameters such as InventoryName.

- 5 Generate a ThinApp project.

snapshot.exe C:\Template.ini -GenerateProject C:\MyProjectDirectory

- 6 (Optional) Delete the temporary C:\Start.snapshot, C:\End.snapshot, and C:\Template.ini files.

- 7 (Optional) To generate multiple projects with different configurations, reuse the original Start.snapshot file and repeat the procedure from [Step 2](#).

Customizing the snapshot.ini File

The snapshot.ini configuration file specifies what registry keys to exclude from a ThinApp project when you capture an application.

For example, if you use Internet Explorer 7, you might need ThinApp to capture the following registry keys:

- HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Desktop\Components
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Connections
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Hardware Profiles\0001\Software\Microsoft\windows\CurrentVersion\Internet Settings

If the snapshot.ini file excludes the

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet

Settings\Connections key by default, you can remove this key from the snapshot.ini file to ensure that ThinApp captures the key in the capture process.

If you do not customize the snapshot.ini file, the snapshot process loads the file from one of these locations:

- Application Data\Thinapp\snapshot.ini

This location is the AppData directory of the user.

- C:\Program Files\VMware\VMware Thinapp\snapshot.ini

This is the location from which ThinApp runs the snapshot.exe utility.

ThinApp File System Formats and Macros

7

ThinApp stores the differences between snapshots during the setup capture process in a virtual file system and virtual registry. The virtual file system uses folder macros to represent Windows shell folder locations.

This information about the virtual file system includes the following topics:

- [“Virtual File System Formats”](#) on page 67
- [“ThinApp Folder Macros”](#) on page 67

Virtual File System Formats

ThinApp generates the following virtual file system formats:

- **Build**

The setup capture process generates this format from files found directly on the physical file system. ThinApp uses folder macros to represent Windows shell folder locations.

- **Embedded**

The `build.bat` file triggers a build process that embeds a read-only file system in executable files. The executable files provide block-based streaming to client computers. ThinApp compresses the file system.

- **Sandbox**

Running the captured application generates the read-write directory structure that holds file data that the application modifies. File modifications that prompt ThinApp to extract embedded virtual files to the sandbox include the following operations:

- Changing the time stamp or attributes of a file
- Opening a file with write access
- Truncating a file
- Renaming or moving a file

The embedded and sandbox file systems use folder macros to enable file paths to dynamically expand at runtime.

ThinApp Folder Macros

ThinApp uses macros to represent file system path locations that might change when virtualized applications run on different Windows operating systems or computers. The use of macros enables shared application profile information to instantly migrate to different operating systems.

For example, you might capture an application on a system that has C:\WINNT as the Windows directory and deploy the application on a system that has C:\Windows as the Windows directory. ThinApp transparently converts C:\WINNT to %SystemRoot% during the capture process for that system and expands %SystemRoot% to C:\Windows during runtime for that system.

If an application registers DLLs to C:\winnt\system32 while running on Windows 2000, the user can quit the application and log in to a Windows XP machine. On the Windows XP machine, the files appear to exist at C:\windows\system32 and all related registry keys point to C:\windows\system32.

On Windows Vista, ThinApp moves Windows SxS DLLs and policy information to match Windows Vista instead of using Windows XP file path styles. This feature enables most applications to migrate to updated or older operating systems.

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

List of ThinApp Macros

ThinApp uses the shfolder.dll file to obtain the location of shell folders. Older versions of the shfolder.dll file do not support some macro names.

Macros requiring shfolder.dll version 5.0 or later include %ProgramFilesDir%, %Common AppData%, %Local AppData%, %My Pictures%, and %Profile%.

Macros requiring shfolder.dll version 6.0 or later include %My Videos%, %Personal%, and %Profiles%.

Table 7-1 lists the available folder macros.

Table 7-1. Folder Macros

Macro Name	Typical Location
%AdminTools%	C:\Documents and Settings\<user_name>\Start Menu\Programs\Administrative Tools
%AppData%	C:\Documents and Settings\<user_name>\Application Data
%CDBurn Area%	C:\Documents and Settings\<user_name>\Local Settings\Application Data\Microsoft\CD Burning
%Common AdminTools%	C:\Documents and Settings\All Users\Start Menu\Programs\Administrative Tools
%Common AppData%	C:\Documents and Settings\All Users\Application Data
%Common Desktop%	C:\Documents and Settings\All Users\Desktop
%Common Documents%	C:\Documents and Settings\All Users\Documents
%Common Favorites%	C:\Documents and Settings\All Users\Favorites
%Common Programs%	C:\Documents and Settings\All Users\Start Menu\Programs
%Common StartMenu%	C:\Documents and Settings\All Users\Start Menu
%Common Startup%	C:\Documents and Settings\All Users\Start Menu\Programs\Startup
%Common Templates%	C:\Documents and Settings\All Users\Templates
%Cookies%	C:\Documents and Settings\<user_name>\Cookies
%Desktop%	C:\Documents and Settings\<user_name>\Desktop
%Drive_c%	C:\
%Drive_m%	M:\
%Favorites%	C:\Documents and Settings\<user_name>\Favorites
%Fonts%	C:\Windows\Fonts
%History%	C:\Documents and Settings\<user_name>\Local Settings\History

Table 7-1. Folder Macros (Continued)

Macro Name	Typical Location
%Internet Cache%	C:\Documents and Settings\ <user_name>\Local Settings\Temporary Internet Files</user_name>
%Local AppData%	C:\Documents and Settings\ <user_name>\Local Settings\Application Data</user_name>
%My Pictures%	C:\Documents and Settings\ <user_name>\My Documents\My Pictures</user_name>
%My Videos%	C:\Documents and Settings\ <user_name>\My Documents\My Videos</user_name>
%NetHood%	C:\Documents and Settings\ <user_name>\NetHood</user_name>
%Personal%	C:\Documents and Settings\ <user_name>\My Documents</user_name>
%PrinHood%	C:\Documents and Settings\ <user_name>\PrinHood</user_name>
%Profile%	C:\Documents and Settings\ <user_name>< td=""></user_name><>
%Profiles%	C:\Documents and Settings
%Program Files Common%	C:\Program Files\Common Files
%ProgramFilesDir%	C:\Program Files
%Programs%	C:\Documents and Settings\ <user_name>\Start Menu\Programs</user_name>
%Recent%	C:\Documents and Settings\ <user_name>\My Recent Documents</user_name>
%Resources%	C:\Windows\Resources
%Resources Localized%	C:\Windows\Resources\ <language_id>< td=""></language_id><>
%SendTo%	C:\Documents and Settings\ <user_name>\SendTo</user_name>
%Startup%	C:\Documents and Settings\ <user_name>\Start Menu\Programs\Startup</user_name>
%SystemRoot%	C:\Windows
%SystemSystem%	C:\Windows\System32
%TEMP%	C:\Documents and Settings\ <user_name>\Local Settings\Temp</user_name>
%Templates%	C:\Documents and Settings\ <user_name>\Templates</user_name>

Processing %SystemRoot% in a Terminal Services Environment

A Terminal Services environment has a shared Windows directory, such as C:\Windows, and a private Windows directory, such as C:\Documents and Settings\User\Windows. In this environment, ThinApp uses the user-specific directory for %SystemRoot%.

Creating ThinApp Scripts

Scripts modify the behavior of virtual applications dynamically. You can create custom code before starting an application packaged with ThinApp or after an application exits. You can use scripts to authenticate users and load configuration files from a physical to virtual environment.

Callback functions run code during specific events. If applications create child processes, use callback functions to run code only in the main parent process.

API functions run ThinApp functions and interact with the ThinApp runtime. API functions can authenticate users and prevent the start of applications for unauthorized users.

Adding scripts to your application involves creating an ANSI text file with the `.vbs` file extension in the root application project directory. The root project directory is the same directory that contains the `Package.ini` file. During the build process, ThinApp adds the script files to the executable file and runs each of the script files at runtime.

ThinApp uses VBScript to run script files. For information about VBScript, see the Microsoft VBScript documentation. You can use VBScript to access COM controls registered on the host system or within the virtual package.

This information includes the following topics:

- [“Callback Functions”](#) on page 71
- [“Implement Scripts in a ThinApp Environment”](#) on page 72
- [“API Functions”](#) on page 75

Callback Functions

Callback functions can run under certain conditions. For example, callback functions run script code only when an application starts or quits.

Callback function names include the following names:

- **OnFirstSandboxOwner** – Called only when an application first locks the sandbox. This callback is not called if a second copy of the same application uses the same sandbox while the first copy runs. If the first application spawns a subprocess and quits, the second subprocess locks the sandbox and prevents this callback from running until all subprocesses quit and the application runs again.
- **OnFirstParentStart** – Called before running a ThinApp executable file regardless of whether the sandbox is simultaneously owned by another captured executable file.
- **OnFirstParentExit** – Called when the first parent process exits. If a parent process runs a child process and quits, this callback is called even if the child process continues to run.
- **OnLastProcessExit** – Called when the last process owning the sandbox exits. If a parent process runs a child process and quits, this callback is called when the last child process exits.

The following callback example shows the `OnFirstSandboxOwner` and `OnFirstParentExit` functions.

```
-----example.vbs -----
Function OnFirstSandboxOwner
msgbox "The sandbox owner is: " + GetCurrentProcessName
End Function

Function OnFirstParentExit
msgbox "Quitting application: " + GetCurrentProcessName
End Function

msgbox "This code will execute for all parent and child processes"
-----
```

Implement Scripts in a ThinApp Environment

You might implement a script in the following circumstances:

- Timing out an application on a specific date.
- Running a `.bat` file from a network share inside the virtual environment.
- Modifying the virtual registry.
- Importing the `.reg` file at runtime.
- Stopping a virtual service when the main application quits.
- Copying an external system configuration file into the virtual environment on startup.

Implement a script

- 1 Save the script contents in a plain text file with the `.vbs` extension in the same directory as your `Package.ini` file.

You can use any filename. ThinApp adds all `.vbs` files to the package at build time.

- 2 Rebuild the application.

.bat Example

The following script runs an external `.bat` file from a network share inside the virtual environment. The `.bat` file makes modifications to the virtual environment by copying files, deleting files, or applying registry changes using `regedit /s regfile.reg`. Run this script only for the first parent process. If you run this script for other processes, each copy of the `cmd.exe` utility runs the script and an infinite recursion develops.

```
Function OnFirstParentStart
Set Shell = CreateObject("Wscript.Shell")
Shell.Run "\\jcdesk2\test\test.bat"
End Function
```

Timeout Example

The following script prevents the use of an application after a specified date. The VBS date uses the `#mm/dd/yyyy#` format, regardless of locale.

This check occurs upon startup of the parent process and any child processes.

```
if Date >= #03/20/2007# then
msgbox "This application has expired, please contact Administrator"
ExitProcess 0
end if
```


Modify the Virtual Registry

The following script procedure modifies the virtual registry at runtime to load an external ODBC driver from the same directory where the package executable file is located.

Modify the registry

- 1 Obtain the path to the package executable files.

```
Origin = GetEnvironmentVariable("TS_ORIGIN")
```

- 2 Find the last slash in the path and obtain the characters that precede the slash.

```
LastSlash = InStrRev(Origin, "\")
SourcePath = Left(Origin, LastSlash)
```

- 3 Form a new path to the ODBC DLL file located outside of the package.

```
DriverPath=SourcePath + "tsodbc32.dll"
```

- 4 Modify the virtual registry to point it to this location.

```
Set WSHShell = CreateObject("Wscript.Shell")
WSHShell.RegWrite "HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\Transoft ODBC Driver\Driver,"
DriverPath
```

This modification causes the application to load the DLL from an external location.

.reg Example

The following script imports 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 a Service Example

The following script stops a virtual or native service when the main application quits.

```
Function OnFirstParentExit
Set WshShell = CreateObject("WScript.Shell")
WshShell.Run "net stop ""iPod Service""""
End Function
```

Copying a File Example

The following script sections shows how to copy a configuration file located in the same directory as the captured executable file into the virtual file system each time the application starts. This script is useful for an external configuration file that is easy to edit after deployment. Because the copy operation occurs each time you run the application, any changes to the external version are reflected in the virtual version.

For example, if your captured executable file is running from \\server\share\myapp.exe, this script searches for a configuration file located at \\server\share\config.ini and copies it to the virtual file system location at C:\Program Files\my application\config.ini.

By putting this code in the OnFirstParentStart function, it is only called once each time the script runs. Otherwise it runs for every child process.

```
Function OnFirstParentStart
```

ThinApp sets up TS_ORIGIN to indicate the full path to a captured executable file package. A virtual application sets the TS_ORIGIN variable to the physical path of the primary data container. If you have a virtual application consisting of the `main.exe` and `shortcut.exe` files, both files reside in `C:\VirtApp`. When you run the `main.exe` file, TS_ORIGIN var is set to `C:\VirtApp\main.exe`. When you run the `shortcut.exe` file, the TS_ORIGIN environment variable is set to `C:\VirtApp\main.exe`. The environment variable is always set to the primary data container, even when you create a shortcut. When you run VBScripts that are included in the package, the variable is already set and available to the scripts.

```
Origin = GetEnvironmentVariable("TS_ORIGIN")
```

You can separate the filename from TS_ORIGIN by finding the last backslash and removing all of the characters following it.

```
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 to copy to might be a different location on different computers if the Program Files directory is mapped to a location other than `C:\`. The following call lets ThinApp expand a macro to obtain the correct location for the local computer.

```
DestFile = ExpandPath("%ProgramFilesDir%\MyApplication\Config.ini")
```

Use the `fileSystemObject` parameter to check the source file exists.

```
Set objFSO = CreateObject("Scripting.filesystemObject")
If objFSO.FileExists(SourceFile) Then
```

If the source file exists, copy it into the virtual file system. The `%ProgramFilesDir%\MyApplication` virtual directory is in the package.

```
objFSO.CopyFile SourceFile, DestFile, TRUE
End if
End Function
```

Add a Value to the System Registry

This script procedure adds a value to the physical system registry.

Add a value to the system registry

- 1 Create a `.reg` file and run the `regedit /s` command as an external process that accesses the system registry instead of the virtual registry.

Function OnFirstParentStart

- 2 Create the `.reg` file in a location that has the `IsolationMode` parameter set to `Merged` so that the virtual environment can access it with this script and the physical environment can it with the `regedit /s` command.

```
RegFileName = ExpandPath("%Personal%\thin.reg")
Set fso = CreateObject("Scripting.filesystemObject")
Set RegFile = fso.CreateTextFile(RegFileName, true)
```

The `%Personal%` directory is a directory that has `Merged` isolation mode by default.

- 3 Construct the `.reg` file.

```
RegFile.WriteLine("Windows Registry Editor Version 5.00")
RegFile.WriteLine(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
```

- 4 Add the information in the system registry.

```
RegEditPid = ExecuteExternalProcess("regedit /s " & chr(34) & RegFileName & chr(34))
WaitForProcess RegEditPid, 0
```

Wait until the process is complete.

- 5 Clean the environment.

```
fso.DeleteFile(RegFileName)
End Function
```

API Functions

You can use API functions that instruct ThinApp to complete operations such as load DLLs as virtual DLLs, convert paths from macro format to system format, and run commands inside of the virtual environment.

AddForcedVirtualLoadPath

The `AddForcedVirtualLoadPath(Path)` 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 depend on DLLs located inside the package.

You can use the `ForcedVirtualLoadPaths` parameter in the `Package.ini` file to achieve the same result as this API function. See [“ForcedVirtualLoadPaths”](#) on page 71.

Parameters

Path

[in] The filename or path for DLLs to load as virtual.

Examples

You can load any DLL located in the same directory as the executable file as a virtual DLL.

```
Origin = GetEnvironmentVariable("TS_ORIGIN")
```

`TS_ORIGIN` is the path from which the executable file is running.

You can delete the filename from `TS_ORIGIN` by finding the last backslash and removing all of the characters that follow it.

```
LastSlash = InStrRev(Origin, "\")
SourcePath = Left(Origin, LastSlash)
```

You can instruct ThinApp to load all DLLs in the same or lower directory from where the source executable file resides.

```
AddForcedVirtualLoadPath(SourcePath)
```

This process enables you to drop additional files in the `SourcePath` tree and have them resolve import operations against virtual DLLs.

ExitProcess

The `ExitProcessExitCode` 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. A value of 0 indicates no error.

Examples

You can exit the process and indicate success.

```
ExitProcess 0
```

When the process exits, the scripting system receives its `OnLastProcessExist` function callback. Any loaded DLLs run termination code to clean up the environment.

ExpandPath

The `ExpandPath(InputPath)` 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.

Examples

```
Path = ExpandPath("%ProgramFilesDir%\Myapp.exe")
```

```
Path = C:\Program Files\myapp.exe
```

All macro paths must escape the % and # characters by replacing these characters with #25 and #23.

```
Path = ExpandPath("%ProgramFilesDir%\FilenameWithPercent#25.exe")
```

This expands to `C:\Program Files\FilenameWithPercent%.exe`.

ExecuteExternalProcess

The `ExecuteExternalProcess(CommandLine)` function runs a command outside of the virtual environment. You can use this function to make physical system changes.

Parameters

`CommandLine`

[in] Representation of the application and command-line parameters to run outside of the virtual environment.

Returns

Integer process ID. You can use the process ID with the `WaitForProcess` function. See [“WaitForProcess”](#) on page 81.

Examples

```
ExecuteExternalProcess("C:\WINDOWS\system32\cmd.exe /c copy C:\systemfile.txt  
C:\newsystemfile.txt")
```

You can run a command that requires quotation marks in the command line.

```
ExecuteExternalProcess("regsvr32 /s " & chr(34) & "C:\Program Files\my.ocx" & chr(34))
```

ExecuteVirtualProcess

The `ExecuteVirtualProcess(CommandLine)` function runs a command inside of the virtual environment. You can use this function to make changes to the virtual environment.

Parameters

`CommandLine`

[in] Representation of the application and command-line parameters to run outside of the virtual environment.

Returns

Integer process ID. You can use the process ID with the `WaitForProcess` function. See [“WaitForProcess”](#) on page 81.

Examples

```
ExecuteVirtualProcess("C:\WINDOWS\system32\cmd.exe /c copy C:\systemfile.txt C:\virtualfile.txt")
```

You can run a command that requires quotation marks in the command line.

```
ExecuteVirtualProcess("regsvr32 /s " & chr(34) & "C:\Program Files\my.ocx" & chr(34))
```

GetBuildOption

The `GetBuildOption(OptionName)` function returns the value of a setting specified in the `[BuildOptions]` section of the `Package.ini` file used for capturing applications.

Parameters

`OptionName`

[in] Name of the setting.

Returns

This function returns a string value. If the requested option name does not exist, the function returns an empty string (`""`).

Examples

`Package.ini` contains:

```
[BuildOptions]
```

```
CapturedUsingVersion=4.0.1-2866
```

The following line appears in a VBS file:

```
Value = GetBuildOption("CapturedUsingVersion")
```

GetFileVersionValue

The `GetFileVersionValue(Filename, Value)` function returns version information value from files such as a specific DLL, OCX, or executable file. You can use this function to determine the internal version number of a DLL or retrieve DLL information about the copyright owner or a product name.

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.

You can retrieve the following values 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 function cannot locate the specified value in the file, the function returns an empty string ("").

Examples

```
FileVersion = GetFileVersionValue("C:\windows\system32\kernel32.dll," "FileVersion")

if FileVersion = "1.0.0.0" then
    MsgBox "This is Version 1.0!"

End if
```

GetCommandLine

The `GetCommandLine` function accesses the command-line parameters passed to the running program.

Returns

This function returns a string that represents the command-line arguments passed to the current running program, including the original executable file.

Examples

```
MsgBox "The command line for this EXE was " + GetCommandLine
```

GetCurrentProcessName

The `GetCurrentProcessName` function accesses 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. In most circumstances, this path is `C:\Program Files\...`, even if the package source runs from a network share.

Examples

```
MsgBox "Running EXE path is " + GetCurrentProcessName
```

GetOSVersion

The `GetOSVersion()` function returns information about the current version of Windows.

Parameters

This function has no parameters.

Returns

This function returns a string in the `MAJOR.MINOR.BUILD_NUMBER.PLATFORM_ID OS_STRING` format.

MAJOR is one the following values:

Windows Vista	6
Windows Server 2008	6
Windows Server 2003	5
Windows XP	5
Windows 2000	5
Windows NT 4.0	4

MINOR is one of the following values:

Windows Vista	0
Windows Server 2008	0
Windows Server 2003	2
Windows XP	1
Windows 2000	0
Windows NT 4.0	0
Windows NT 3.51	51

BUILD_NUMBER is the build number of the operating system.

PLATFORM_ID assigns 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 information about the operating system such as `Service Pack 2`.

Examples

```
if GetOSVersion() = "5.1.0.2 Service Pack 2"
    then MsgBox "You are running on Windows XP Service Pack 2!"
endif
```

GetEnvironmentVariable

The `GetEnvironmentVariable(Name)` function returns the environment variable associated with the `Name` variable.

Parameters

`Name`

[in] The name of the environment variable for which the value is retrieved.

Returns

This function returns the string value associated with the `Name` environment variable.

Examples

```
MsgBbox "The package source EXE is " + GetEnvironmentVariable("TS_ORIGIN")
```

RemoveSandboxOnExit

The `RemoveSandboxOnExit(YesNo)` function set toggles that determine whether to delete the sandbox when the last child process exits.

If you set the `RemoveSandboxOnExit` parameter to 1 in the `Package.ini` file, the default cleanup behavior for the package with is Yes. You can change the cleanup behavior to No by calling `RemoveSandboxOnExit` with the value of 0. If you do not modify the `RemoveSandboxOnExit=1` entry in the `Package.ini` file, the default cleanup behavior for the package is No. You can change the cleanup behavior 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

Examples

The following example turns on cleanup.

```
RemoveSandboxOnExit 1
```

The following example turns off cleanup.

```
RemoveSandboxOnExit 0
```

SetEnvironmentVariable

The `SetEnvironmentVariable(Name, Value)` function set the value of an environment variable.

Parameters

`Name`

[in] The name of the environment variable to store the value.

`Value`

[in] The value to store.

Examples

```
SetEnvironmentVariable "PATH", "C:\Windows\system32"
```


SetfileSystemIsolation

The `Setfile systemIsolation(Directory, IsolationMode)` 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 set.

1 = WriteCopy

2 = Merged

3 = Full

Examples

You can set the Merged isolation mode for the temp directory.

```
Setfile systemIsolation GetEnvironmentVariable("TEMP"), 2
```

SetRegistryIsolation

The `SetRegistryIsolation(RegistryKey, IsolationMode)` function sets the isolation mode of a registry key.

Parameters

RegistryKey

[in] The registry key on which to set the isolation mode. Start with HKLM for HKEY_LOCAL_MACHINE, HKCU for HKEY_CURRENT_USER, and HKCR for HKEY_CLASSES_ROOT.

IsolationMode

[in] Isolation mode to set.

1 = WriteCopy

2 = Merged

3 = Full

Examples

You can set the Full isolation mode for HKEY_CURRENT_USER\Software\Thinapp\Test.

```
SetRegistryIsolation "HKCU\Software\Thinapp\Test," 3
```

WaitForProcess

The `WaitForProcess(ProcessID, TimeOutInMilliseconds)` function waits until the process ID is finished running.

Parameters

ProcessID

[in] The process ID to end. The process ID can come from `ExecuteExternalProcess` or `ExecuteVirtualProcess`.

TimeOutInMilliseconds

[in] The maximum amount of time to wait for the process to finish running before continuing. A value of 0 specifies INFINITE.

Returns

This function returns an integer.

0 = Timeout fails

1 = Process exits

2 = Process does not exist or security is denied

Examples

```
id = ExecuteExternalProcess("C:WINDOWS\system32\cmd.exe")  
WaitForProcess(id, 0)
```

Monitoring and Troubleshooting ThinApp

9

You can use Log Monitor to generate trace files and troubleshoot the ThinApp environment. Log Monitor is compatible only with an application captured using the same version of ThinApp.

This information includes the following topics:

- [“Providing Information to Technical Support”](#) on page 83
- [“Log Monitor Operations”](#) on page 83
- [“Troubleshooting Specific Applications”](#) on page 90

Providing Information to Technical Support

VMware technical support requires the following information from you to troubleshoot a ThinApp environment:

- Step-by-step reproduction of the procedure that you performed when you encountered the problem.
- Information on the host configuration. Specify the Windows operating system, the use of Terminal Server or Citrix Xenapp, and any prerequisite programs that you installed on the native machine.
- Copies of the Log Monitor trace files. See [“Log Monitor Operations”](#) on page 83.
- Exact copy of the capture folder and all content. Do not include the compiled executable files from the /bin subfolder.
- Description of the expected and accurate behavior of the application.
- (Optional) Copies of the applications that you captured. Include the server components configuration for Oracle Server or Active Directory.
- (Optional) Native or physical files or registry key settings that might be relevant to the problem.
- (Optional) System services or required device drivers.
- (Optional) Virtual machine that reproduces the defect. VMware support might request this if the support contact is unable to reproduce the problem.
- (Optional) One or more WebEx sessions to facilitate debugging in your environment.

Log Monitor Operations

Log Monitor captures detailed chronological activity for executable files that the captured application starts. Log Monitor intercepts and logs names, addresses, parameters, and return values for each function call by target executable files or DLLs. Log Monitor captures the following activity:

- Win32 API calls from applications running in the ThinApp virtual operating system.
- Potential errors, exceptions, and security events within the application.
- All DLLs loaded by the application and address ranges.

The generated log files can be large and over 100MB depending on how long the application runs with Log Monitor and how busy an application is. The only reason to run Log Monitor for an application is to capture trace files. Trace files are critical for troubleshooting problems by analyzing and correlating multiple entries within the trace file.

Troubleshoot Activity with Log Monitor

You can use Log Monitor to perform basic troubleshooting.

Troubleshoot ThinApp logs

- 1 Shut down the captured application to investigate.
- 2 On the computer where you captured the application, select **Start > Programs > VMware > ThinApp Log Monitor**.

To start Log Monitor on a deployment machine, copy the `log_monitor.exe`, `logging.dll`, and `Setup Capture.exe` files from `C:\Program Files\VMware\VMware ThinApp` to the deployment machine and double-click the `log_monitor.exe` file.

- 3 Start the captured application.

As the application starts, a new entry appears in the Log Monitor list. Log Monitor shows one entry for each new trace file. Each file does not necessarily correspond with a single process.

- 4 End the application as soon as it encounters an error.
- 5 Generate logs for each trace file you want to investigate.

- a Select the `.trace` file in the list.
- b Click **Generate text trace report**.

Child processes that the parent process generates reside in the same log. Multiple independent processes do not reside in the same log.

ThinApp generates a `.trace` file. Log Monitor converts the binary `.trace` file into a `.txt` file.

- 6 (Optional) Open the `.txt` file with a text editor and scan the information. In some circumstances, the `.txt` file is too large to open with the text editor.
- 7 Zip the `.txt` files and send the files to VMware support.

Perform Advanced Log Monitor Operations

Advanced operations in Log Monitor include stopping applications or deleting trace files. If an application is busy or experiencing slow performance with a specific action, you can perform suspend and resume operations to capture logs for a specific duration. The resulting log file is smaller than the typical log file and easier to analyze. Even when you use the suspend and resume operations, the cause of an error might occur outside of your duration window. Suspend and resume operations are global and affect all applications.

For more information about using these options, contact VMware support.

Perform advanced Log Monitor operations

- 1 Shut down the captured application to investigate.
- 2 On the computer where you captured the application, select **Start > Programs > VMware > ThinApp Log Monitor**.

To start Log Monitor on a deployment machine, copy the `log_monitor.exe`, `logging.dll`, and `Setup Capture.exe` files from `C:\Program Files\VMware\VMware ThinApp` to the deployment machine and double-click the `log_monitor.exe` file.

- 3 (Optional) Capture logs for a specific duration to troubleshoot an exact issue.
 - a Select the **Suspend** check box.
 - b Start the captured application and let it run to the point where the error occurs or the performance problem starts.
 - c In Log Monitor, deselect the **Suspend** check box to resume the logging process.
You can check the application behavior to isolate the issue.
 - d Select the **Suspend** check box to stop the logging process.
- 4 (Optional) Select a file in the trace file list to delete and click **Delete File**.
- 5 (Optional) Click **Kill App** to stop a running process.
- 6 (Optional) Click the **Compress** check box to decrease the size of a trace file.
This operation slows the performance of the application.
- 7 (Optional) Generate a trace file report.
 - a Select a trace file in the file list, type a trace filename, or click **Browse** to select a trace file on your system.
 - b (Optional) Type or change the name of the output report.
 - c Click **Generate text trace report** to create a report.
You can view the file with a text editor that supports UNIX-style line breaks.

Locating Errors

ThinApp logging provides a large amount of information. The following tips might help advanced users investigate errors:

- Review the **Potential Errors Detected** section of the `.txt` trace file.
Entries might not indicate errors. ThinApp lists each Win32 API call where the Windows error code changed.
- Review exceptions that the applications generate.
Exceptions can indicate errors. Exception types include C++ and .NET. The trace file records the exception type and DLL that generates the exception. If the application, such as a .NET or Java application, creates an exception from self-generating code, the trace file indicates an unknown module.
The following example is a `.trace` entry for an exception.
*** Exception EXCEPTION_ACCESS_VIOLATION on read of 0x10 from unknown_module:0x7c9105f8
If you find an exception, scan the earlier part of the trace file for the source of the exception. Ignore the floating point exceptions that Virtual Basic 6 applications generate during typical use.
- Review child processes.
Log Monitor produces one `.trace` file for each process. If an application starts several child processes, determine which process is causing the problem. Sometimes, such as in circumstances involving out-of-process COM, a parent application uses COM to start a child process, runs a function remotely, and continues to run functions.
- When you run applications from a network share that generates two processes, ignore the first process.
ThinApp addresses the slow performance of Symantec antivirus applications by restarting processes.

- Search for the error message displayed in dialog boxes.

Some applications call the `MessageBox Win32` API function to display unexpected errors at runtime. You can search a trace file for `MessageBox` or the contents of the string displayed in the error and determine what the application was running just before the dialog box appeared.

- Narrow the focus on calls originating from a specific DLL and thread.

The log format specifies the DLL and thread that makes a call. You can often ignore the calls from system DLLs.

Log Format

A trace file includes the following sections:

- System configuration

This section includes information about the operating system, drives, installed software, environment variables, process list, services, and drivers.

The information starts with a `Dump started` on string and ends with a `Dump ended` on string.

- Header

This section shows contextual information for the instance of the process that Log Monitor tracks. Some of the displayed attributes show logging options, address ranges when the operating system runtime is loaded, and macro mapping to actual system paths.

ThinApp marks the beginning of the header section with sequence number 000001. In typical circumstances, ThinApp marks the end of this section with a message about the Application Sync utility.

- Body

This section includes trace activity as the application starts and performs operations. Each line represents function calls that target executable files or one of the DLLs make.

The section starts with a `New Modules detected in memory` entry, followed by the `SYSTEM_LOADED` modules list. The section ends with a `Modules Loaded` entry.

- Summary

This section includes modules that the captured application loads, potential errors, and a profile of the 150 slowest calls.

The section starts with the `Modules Loaded` message.

General API Log Message Format

The following message shows a format example for API calls.

```
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 ->B00L=1h ()
```

This example includes the following entries:

- 000257 indicates the log entry number. Each log entry has a unique number.
- 0a88 indicates the current running thread ID. If the application has one thread, this number does not change. If two or more threads record data to the log file, you might use the thread ID to follow thread-specific sequential actions because ThinApp records log entries in the order in which they occur.
- mydll.dll indicates the DLL that makes the API call.
- 4ad0576d indicates the return address for the API call that mydll.dll makes. In typical circumstances, the return address is the address in the code where the call originates.
- -> indicates the process of entering the call. For the call entry log element, ThinApp displays the input parameters. These parameters are in and in/out parameters.

- <- indicates the process of the call returning to the original caller. For call exit log entries, ThinApp displays the output parameters. These parameters are out and in/out parameters.
- kernel32.dll indicates the DLL where the API call lands.
- 7c81b1f0 indicates the address of the API inside kernel32 where the call lands. If you disassemble kernel32.dll at the 7c81b1f0 address, you find the code for the SetConsoleMode function.
- ->B00L=1h indicates the API returns the value of 1 and the return code has the BOOL type.

Application Startup Information

The following entries shows basic information about the application, such as the module name and process ID (PID), and about Log Monitor, such as the version and options.

```
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 `Modules loaded` section is located near the end of the log file and describes the DLLs that are loaded into memory at runtime and the DLL addresses. The information shows whether Windows or ThinApp loads the DLLs.

This example includes a summary of the length of the longest calls and the following entries:

- `SYSTEM_LOADED` indicates that Windows loads the DLL. The file must exist on the disk.
- `MEMORY_MAPPED_ANON` indicates that ThinApp loads the DLL. ThinApp might load the file from the virtual file system.
- `46800000-46873fff` indicates the address range in virtual memory where the DLL resides.
- `PRELOADED_BY_SYSTEM` and `PRELOADED_MAP` are duplicate entries and refer to the memory address range where the executable image file is mapped into memory.

```
---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
MEMORY_MAPPED_ANON 013b0000-020affff, C:\Program Files\Adobe\Reader
8.0\Reader\AcroRd32.dll
```

```
----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
.
```

```

.
... 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
.
.
... 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
.
.
... 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

```

Potential Errors

The **Potential Errors Detected** section marks log entries that might post problems with three asterisks (***). For information about interpreting this section, see [“Locating Errors”](#) on page 85.

----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-Controls.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-Controls.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 :78131dc<-dbghelp.dll :59a603a1 *** GetModuleHandleA
->HMODULE=0h () *** GetLastError() returns 126 [0]: The specified module could not be found.

```

Troubleshooting Example for cmd.exe Utility

In the troubleshooting example, ThinApp packages the `cmd.exe` utility with logging turned on. The example shows how you can simulate application failure by running an invalid command. If you request the `cmd.exe` utility to run the `foobar` command, the utility generates the `foobar is not recognized as an internal or external command` message. You can scan the trace file and check the **Potential Errors Detected** section to find the API functions that modified the `GetLastError` code.

The example shows the `C:\test\cmd_test\bin\foobar.*`, `C:\WINDOWS\system32\foobar.*`, and `C:\WINDOWS\foobar` paths as the locations where the `cmd.exe` utility looks for the `foobar` command.

The example shows the %drive_C%\test\cmd_test\bin, %SystemSystem%\foobar, and %SystemRoot%\foobar paths as the locations in the virtual file system that ThinApp 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=fffffffh .. *** 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]
```

Perform Advanced Examination for cmd.exe Log Entries

A more thorough examination of an entry from the Potential Errors section of a trace file might involve searching the full body of the Log Monitor trace file for that specific entry and reviewing the system calls and conditions leading to the potential error.

For example, the following entry for the cmd.exe utility in the Potential Errors section might require a more thorough examination throughout the Log Monitor trace file.

```
001550 *** FindFirstFileW 'C:\test\cmd_test\bin\foobar.*' -> INVALID_HANDLE_VALUE *** failed
[system probe
```

Perform an advanced examination of the cmd.exe entry

- 1 To determine why the cmd.exe utility probes c:\test\cmd_test\bin, scan the log for this log entry number and determine what occurs before this call.
- 2 To determine the locations where the cmd.exe utility obtains the c:\test\cmd_test path, scan the log for GetCurrentDirectoryW and GetFullPathNameW entries.

```
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 ()
.
.
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")
.
.
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]
```

The `cmd.exe` utility obtains the first location by calling `GetCurrentDirectoryW` and the second location by calling `GetFullPathNameW` with "." as the path specifies. These calls return the path for the current working directory. The log file shows that the `cmd.exe` utility creates the `C:\test\cmd_test\bin>` prompt. The utility queries the `PROMPT` environment variable that returns `PG` and uses the `WriteConsoleW` API function to print the prompt to the screen after internally expanding `PG` to `C:\test\cmd_test\bin>`.

Troubleshooting Specific Applications

Troubleshooting tips are available for capturing Microsoft Outlook, Explorer.exe, and Java Runtime Environment.

Troubleshoot Registry Setup for Microsoft Outlook

Microsoft Outlook stores account settings in registry keys and files. When you start Microsoft Outlook for the first time, it checks that the keys exist. If Microsoft Outlook cannot find the keys, it prompts you to create an account.

This process works properly in the virtual environment when Microsoft Outlook is not installed on the physical system. If the user already has Microsoft Outlook installed on the physical system, the captured version finds the registry keys in the system registry and uses those settings. You must use Full isolation mode for the registry keys and files where Microsoft Outlook stores its settings.

Set up Full isolation mode for Microsoft Outlook registry keys

- 1 Add the following entries 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
```
- 2 Create a `##Attributes.ini` file with the following entries:


```
[Isolation]
DirectoryIsolationMode=Full
```
- 3 Place the `##Attributes.ini` file in each of the following subdirectories.


```
%AppData%\Microsoft\AddIns
%AppData%\Microsoft\Office
%AppData%\Microsoft\Outlook
%Local AppData%\Microsoft\FORMS
%Local AppData%\Microsoft\Outlook
```
- 4 (Optional) If the subdirectories do not exist, create the directories.

Viewing Attachments in Microsoft Outlook

Microsoft Outlook creates a default directory to store attachments when you open an attachment for viewing. The typical location is `C:\Documents and Settings\<user_name>\Local Settings\Temp\Temporary Internet Files\OLK<xxxx>`. The last `xxxx` is replaced by a random entry.

You can view attachments when the viewing application runs in the same virtual sandbox as Microsoft Outlook. External applications might not be able to find the file to display because Microsoft Outlook stores the file in the sandbox. You must use the Merged isolation mode for the directory that stores the attachments.

Set up Merged isolation mode to view Microsoft Outlook attachments

- 1 Add a value to the `HKEY_CURRENT_USER.txt` file that sets the name of the attachment directory:


```
isolation_full
HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Outlook\Security
Value=OutlookSecureTempFolder
REG_SZ~%Profile%\Local Settings\OutlookTemp\xxxx#2300
```

In this example, 11.0 in the key name is for Microsoft Outlook 2003.

- 2 Replace the last four xxxx characters with random alphanumeric entries to increase security.
- 3 Create a directory that is named in the OutlookSecureTempFolder registry key in your ThinApp project.
For example, create the %Profile%\Local Settings\OutlookTempxxxx directory.
- 4 In the %Profile%\Local Settings\OutlookTempxxxx directory, create a ##Attributes.ini file with the following entries:

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

Starting Explorer.exe in the Virtual Environment

Running one instance of the explorer.exe utility on a Windows operating system makes it difficult to add an entry point to Windows Explorer and start it inside the virtual environment.

You can use the following methods to open a Windows Explorer window inside the virtual environment:

- Add an entry point to iExplorer and start it with the -E parameter.

For example, add the following entries to the Package.ini file:

```
[iexplore.exe]
Shortcut=xxxx.exe
Source=%ProgramFilesDir%\Internet Explorer\iexplore.exe
CommandLine=%ProgramFilesDir%\Internet Explorer\iexplore.exe -E
```

- Add the following virtual registry key:

```
isolation_full HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer
Value=DesktopProcess
REG_DWORD=#01#00#00#00
```

- Add the following entries to the Package.ini file:

```
[explorer.exe]
Shortcut=xxxxxx.exe
Source=%SystemRoot%\explorer.exe
```

Use this method to browse the virtual file system with a familiar interface and enable accurate file type associations without system changes, especially when using portable applications. You can access shell-integrated components without system changes.

Troubleshooting Java Runtime Environment Version Conflict

A conflict might occur if one version of Java is installed on the physical system and another version is included in a captured executable file. Updated versions of Java install a plug-in DLL that Internet Explorer loads. This plug-in DLL overwrites virtual registry keys and conflicts with a virtualized copy of older Java runtimes.

Prevent Internet Explorer from loading plug-in DLLs

Add the following entry to the beginning of the HKEY_LOCAL_MACHINE.txt file.

```
isolation_full HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser
Helper Objects
```


Glossary

A **Application Link**

A utility that links dependent applications to a base application at runtime and starts all the applications together when you start the base application. You can use the utility to deploy and update component packages separately rather than capture all components in the same package.

Application Sync

A utility that updates an application by detecting a new packaged version on a server or network share. You can configure update settings, such as the checking of an update server at certain intervals. ThinApp detects the most recent application executable file and downloads the differences.

attributes.ini

The file that applies configuration settings at the directory level of the package rather than the entire package. The `##Attributes.ini` settings override the overall `Package.ini` settings.

B **build**

To convert a ThinApp project into a package. You can build a package with the Setup Capture wizard or with the `build.bat` utility.

C **capture**

To package an application into a virtual environment and set initial application parameters. ThinApp provides the Setup Capture wizard or the `snapshot.exe` utility to create a portable application package that is independent of the operating system it runs on.

clean machine

The computer or virtual machine, installed with only the basic Windows operating system, on which you capture the application. The Windows operating system version must be the earliest version of Windows that you expect the application to run on.

E **entry point**

An executable file that starts the captured application. An application might have multiple entry points. For example, the `Firefox.exe` file might serve as an entry point for a Mozilla Firefox application. The primary data container file can exist within an entry point or as a `.dat` file.

I **inventory name**

A name that ThinApp uses for internal tracking of the application. The inventory name sets the default project directory name and appears in the **Add or Remove Programs** dialog box for Windows.

isolation mode

A package setting that determines the read and write access to the physical environment. ThinApp has WriteCopy, Merged, and Full isolation modes.

L logging.dll

A utility that generates `.trace` files.

Log Monitor

A utility that captures chronological activity for executable files that the captured application starts. The `log_monitor.exe` file is compatible only with applications captured using the same version of ThinApp.

M MSI

A Windows Installer container that is useful for application deployment tools. You can deliver the captured application as an MSI file instead of an executable file.

N native

Refers to the physical environment rather than the virtual environment. *See also* [physical](#).

network streaming

The process of running a package from a central server. ThinApp downloads blocks of the application as needed to ensure quick processing and display.

P package

The virtual application files that the ThinApp build process generates. The package includes the primary data container file and entry point files to access the application.

package.ini

The file that applies configuration settings to the package and that resides in the captured application folder. The Setup Capture wizard sets the initial values of the configuration settings.

physical

Refers to the computer memory and file system in which all standard Windows processes run. Depending on ThinApp isolation mode settings, processes in the virtual environment can access the physical environment. *See also* [native](#), [virtual](#).

postscan

To establish an image or snapshot of a machine after you install the application you want to capture. The capture process stores in a virtual file system and virtual registry the differences between the prescan and postscan images. *See also* [prescan](#), [snapshot](#).

prescan

To establish a baseline image or snapshot of a machine before you install the application you want to capture. The capture process stores in a virtual file system and virtual registry the differences between the prescan and postscan images. *See also* [postscan](#), [snapshot](#).

primary data container

The main virtual application file. The file is a `.exe` file or a `.dat` file that includes the ThinApp runtime and the read-only virtual file system and virtual registry. The primary data container must reside in the same `/bin` directory with any subordinate application executable files because entry points use the information in the primary data container.

project

The data that the capture process creates before you build a package. The capture process uses the inventory name as the default project directory name. You can customize parameters in the project files before you build an application package. You cannot deploy a captured application until you build a package from the project.

S**sandbox**

The physical system folder that stores runtime user changes to the virtual application. When you start the application, ThinApp incorporates changes from the sandbox. When you delete the sandbox, ThinApp reverts the application to its captured state. The default location of the sandbox is %APPDATA%\Thinstall\<application_name>.

sbmerge.exe

A utility that makes incremental updates to applications, such as the incorporation of a plug-in or a change in a browser home page. The `sbmerge.exe` utility merges runtime changes recorded in the sandbox back into a ThinApp project.

snapshot

A recording of the state of the Windows file system and registry during the application capture process. The Setup Capture process uses the `snapshot.exe` utility to take a snapshot before and after the application is installed and stores the differences in a virtual file system and virtual registry. *See also* [postscan](#), [prescan](#).

snapshot.exe

A utility that creates the snapshots of a computer file system and registry and facilitates the prescan and postscan operations during the capture process. Only advanced users who build ThinApp functionality into other platforms might make direct use of this utility. *See also* [postscan](#), [prescan](#), [snapshot](#).

snapshot.ini

A configuration file that specifies the directories and subkeys to exclude from a ThinApp project when you capture an application. You can customize this file for applications.

T**template.msi**

A template for MSI files that you can customize to adhere to company deployment procedures and standards. For example, you can add registry settings for ThinApp to add to client computers as part of the installation.

thinreg.exe

A utility that establishes file type associations, sets up **Start** menu and desktop shortcuts, and facilitates the opening of files. You must run the `thinreg.exe` utility to register executable files. MSI files automate the `thinreg.exe` registration process.

tlink.exe

A utility that links key modules during the build process.

V**vftool.exe**

A utility that compiles the virtual file system during the build process.

virtual

Refers to the logical file and memory within which a captured application runs. Processes in a physical environment cannot access the virtual environment. *See also* [physical](#).

virtual application

An application that you capture to make it portable and independent of the operating system it runs on.

virtual file system

The file system as the captured application sees it.

virtual registry

The registry as the captured application sees it.

vregtool.exe

A utility that compiles the virtual registry during the build process.

Index

Symbols

##Attributes.ini

comparing to Package.ini 22

editing 22

A

Active Directory

authorizing group access 16

controlling access to applications 39

using Package.ini parameters 39

API parameters

AddForcedVirtualLoadPath 75

ExecuteExternalProcess 76

ExecuteVirtualProcess 77

ExitProcess 75

ExpandPath 76

GetBuildOption 77

GetCommandLine 78

GetCurrentProcessName 78

GetEnvironmentVariable 80

GetFileVersionValue 77

GetOSVersion 79

RemoveSandboxOnExit 80

SetEnvironmentVariable 80

SetfileSystemIsolation 81

SetRegistryIsolation 81

WaitForProcess 81

Application Link

defining 47, 50

defining access with the PermittedGroups parameter 53

effect on isolation modes 53

file and registry collisions 54

linking packages to base applications and using Application Sync 54

sample workflow 51

setting up nested links 52

storing multiple versions of linked applications 54

view of 51

Application Sync

clashing with automatic update capabilities 47

defining 47

editing parameters 48

effect on entry point executable files 49

effect on thinreg.exe 34

fixing incorrect updates 48

forcing updates with appsync.exe commands 55

maintaining the primary data container name 49

updating base applications with linked packages 54

updating thinreg.exe registrations 49

applications

capturing 13

controlling access for Active Directory groups 39

data statistics 18

difference between Application Sync and Application Link 47

not supported by ThinApp 10

sandbox considerations during upgrade processes 58

streaming requirements and recommendations 42

updating 47

C

capturing applications

IE6 on Windows XP 22

phases of 13

requirements and dependencies 13

using ThinApp Converter 24

with the Setup Capture wizard 14–21

with the snapshot.exe utility 65

cmd.exe, defining 15

compression

for executable files 20

for trace files 85

computers

defining a clean system 10

using virtual machines for clean systems 11

cut and paste operations, ThinApp limitations 43

D

data container, See primary data container

DCOM services, access for captured applications 10

deploying

applications on network share 34

applications with deployment tools 33

executable files 34

- MSI files **33**
- deployment tools, using MSI files **33**
- device drivers, incompatible with ThinApp **10**
- DLLs
 - loading into memory **87**
 - recording by Log Monitor **83**
- drivers, support for **43**

E

- entry points
 - defining **15**
 - for troubleshooting **15**
 - in Setup Capture wizard **15**
 - updating with Application Sync **49**

G

- global hook DLLs, reduced function with ThinApp **10**

I

- IE6 on Windows XP
 - capturing **22**
 - requirements **22**
- iexplore.exe, defining **15**
- installing ThinApp **11**
- inventory name, purpose of **19**
- isolation modes
 - defining **16**
 - Merged **16**
 - sample configuration **45**
 - using Application Link **53**
 - WriteCopy **17**

L

- log format **86**
- Log Monitor
 - extra options **84**
 - suspending and resuming logging **84**
 - troubleshooting procedures **84**
 - using **83**

M

- Merged isolation mode **16**
- Microsoft Vista, deploying MSI files **39**
- MSI files
 - automating the thinreg.exe utility **19**
 - building the database **37**
 - customizing parameters **37**
 - deploying on Microsoft Vista **39**
 - generating **19, 20**
 - modifying the Package.ini **37**
 - overriding the installation directory **38**

N

- nested links, using Application Link **52**
- network, streaming packages **41**

O

- operating systems
 - support for **9**
 - using the lowest version for ThinApp installation **11**

P

- Package.ini
 - Active Directory parameters **39**
 - common parameters **21**
 - editing Application Sync parameters **48**
 - modifying MSI parameters **37**
 - MSI parameters **37**
- packages
 - building **21**
 - configuring **20, 21**
 - defining **19**
- parameters
 - applying settings at folder level instead of package level **22**
 - for MSI files **37**
 - for sbmerge.exe **56**
 - for thinreg.exe **35**
- PermittedGroups, effect on Application Link **53**
- primary data container
 - defining **19**
 - maintaining the name with Application Sync **49**
 - size implications **19**
- project files **20**
- projects, opening during capture process **20**

R

- regedit.exe, defining **15**
- relink
 - defining **58**
 - examples **58**

S

- sandbox
 - considerations for upgraded applications **58**
 - defining **18**
 - location **18, 61**
 - search order **59**
 - structure **62**
- sbmerge.exe
 - commands **56**
 - defining **54**
 - merging runtime changes **55**
- scripts

- .bat example **72**
- .reg example **73**
- callback functions **71**
- file copy example **73**
- reasons for **72**
- service example **73**
- system registry example **74**
- timeout example **72**
- virtual registry example **73**
- services
 - automatic startup **40**
 - starting and stopping in packages **40**
- Setup Capture wizard
 - authorizing user groups **16**
 - browsing projects **21**
 - building packages **21**
 - compressing packages **19**
 - entry points **15**
 - installing applications **14**
 - inventory name **19**
 - package settings **20**
 - postscan operation **14**
 - prescan operation **14**
 - project location **19**
 - setting isolation modes **18**
- shell integration, reduced functions with ThinApp **10**
- snapshot.exe
 - creating snapshots from the command line **63**
 - sample commands **65**
 - sample procedure **65**
- snapshot.ini, defining **63, 66**
- support
 - for applications **9**
 - for operating systems **9**
- T**
- technical support
 - required information for troubleshooting **83**
- ThinApp
 - applications that are not supported **10**
 - browsing project files **20**
 - deployment options **33**
 - directory files **11**
 - folder macros **67**
 - in a VMware View environment **33**
 - installing **11**
 - recommendation for clean computers **10**
 - requirements for installing and capturing applications **9**
 - streaming packages from the network **41**
 - supported operating systems and applications **9**
 - updating applications **47**
 - updating runtime in packages **58**
 - using thinreg.exe **34**
- ThinApp Converter
 - capturing multiple applications **24**
 - configuration file **25**
 - detecting application installation processes **29**
 - process overview **24**
 - system requirements **25**
- ThinAppConverter.ini
 - configuring AppSettings **30**
 - configuring HostEnvironment **25**
 - configuring Settings **28**
 - configuring VirtualMachineN **27**
 - predefined environment variables **31**
- ThinDirect
 - extracting and registering **24**
- thinreg.exe
 - defining **34**
 - parameters **35**
 - running **35**
 - starting with MSI files **19**
 - updating registrations with Application Sync **49**
 - with Application Sync **34**
- troubleshooting
 - Explorer.exe **91**
 - Java Runtime Environment **91**
 - Microsoft Outlook **90**
 - providing required information to support **83**
 - with Log Monitor **84**
- U**
- upgrading applications, methods and considerations **47–58**
- V**
- virtual file system
 - format stages **67**
 - representing path locations with macros **67**
 - using **67**
- VMware View, using captured applications **33**
- vregtool, listing virtual registry contents **62**
- W**
- WriteCopy isolation mode **17**

