VMware ThinApp User's Manual

VMware ThinApp 4.0.1

VMware ThinApp User Manual
Item: EN-000117-00

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

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

Contents

About This Book

The *VMware ThinApp User's Manual* provides information about how to install 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 wants to install and use ThinApp. Typical users are system administrators responsible for the distribution and maintenance of corporate software packages.

Document Feedback

VMware welcomes your suggestions for improving our documentation. If you have comments, send your feedback to docfeedback@vmware.com.

Technical Support and Education Resources

The following sections describe the technical support resources available to you. 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.html.

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.

1 Installing ThinApp

Review the ThinApp installation requirements and process. This information discusses the following topics:

Reviewing ThinApp Requirements

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

Operating Systems, Applications, and Systems That ThinApp Supports

ThinApp supports these operating systems, applications, and systems:

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

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

ThinApp does not support these operating systems and applications:

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

Applications that ThinApp Cannot Virtualize

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

You must use traditional installation technologies to deploy the following types of applications:

- Applications requiring installation of kernel-mode device drivers

  ODBC drivers work because they are user mode
- 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 open, edit, and save PDF files, but you cannot see or use the PDF printer driver that allows you to save documents to PDF format.

Shell Integration

Some applications that provide shell integration might have reduced functions as a ThinApp package. When ThinApp virtualizes applications, the applications might lose some shell integration functions with the system explorer shell.

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 DLLs

Some applications use the `SetWindowsHookEx` API function to add a DLL to all processes on the host computer. The added 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

Review the recommendations for installing ThinApp.

Using a Clean Computer

VMware recommends using a clean system to install ThinApp because the environment affects the application capture process. 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 and the application might fail to run on other computers where the files do not exist. A clean machine allows the capture process to scan the computer file system and registry quickly.

A clean system is a physical or virtual machine with a Windows operating system installed and nothing else. In a corporate environment where you have a base desktop image, the base desktop image is your clean system. The desktop computer might already have some components and libraries installed.

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

Using Virtual Machines for Clean Systems

The easiest way to set up a clean system 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 VMware Workstation or other VMware products to create virtual machines. For information about VMware products, see the VMware Web site.

Using the Lowest Operating System Required For Users

Install ThinApp on a clean machine with the lowest version of the operating system you plan to support. In most cases, the lowest 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 does not have. 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

Use the `ThinApp.msi` file to install ThinApp. For information about the directory that the installation creates, see Appendix C, "ThinApp Directory Files," on page 135.

To install ThinApp

1 Download ThinApp to a clean physical or virtual Windows machine.

2 Double-click the `ThinApp.msi` file.

3 Accept the license, enter the serial number, and enter the license display name that will appear when you launch applications captured by ThinApp.

4 Click **Next** to install ThinApp.

Capturing Applications 2

You can capture applications with the Setup Capture wizard. For information about capturing applications from the command line, see Appendix D, "ThinApp Snapshot Files," on page 137.

This chapter uses Mozilla Firefox as a key example for application capture and includes the following topics:

Reviewing the Capture Process

The capture process involves these phases:

- Snapshot of the clean machine.
- Installation of the application that ThinApp needs to capture.
- Snapshot of the machine after the application installation.

  ThinApp assesses the differences between the initial snapshot and this snapshot.
- Configuration of the application.

  For example, setting Firefox as a default browser, setting a home page, and setting default security settings.

  You can configure the ThinApp package to set compression, sandbox, domain user access to applications, and more.
- Build of the ThinApp application package.

Assessing Application Dependencies Before the Capture Process

Before capturing an application, assess whether the application has any dependencies on other applications, libraries, or frameworks and whether to capture these dependencies. VMware recommends using the Application Link utility to link separate components at runtime. See Chapter 4, "Updating Applications," on page 51.

Recommended Tasks Prior to the Capture Process

Review the recommended tasks prior to starting the Setup Capture wizard:

- Shut down applications, such as virus scans, that might change the file system while ThinApp takes snapshots.
- Review other areas of this manual if you are not already familiar with the ThinApp sandbox that stores application run-time changes, isolation modes that affect resource visibility and access outside the captured application package, application compression options, generation of MSI files instead of executable files, and the folders with macros that represent file system paths in the captured application directory.

Capture an Application with the Setup Capture Wizard

Use the Setup Capture wizard to package an application and set initial application parameters. If you use a virtual machine, consider taking a snapshot prior to running the wizard. A snapshot of the original clean state allows you to revert to the snapshot anytime you want to capture another application.

To capture an application with the Setup Capture wizard

1 Download the applications you want to capture.

   For example, download `Firefox Setup 2.0.0.3.exe` and copy it to the clean machine you are working with.

2 Go to **Start** > **Programs** > **VMware** > **ThinApp Setup Capture**.

3 (Optional) In the dialog box that clarifies the definition and use of a clean computer, click the **Advanced Settings** button 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 location. In rare cases, you might want to avoid scanning a registry hive if you know the application installer does not modify the registry.

4 Click **Next** to begin the first snapshot of the hard drive and registry files.

The scanning process takes about 10 seconds for Windows XP.

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

For example, double-click `Firefox Setup 2.0.0.3.exe to` install Firefox. If the application needs to reboot after the installation, reboot the system. The reboot launches the Setup Capture wizard again.

6 Make any necessary configuration changes to comply with your organization's policies, such as using specific security settings or a particular home page.

If you do not make configuration changes, each user will have to make changes that might lead to inconsistent setup.

7 (Optional) Launch the application and address any prompts for information prior to continuing with the Setup Capture wizard.

If you do not address any prompts ahead of time, every user who uses the application will have to go through that process during the initial launch.

8 Close the application.

9 Maximize the Setup Capture wizard and click **Next** to proceed with another snapshot of the machine.

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

10 Specify the entry points, primary data container, and inventory name:

a Select the check boxes for user-accessible entry points.

Entry points are the executable files that start the virtual application. The entry points you can choose from depend on the executables 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.

If you generate MSI files instead of executables and deploy the application on target machines, these entry points create desktop shortcuts.

For information about the `cmd.exe`, `regedit.exe`, and `iexplore.exe` entry point choices, see "Entry Points for Troubleshooting" on page 26.

b Select the primary data container, the file that stores virtual files and registry information, from the list based on the selected entry pointed.

- If the size of the 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 main data container.
- If the size of the container is larger than 200MB, ThinApp creates a separate `.dat` file as the primary container because of problems associated with large `.exe` files, such as the inability of Microsoft Windows to show shortcut icons stored in large `.exe` files. ThinApp generates small `.exe` files along with the `.dat` file to store the icons for Windows to display.
- If the size of the primary container is between 200MB and 1.5GB, ThinApp creates the default `.dat` file but you can override the default and select a `.exe` file. This choice generates a warning that you can dismiss. This selection allows all applications to work properly but might prevent the proper display of icons.

c (Optional) Change the inventory name that ThinApp uses for internal tracking of the application in the `Package.ini` file.

ThinApp uses the inventory name during the update process with the Application Sync utility.

11 (Optional) Select the Active Directory groups that you want to authorize for access to the application and the sandbox location:

a Click **Add**.

- To specify objects, click **Object Types**.
- To specify a location in the forest, click **Locations**.
- To search of object names, enter the names according to the examples in the dialog box.
- To locate user names in the Active Directory forest, click **Advanced** and use the **Common Queries** tab to search for groups according to names, descriptions, disabled accounts, passwords, and days since the last login.

For example, you might restrict access to an application to ensure users do not pass it to unauthorized users.

b Select the ThinApp sandbox location depending on whether you want to deploy it to a local user machine, carry it on a mobile USB stick, or store it in a network location.

If you deploy the sandbox to a local machine, use the user's profile. If you store the sandbox in a network drive, enter the relative path to the location where you want the sandbox created. A sample path is `\\thinapp\sandbox\Firefox`. You can select a network locations even if an application is installed on a local machine.

12 Select the isolation mode to determine which files and registry keys are visible and written by the virtual application you create:

- To allow the application to read resources on and write to the local machine, keep the default **Merged** isolation mode.

  This means that the application can modify elements outside of the virtual application package. Some applications rely on DLLs and registry information in the local system image. VMware recommends this mode for applications that do not write to system directories, such as Firefox. ThinApp directs write operations to the physical file system, with the exception of the following directories and their subdirectories that redirect write operations to the sandbox:

  - `%AppData%`
  - `%Common AppData%`
  - `%Local AppData%`
  - `%Program Files Common%`
  - `%ProgramFilesDir%`
  - `%SystemRoot%`
  - `%SystemSystem%`

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

- To allow the application to read resources on the local machine and restrict most modifications to the sandbox, select the **WriteCopy** isolation mode.

  ThinApp intercepts write operations to all directories and redirects them to the sandbox, with the exception of the following directories and their subdirectories that redirect write operations to the physical system:

  - `%Desktop%`
  - `%Personal%`
  - `%SystemSystem%\Spool`

Regardless of the choice for Merged or WriteCopy isolation mode, ThinApp treats write operations to network drives according to the `SandboxNetworkDrives` parameter in the `Package.ini` file. This parameter has a default value of 0 that directs write operations to the physical drive. ThinApp treats write operations to removable disks according to the `SandboxRemovableDrives` parameter in the `Package.ini`. This parameter has a default value of 0 that directs write operations to the physical drive.

VMware recommends this mode for legacy or untrusted applications. Although the WriteCopy mode might make it difficult to locate user data files that reside in the sandbox instead of the actual system, this mode is useful for locked down desktops where you want to prevent users from affecting the operating file system and registry files.

NOTE Any modification to any virtual element in the captured application package is stored in the sandbox, regardless of the isolation mode setting. If you capture Microsoft Office and use group policies, isolation modes do not affect the application because group policies can apply registry changes only to the physical environment. Virtual and physical registry elements appear as one to the captured application. If virtual entries exist for the registry, the Merged isolation mode cannot affect the application because Microsoft Office resorts to the virtual entries instead of physical entries. If necessary, remove such virtual entries to ensure the application affects the physical system, regardless of the isolation mode.

---

13 (Optional) Change the directory where you want to save the application package.

The package stores the captured software applications. 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)`.

14 (Optional) Select the **Build MSI package** check box and change the MSI filename.

A typical Firefox application does not require an MSI installation. But 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.

Unlike executable files that involve running the `thinreg.exe` utility on every machine to register file-type associations, register desktops shortcuts, display control panel extensions, and more to facilitate file launching, an MSI package automates this process when you deploy the application on target machines.

15 (Optional) If you want a lighter executable file for locations such as a USB stick, select the **Fast compression** radio button.

In typical circumstances, compression reduces the on-disk storage requirement by 50 percent but slows performance to uncompress the package on target machines.

16 Click **Next** to create the ThinApp project.

17 In the final dialog box, launch one of these options and click **Finish**.

- Click **Browse Project** to look at the ThinApp project files in Windows Explorer.

  For example, if you captured Firefox 2.0.0.3, the location of the project files might be `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3`. You might browse the project prior to building the application executable or MSI file to update a setting, such as an Active Directory specification, in the `Package.ini` file that contains the parameters set during the wizard.

  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. The isolation mode setting at the granular folder level overrides the overall isolation mode setting of the `Package.ini` file.

- Click the **Build Now** button to build an executable or MSI file containing the files you installed during the Setup Capture process. The build output appears in the display box.

You can rebuild the project at any time after clicking **Finish** if you need to make changes.

Entry Points for Troubleshooting

If you want to troubleshoot or debug your environment, you can select these entry points during the setup capture process:

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

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

Performing Post-Capture Tasks

Post-capture activity might include the following tasks:

- "Modifying Isolation Modes" on page 26
- "Modifying Settings in the Package.ini File" on page 27
- "Modifying Settings in the ##Attributes.ini File" on page 28

Modifying Isolation Modes

ThinApp provides the `Merged` and `WriteCopy` isolation mode choices in the Setup Capture wizard. For information about those modes, see Step 12 of "Capture an Application with the Setup Capture Wizard" on page 20.

You can use a third isolation mode, `Full`, outside the wizard in the project text files. The `Full` isolation mode secures the virtual bubble by blocking visibility to system elements outside the virtual application package. This mode restricts generated data to the sandbox and ensures no interaction exists with the environment outside the virtual application package.

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

You can modify isolation modes in the `Package.ini` and `##Attributes.ini` files. See "Edit the Package.ini File" on page 27 and "Edit the ##Attributes.ini File" on page 29. For information about the impact of application updates on isolation modes, see "Impacting Isolation Modes with Application Link" on page 57.

Modifying Settings in the Package.ini File

The `Package.ini` file contains configuration settings and resides 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`.

The following parameters are examples of settings that you might customize:

- `DirectoryIsolationMode` – Sets the isolation mode to `Merged`, `WriteCopy`, or `Full`.

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

- `PermittedGroups` – Restricts use of an application package to a specific set of Active Directory users.

- `SandboxName` – Names the ThinApp sandbox.

  You might keep the name for incremental application updates and change the name for major updates.

- `SandboxPath` – Sets the sandbox location.

  You can set the sandbox in a USB location if the application executable resides in that 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.

  If ThinApp cannot import a package, ThinApp stops the base application.

- `OptionalAppLinks` – Specifies a list of external ThinApp packages to import to the current package at runtime.

  If ThinApp cannot import a package, ThinApp allows the base application to start.

For general information about all `Package.ini` parameters, see Appendix A, "Package.ini Parameters," on page 91. For more information about parameters that affect MSI file generation, see . For information about parameters that affect application updates, see Chapter 4, "Updating Applications," on page 51.

Edit the Package.ini File

Use Notepad or another text editor to update the `Package.ini` file.

To edit 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 you want to edit by removing the semicolon at the beginning of the line.

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

```
RemoveSandboxOnExit=1
```

Another example might involve commenting out the `Protocols` parameter if you do not want Firefox to take over the protocols.

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 applies settings at the directory level while the `Package.ini` file applies settings at the overall application 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 granular 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.

If you need to compress only certain folders with large files rather than an entire application, you can compress files at the folder level with the `CompressionType` parameter in the `##Attributes.ini` file.

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`. Use Notepad or another text editor to update the file.

Edit the ##Attributes.ini File

Use Notepad or another text editor to update the `##Attributes.ini` file.

To edit the ##Attributes.ini file

1 In the `##Attibutes.ini`, uncomment, update, or delete the parameter.

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

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

3 Deploying Applications

Working with captured applications might involve working with deployment tools, `thinreg.exe`, MSI files, and Active Directory. This information discusses the following topics:

Reviewing ThinApp Deployment Options

You can deploy ThinApp in the following circumstances

- Using 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.

  You can create MSI files for the captured applications and follow the same process you use to deploy native MSI files. Refer to deployment instructions from the tool vendors. For information about MSI files, see "Building an MSI Database" on page 37.

- In a VMware View environment.

  If you work with VMware View, the general workflow involves 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 non-persistent desktops. See "Facilitating File Launching with thinreg.exe" on page 32.
  - Controlling user access to fileshares. IT administrators might address this by organizing network shares based on function and associating permissions with network shares based on those functional boundaries.

- On a network share.

  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 this network share option and the VMware View option is that this options assumes a mix of physical and virtual (persistent) desktops and users run the `thinreg` utility directly instead of relying on login scripts.

- Using executable files.

  This is the basic option that applies to an environment conscious of disk usage. You can create executable files for the captured applications, copy them from a central repository, run the `thinreg.exe` utility manually to register file-type associations, desktop shortcuts, and the application package on the system.

Facilitating File Launching with thinreg.exe

ThinApp requires you to use the `thinreg.exe` file to facilitate the launching of files, such as a `.doc` document or an `.html` page. For example, if you click a URL on an email message, ThinApp needs to know to launch Firefox. You do not have to run the `thinreg.exe` file for MSI files because MSI files invoke the utility automatically during installation.

`Thinreg.exe` creates the Start menu and desktop shortcuts, sets up file type associations, adds uninstall information to the system control panel, and unregisters previously registered packages. The utility allows 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, `Thinreg.exe` allows you to see the same menu options for a `.doc` file in a native environment.

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

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

Impact of Application Sync on thinreg.exe

Review the impact of the `Application Sync` utility on `thinreg.exe`:

- If you add, modify, or remove executable files, `thinreg.exe` reregisters the file type associations, shortcuts, and icons.
- If you install protocols, MIME types, control panel applets, and templates other than executables, `thinreg.exe` reregisters these elements.

Run the thinreg.exe Utility

This procedure provides some sample `thinreg.exe` commands. The package name in the thinreg.exe commands can appear in the following ways:

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

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

  If the path or filename contains spaces, you must enclose the path in double quotes. A sample command is `thinreg.exe "\\DEPLOYSERVER\ThinApps\Microsoft Office Word 2007.exe"`.

For information about `thinreg.exe` parameters, see "Optional thinreg.exe Parameters" on page 34.

To run the thinreg.exe utility

1 Determine the executable files that ThinApp has to register with the local environment.

2 From the command prompt, type:

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

If the server name is DEPLOYSERVER and the share is `ThinApps`, use this sample command to register Microsoft Word for the logged-in user:

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

Use this sample command 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

Thinapp provides optional parameters for the thinreg.exe utility. See Table 3-1.

Standard use of `thinreg.exe` accounts for the `PermittedGroups` setting in the `Package.ini` file, registering and unregistering packages as needed. When `thinreg.exe` 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 executables.

When `thinreg.exe` 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 quotes.

For information about the `PermittedGroups` setting, see the Appendix A, "Package.ini Parameters," on page 91.

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"` This command requires administrator rights.
/q, /quiet	Blocks output.	`thinreg.exe /q "\\<server>\<share>\Microsoft Office 2007 Word.exe"`

/u, /unregister, /uninstall	Unregisters a package.
	This command removes the software from the Add/Remove Programs control panel applet.
		Unregisters Microsoft Word for the current user:
		`thinreg.exe /u "\\<server>\<share>\Microsoft Office 2007 Word.exe"`
		Unregisters all Microsoft Office applications for the current user and removes the Add/Remove Programs entry:
		`thinreg.exe /u "\\server\share\Microsoft Office *.exe"`

/r, /reregister	Re-register a package. Under typical circumstances, thinreg.exe can detect whether a package is already registered and skips it. The /r option forces `thinreg.exe` to re-register the package.	`thinreg.exe /r "\\<server>\<share>\Microsoft Office 2007 Word.exe"`

Table 3-1. Optional thinreg.exe parameters (Continued)

Parameter	Purpose	Sample Usage
`/k`, `/keepunauthorized`, `/keep`	Prevents the removal of registration information even if you are no longer authorized to access an application package. Without this option, thinreg.exe 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 Wword.exe"`
`/norelaunch`	Launches `thinreg.exe` on Microsoft Vista without elevated privileges. Standard users can invoke the utility without a UAC pop-up winder. When `thinreg.exe` detects a need for more privileges, such as the privileges required for the `/allusers` parameter, the utility relaunches itself as a privileged process and generates a UAC pop-up window. The `/norelaunch` option blocks this relaunch 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 Setup Capture wizard, 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 having shortcuts and file type associations automatically created. Basic Active Directory group policies provide ways to push and launch MSI packages.

ThinApp creates an MSI database that contains certain files depending on the size:

- For databases smaller than 2GB, the MSI database consists of captured executable files, installer logic, and the `thinreg.exe` utility.
- For databases larger than 2GB, the MSI database consists of installer logic and the`thinreg.exe` utility. ThinApp stores the captured executable files in cabinet files. For example, the files might be `<inventoryname>_1.CAB` and `<inventoryname>_2.CAB`. The `.CAB` files must be in the same directory as the MSI files. ThinApp must distribute these files with the MSI file to have a complete installer.

Customizing MSI Files with Package.ini Parameters

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

- The `MSIInstallDirectory` parameter sets the installation directory for the package.

  For example, include this line in the `Package.ini` file:

  ```
  MSIInstallDirectory=C:\Program Files\
  ```

- The `MSIDefaultInstallAllUsers` parameter sets a per-user installation of the package. ThinApp installs the package in the user's `%AppData%` directory.

  For example, include this line in the `Package.ini` file:

  ```
  MSIDefaultInstallAllUsers=0
  ```

  For more information on this parameter, see "Specifying Per-User and Per-Machine Database Installation" on page 39.

- The `MSIFileName` parameter names the package.

  For example, include this line in the `Package.ini` file:

  ```
  MSIFilename=Firefox30.msi
  ```

- The `MSIRequireElevatedPrivileges` parameter indicates whether an installer needs elevated privileges for deployment on Microsoft Vista. Per-user installation do not usually need elevated privileges but per-machine installations require such privileges.

  For example, include this line in the `Package.ini` file:

  ```
  MSIRequireElevatedPrivileges=1
  ```

- The `MSIProductCode` parameter makes it easier to install a new version of the application. An MSI database contains a number of codes, including 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 value specified at build time, you can still override the settings at deployment time. See "Force a Per-User or Per-Machine MSI Deployment" on page 39.

Modify the Package.ini File to Create MSI Files

You must enter a value for the `MSIFilename` parameter to generate MSI files. For more information about MSI parameters, see "Customizing MSI Files with Package.ini Parameters" on page 37 and "MSI Generation" on page 121.

To modify the Package.ini to create MSI files

1 In the `Package.ini` file, enter the MSI filename:

  ```
  MSIFilename=<filename>.msi
  ```

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

  You must enter a value for this parameter to generate MSI files.

2 (Optional) Update other MSI parameters.

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

Specifying Per-User and Per-Machine Database Installation

ThinApp installs the MSI database across all machines. You can change the default installation:

- If you need to create a database with a per-user installation, use a value of 0 for the `MSIDefaultInstallAllUsers` parameter in the `Package.ini` file. This creates per-user `msiexec` parameters.
- If you need to create a database with a a per-machine installation for administrators and per-user installation for other users, use a value of 2 for the `MSIDefaultInstallAllUsers` parameter. Administrators belong to the Administrators Active Directory group.

Force a Per-User or Per-Machine MSI Deployment

Regardless of the parameter value specified at build time, you can still override the settings at deployment time. For example, if you created the database with a value of 1 for the `MSIDefaultInstallAllUsers` parameter, you can still force a per-user deployment 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 user's `%AppData%` directory.

To force a per-user MSI deployment

From the command line, type:

```
msiexec /i <database>.msi ALLUSERS=""
```

To force a per-machine deployment

From the command line, type:

```
msiexec /i <database>.msi ALLUSERS=1
```

Override the MSI Installation Directory

When ThinApp performs a per-machine MSI deployment, the default installation directory is the localized equivalent of `%ProgramFilesDir%\<inventory name> (VMware ThinApp)`. If you install a Firefox package per machine, the package resides in `%ProgramFilesDir%\Mozilla Firefox (VMware ThinApp)`.

When ThinApp performs a per-user MSI deployment, 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.

To override the MSI installation directory

From the command line, type:

```
msiexec /i <database>.msi INSTALLDIR=C:\<my_directory>\<my_package>
```

Deploying MSI Files on Microsoft Vista

When you deploy MSI files on Vista, you must indicate whether an installer needs elevated privileges. Typical per-user installations do not require elevated privileges but per-machine installations require such privileges. ThinApp provides the `MSIRequireElevatedPrivileges` parameter in the `Package.ini` file that specifies the need for elevated privileges when set to 1. Specifying a value of 1 for this parameter or forcing a per-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 SID values are stored in packages for future validation, the following considerations apply:

- You must be connected to your Active Directory domain during the build process and the groups you specified must exist. ThinApp looks up the SID value during build.
- If you delete a group and recreate it, the SID might change. 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 user can log into his or her machine, authentication still works. Use a group policy to set the period of time 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 settings and group policy and security settings stored in Active Directory. You might need to log off 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.

Reviewing 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 launch 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.

In this sample entry from the `Package.ini` file, App1 and App2 will both inherit `PermittedGroups` values:

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

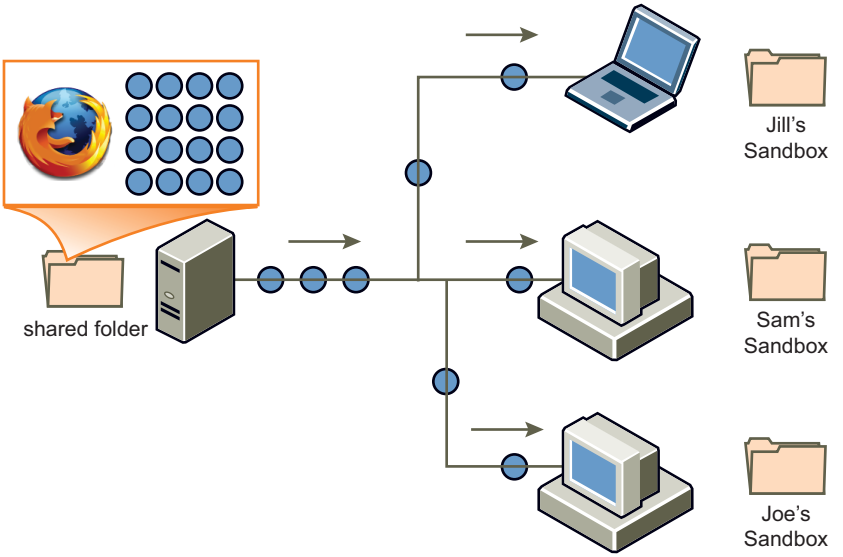
In this sample entry, only users belonging to the `App1users` group can use `App1.exe`, and members of the `Everyone` group can use `App2.exe`. The default message for denied users changes for App1.

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

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. Block-Based Streaming over a Network Share



On the end user desktop, you can create shortcuts that point to the centrally hosted executable packages. When the user clicks on 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.

When the application can begin running, the status bar slides down and you can use the application.

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. The size of the package does

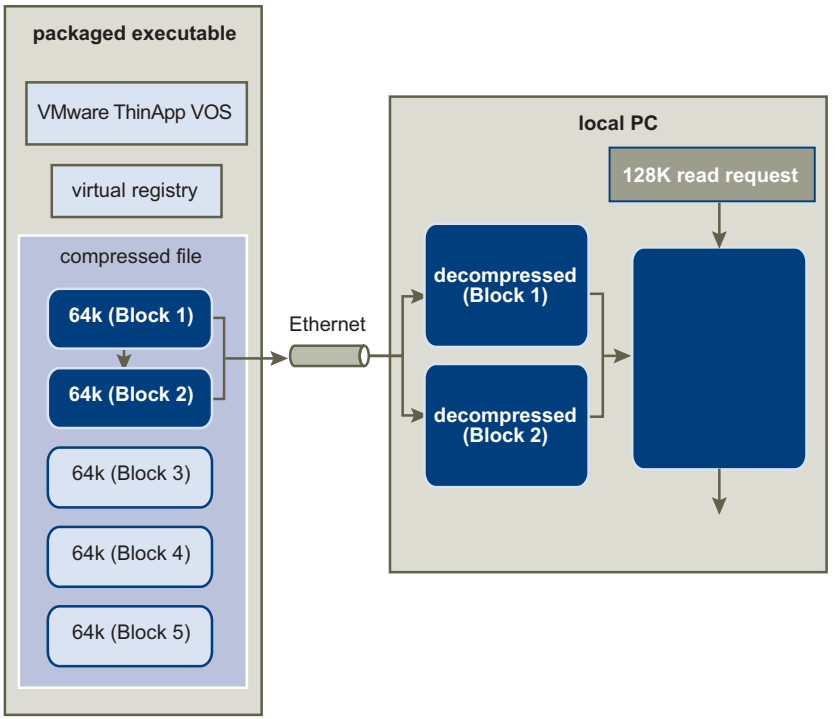
not affect the startup time of an application. If you add an extra 20GB file to a package that is not used at runtime, the package loads at the same speed. If the application opens and reads 32KB of data from the 20GB file, ThinApp only requests 32KB of data.

The ThinApp runtime client is a small part of the executable package. When ThinApp loads the runtime client, it sets up the environment and starts the target executable. The target executable 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



Reviewing 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 on LAN-based environments with a minimum of 100MB networks.

For WAN and Internet deployments that involve frequent or unexpected disconnections, VMware recommends one of these solutions:

- Deploy applications by using a URL.
- 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 the branch office.

Reviewing 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 contents. As multiple users stream a package from a shared location, ThinApp stores changes that the application makes in the user's 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's sandbox on another central storage device. The user can use any computer and retain individual application settings at a central share. When packages stream from a central share, they remain locked until all users exit the application.

Stream Packages from the Network

Users can access packaged applications through the network.

To stream ThinApp 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 applications installed on the desktop.

Performing Cut and Paste Operations

Review these cut and paste scenarios and limitations with ThinApp:

- **Pasting from system installed applications to captured application**s – This scenario has no limitations. 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 they are pasted into native applications.

Accessing Printers

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

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

Accessing Drivers

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

In some cases, an application without an associated driver might function with some limitations. For example, Adobe Acrobat installs a printer driver that allows 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 installation. Other applications do not detect a new printer driver unless the driver is installed.

Accessing the Local Disk, Removable Disk, and Network Shares

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

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

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, all registry modifications from captured applications are saved in an isolated sandbox and the system remains unchanged.

Networking and Sockets

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

Shared Memory Named Pipes

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

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

COM, DCOM, and Out-of-Process COM

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

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.

File Type Associations

Captured applications can execute system-installed applications using file type association. You can add file type association to the local computer registry to point to captured executable files on a per-user or per-machine basis.

Sample Isolation Mode Configuration Depending on the Deployment Context

Isolation modes control the read and write access for specific system directories and system registry subkeys. See "Modifying Isolation Modes" on page 26.

You can adjust isolation modes to resolve the following problems in Table 3-2.

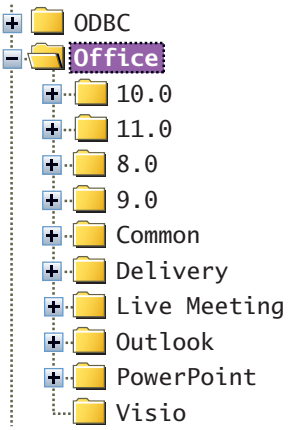
Table 3-2. Sample Problems and Solutions using Isolation Modes

Problem	Solution
An application fails to run because of previous or future versions existing simultaneously or not uninstalling correctly.	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's installer creates. This situation involves full isolation. For directories and subkeys that have full isolation, the applications only see virtual files and subkeys. Any system values that exist in the same location are invisible to the application.
An application fails because no one designed or tested it for a multiuser environment. The application fails to modify files and keys without impacting other users.	ThinApp makes copies of registry keys and files that the application writes and performs all the modifications in a user-specific sandbox. This situation involves WriteCopy Isolation. For directories and subkeys that have WriteCopy isolation, the application recognizes both the host computer files and virtual files. All write operations convert host computer files into virtual files in the sandbox.
An application fails because it should have write permission to global locations and is not designed for locked-down desktop environments found in a corporate environment or on Windows Vista.	

View of Isolation Mode Impact 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.

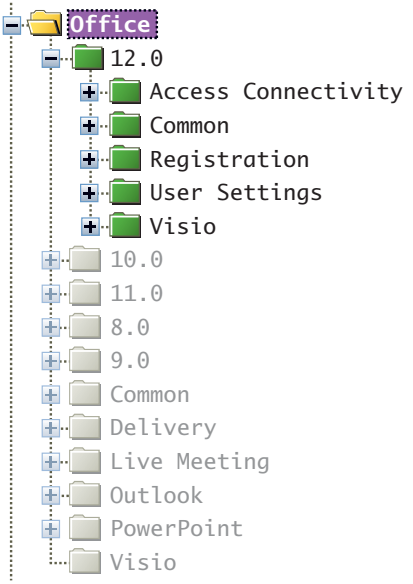
Figure 3-3. Windows Registry as seen by Windows Regedit



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 Applications 4

You can update captured applications with different utilities depending on the extent of change and dependencies on other applications. This chapter includes these topics:

Reviewing Application Updates Triggered by the End-User

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

Reviewing the Application Sync Utility

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 see if an updated version of the executable 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 need to update Firefox to the next major version.

Using Application Sync in a Managed or Unmanaged Environment

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

If you have an unmanaged environment that does not make use of application auto-update capabilities, use the Application Sync utility to update applications.

Edit Application Sync Parameters in the Package.ini File

You can configure the Application Sync utility by editing 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 "Application Sync" on page 118,

To edit Application Sync parameters

1 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`.

2 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.

3 Change the value of the parameters and save the file.

For example, you can copy an executable of the latest Firefox version to a mapped network drive and enter 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, a sample path is `file:///Z:/Firefox/Firefox.exe`.

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`.

If you launch the new executable in the `\bin` directory, such as `Mozilla Firefox 3.exe`, ThinApp automatically converts the original application to the new version. If you cannot see the update and you originally copied the update to a network drive, verify the connection to that drive.

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.

To fix an incorrect update with Application Sync

Place a desired update on the server that ThinApp can pick up the next time someone launched the application on a client machine.

Application Sync Impact 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. For more information about the primary data container, see Step 10 in "Capture an Application with the Setup Capture Wizard" on page 20.

Reviewing the Application Link Utility

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.

The Application Link utility is useful for these 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 does not allow for the installation of .NET or already has a different version of .NET.

- **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.

  The Application Link utility allows you to deploy a single virtualized Microsoft Office to all users and deploy individual add-on components for each user.

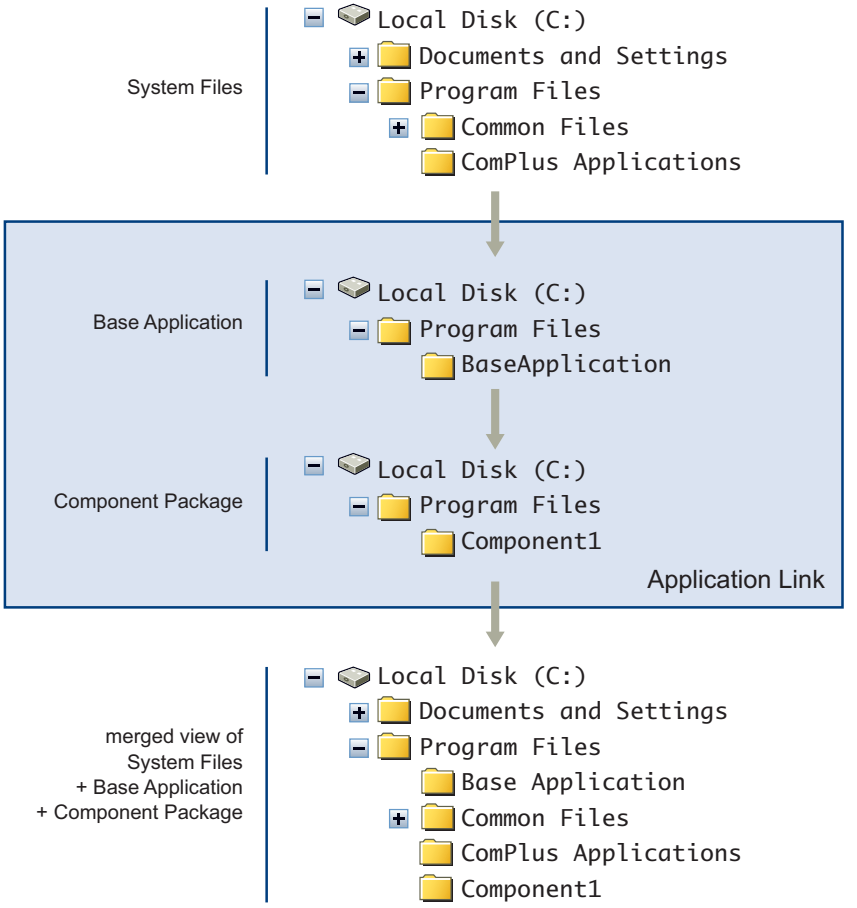
- **Hot fixes and service packs** – Link updates to an application and roll back to a previous version if users experience significant issues with the new version. You can deploy minor patches to applications as a single file and reduce the need for rollbacks.

A key benefit is 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 need to 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.

High-Level View of the Application using Application Link

The Application Link utility presents 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. See Figure 4-1.

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 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 19.

For information about required and optional Application Link parameters in the `Package.ini` file, see "Application Link" on page 115.

To link a base application to Microsoft .NET Framework

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 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.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 this line after the `[BuildOptions]` entry:

```
RequiredAppLinks=dotnet.dat
```

Application Link parameters need to 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 that requires AppB and AppB that requires AppC. Assume this 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 "Application Link" on page 115.

To set up nested links with Application Link

1 Capture Application A.

2 In the `Package.ini` file, specify Application B as a required or optional application link.

For example, type this line:

```
RequiredLinks=\AppFolder\AppB\AppB.exe
```

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, type this line:

```
RequiredLinks=\AppFolder\AppC\AppC.exe
```

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.

Impacting Isolation Modes with Application Link

ThinApp loads an Application Link layer during application startup and examines 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 an already merged layer, ThinApp uses the isolation mode specified in the layer being loaded.

If the registry subkey or file system directory already exist already in the main package or an already merged layer, 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.

Reviewing the PermittedGroups Impact on Application Link

If you specify a value for the `PermittedGroups` parameter, the user account used for launching the application must be a member of at least one of the Active Directory groups for this parameter in both the Firefox `Package.ini` file and the Adobe Reader `Package.ini` file. For information about these parameters, see "Access Control" on page 109.

Reviewing File and Registry Collisions in Linked Packages

If the base application contains a file or registry entry at the same location as a dependent package, a collision occurs. When this happens, the order of import operations determines which package wins. The last package imported wins in such cases and the file or registry contents from that package are visible to the running application. ThinApp imports applications according to the `RequiredAppLinks` or `OptionalAppLinks` parameter. If either parameter specifies a wildcard character that matches more than one file, alphabetical order determines which package is imported first.

If two or more packages include VB scripts, the run order for the scripts is alphabetical by package without regard to order of import operations. Use unique file names for VB scripts. VB script name collisions might prevent scripts from other imported packages from running. If two packages contain a VB script with the same name, ThinApp runs only the version of the VB script from the last imported package.

The `OptionalAppLinks` parameter might appear as:

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

In this example using the `a.exe` and `b.exe` executables, ThinApp imports these elements in this order:

- Base application
- `a.exe`
- `b.exe`
- plug-ins loaded in alphabetical order
- nested plug-ins for `a.exe`
- nested plug-ins for `b.exe`

- nested plug-ins for first set of plug-ins above

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 the Base Application and Linked Packages

If you use Application Link to link packages to a base package, you can use Application Sync to update all packages. For example, if you have Microsoft Office 2007 and link Adobe Reader, you can have Application Sync entries in the Package.ini files for both packages to update them. If any linked package fails to download or is expired, ThinApp terminates the linked packages and the main application.

Review Application Updates Triggered by the Administrator

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 change the home page of a Web site to point to a different default site.

Force an Application Sync Update with AppSync.exe

You can use `AppSync.exe` 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 the Application Sync parameters to check for updates upon launch of the application because the 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 `AppSync.exe`.

To force an application sync update from the command line

Type this command:

```
AppSync <Application_Sync_URL> <executable_path>
```

The value of the URL is the same as the Application Sync URL in the `Package.ini` file and the executable path is the path to the executable that requires the update.

Reviewing the sbmerge.exe Workflow

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

This procedure uses Firefox 2.0.0.3 as an example of the captured application.

To merge the sandbox changes with the Firefox application

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. Launch Firefox and make a change to the settings.

   For example, change the home page.

5 From the command prompt, 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 Type this command:

```
"C:\Program Files\VMware\VMware ThinApp\sbmerge" Print
```

ThinApp prints the changes that affected the sandbox folder when using the captured application.

7 Type this command:

```
"C:\Program Files\VMware\VMware ThinApp\sbmerge" Apply
```

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

sbmerge.exe Commands

ThinApp provides the `sbmerge.exe Print` and `sbmerge.exe Apply` commands.

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

`Apply` 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

Parameter	Description
`–ProjectDir <project_path>`	If you start the `sbmerge.exe` command from a location other than the application project folder, use the absolute or relative path to the project directory using the `–ProjectDir <project_path>` parameter. A sample command is "`C:\Program Files\VMware\VMware ThinApp\sbmerge" Print –ProjectDir "C:\<project_folder_path>""`.
`–SandboxDir <sandbox_path>`	When you start a captured application, it searches for the sandbox in a particular order. See "Reviewing the Search Order for the Sandbox" on page 129. If you use a custom location for the sandbox, use the `–SandboxDir <sandbox_path>` parameter to specify the location.
`–Quiet`	Blocks the printing of progress messages.
`–Exclude <excluded_file>.ini`	Prevents the merging of specific files or registry entries from the sandbox. You can specify a `.ini` file to determine the content for exclusion. This file contains separate sections to specify files, such as the `FileSystemIgnoreList` and the `RegistryIgnoreList`. The `sbmerge.exe` utility uses the `snapshot.ini` file in the ThinApp installation folder by default to exclude certain content from the merge process. This option allows you to specify another `.ini` file to ensure additional exclusion of content.

Reviewing Application Updates using Auto-Update Capabilities

If an application has auto-update capabilities, its update mechanism still 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 in the sandbox and not from the original package.

For example, if you capture Firefox 1.5, your auto update 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.

If an administrator need to merge changes made by an auto-update mechanism with the original package to build an updated executable, use the `sbmerge.exe` utility. See "Review Application Updates Triggered by the Administrator" on page 59.

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 52.

Benefits of 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 need to execute `ngen.exe` to generate native image assemblies for startup performance. In typical circumstances, `ngen.exe` writes to `HKLM` and `C:\WINDOWS`, both of which are only accessible with administrator accounts. With ThinApp, `ngen.exe` 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 central network shares as a new captured executable file. Review the following options for applying updates:

- Apply updates during the setup capture process.
- Apply updates 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` entry point. Changes occur in the sandbox during auto-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 merge sandbox changes made by the update with the application using the sbmerge.exe utility. See "Review Application Updates Triggered by the Administrator" on page 59.

- Apply updates by replacing files in the captured project.

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

Reviewing File Locks

Launching an application locks the executable 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 sub-processes are running.

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

Upgrade a Running Application

You can copy a the 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 need to update shortcuts.

To 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 location at `\\<server>\<share>\Firefox.exe`.

Assume two users launch `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 launches 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 exiting the application and restarting the application.

5 If you need to deploy a more current update of Firefox, place it in the same directory with a higher number at the end.

Copy Version 2.0 of `Firefox.exe` to central share at `\\<server>\<share>\Firefox.2`

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

Sandbox Considerations for Upgraded Applications

When you upgrade an application, you can control whether users continues to use their previous settings by keeping the sandbox name consistent in the `Package.ini` file. If you do not want users to use an older sandbox with an upgraded application, you can package the upgraded application with a new name for the sandbox. The first launch of the upgraded application creates the sandbox with the new name.

Monitoring and Troubleshooting ThinApp

5

You can monitor and troubleshoot the environment. This chapter includes these topics:

Providing Information to VMware Support

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

- Step-by-step reproduction of the procedure you performed when you encountered the problem.
- Information on the host configuration. Include the Windows operating system you use, whether you use Terminal Server and Citrix Xenapp, and any prerequisite programs that you installed on the native machine.
- Copies of the Log Monitor trace files. See "Using Log Monitor" on page 68.
- Exact copy of the capture folder and all content. Do not include the compiled executables from the `/bin` subfolder.
- Description of the expected correct 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.

Using Log Monitor

Log Monitor captures detailed chronological activity for one or more executable files invoked by the captured application. 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

Review the basic procedure to troubleshoot ThinApp activity with Log Monitor.

To troubleshoot activity with Log Monitor

1 Shut down the captured application that you need to investigate.

2 On the computer where you captured the application, go to **Start** > **Programs** > **VMware** > **ThinApp Log Monitor**.

If you want to start Log Monitor on a deployment machine, copy the `log_monitor.exe` file from `C:\Program Files\VMware\VMware ThinApp` to the deployment machine.

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 Terminate 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 the **Generate text trace report** button.

   In typical circumstances, child processes spawned by the parent process 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, such as TextPad, 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 them to VMware support.

Log Monitor Options

You can perform operations such as stopping an application or deleting a trace file.

For applications that are busy or experiencing slow performance with a specific action, you can suspend and resume Log Monitor to capture logs for a specific duration. The resulting log file is smaller than the typical log file and easier to analyze. Even with the suspend and resume options, the root cause of an error might occur outside of your duration window. Suspend and resume options are global and affect all applications.

For more information on whether or not to use these options, contact VMware Support.

To perform advanced operations in Log Monitor

1 Shut down the captured application that you need to investigate.

2 On the computer where you captured the application, go to **Start** > **Programs** > **VMware** > **ThinApp Log Monitor**.

   If you want to start Log Monitor on a deployment machine, copy the `log_monitor.exe` file from `C:\Program Files\VMware\VMware ThinApp` to the deployment machine.

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.

c In Log Monitor, deselect the **Suspend** check box to resume the logging process.

d Select the **Suspend** check box to stop the logging process.

You might use the **Suspend** button when an application is running slowly and you want to suspend the logging until the performance problem starts, resume logging, and check the application behavior to isolate the issue.

4 (Optional) Click the **Delete File** button to remove any `.trace` files selected in the file list.

5 (Optional) Click the **Kill App** button to stop a running process.

6 (Optional) Click the **Compress** check box to compress a `.trace` file.

This operation slows the performance of the application.

7 (Optional) Select a trace file in the file list, enter a a trace filename, or click the **`Browse`** button to select a trace file on your system.

8 (Optional) Enter or change the name of the output report.

This file is generated by the **Generate text trace report** button. 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:

- Look at the `Potential Errors Detected` section of the `.txt` trace file.

  Many entries do not indicate errors. ThinApp lists each Win32 API call where the Windows error code changed.

- Look at exceptions generated by applications.

  Exceptions might indicate errors. Exceptions have different types, such as C++ and .NET. The trace file records the type of exception and the DLL that generated the exception. If the application created an exception from self-generating code, as might be the case for .NET and Java applications, the trace indicates `unknown_module`.

  Here is a sample `.trace` entry for an exception:

```
*** Exception EXCEPTION_ACCESS_VIOLATION on read of 0x10 from
unknown_module:0x7c9105f8
```

If you find an exception, look higher in the trace file for the source of the exception. Because Virtual Basic 6 applications throw many floating point exceptions during normal execution, you can ignore such exceptions.

- Look at 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. In some cases, such as out-of-process COM, a parent application uses COM to launch a child process, executes a function remotely, and continues executing functions.

- When you run applications from a network share that generates two processes, ignore the first one.

  To work around the slow performance of Symantec antivirus applications, ThinApp relaunches processes when running from a network share.

- Search for the error message you see in dialog boxes.

  Many applications call the Win32 API function MessageBox to display unexpected errors at runtime. You can search your trace file for `MessageBox` or the contents of the string displayed in the error and determine what the application was executing just before the dialog box appeared.

- Narrow the focus on calls originating from a specific DLL and thread.

  The log format specifies which DLL and thread is making a call. You can often ignore calls from system DLLs.

Log Format

A trace file involves four sections:

- **System configuration**. This section includes information about the operating system, drives, installed software, environment variables, process list, services, and drivers.

  This section 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 under Log Monitor. Some of the displayed attributes show logging options, address ranges where the operating system runtime is loaded, and mapping of macros to actual system paths.

  ThinApp marks the beginning of this 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 consists of trace activity as the application starts and executes operations. Each line represents function calls made by target executable or one of its DLLs.

  The beginning of this section is marked by `New Modules detected in memory` followed by the `SYSTEM_LOADED` modules list. The end of this section is marked by `Modules Loaded`.

- **Summary**. This section list modules loaded by the captured application, potential errors, profile of 150 slowest calls, and more.

  The beginning of this section is noted by the `Modules Loaded` message.

General API Log Message Format

The general format for API calls resembles the following message:

```
000257 0a88 mydll.dll :4ad0576d–>kernel32.dll:7c81b1f0 SetConsoleMode (IN
HANDLE
hConsoleHandle=7h, IN DWORD dwMode=3h)
000258 0a88 mydll.dll :4ad0576d<–kernel32.dll:7c81b1f0 SetConsoleMode
–>BOOL=1h ()
```

`000257`—Indicates the log entry number. Each log entry has a unique number. The "----Potential Errors Detected ---" refers to individual log entries using their log entry numbers.

`0a88`—Indicates the currently executing thread ID. If the application has only one thread, this number does not change.. Because log entries are recorded in order of execution, if two or more threads are recording data to the log file, you might need to use the thread ID to follow thread-specific sequential actions.

`mydll.dll`—Indicates the DLL that is making the API call.

`4ad0576d`—Indicates the return address for the API call made by mydll.dll. Typically this tells you the address in the code where the call is originating.

`–>` Indicates the call is being entered. For call entry log element, all of the input parameters are displayed (in and in/out parameters)

<–Indicates the call is returning to the original caller. For call exit log entries, all of the output parameters are displayed (out and in/out parameters).

`kernel32.dll`—Indicates the DLL where the API call is landing.

`7c81b1f0`—Indicates the address of the API inside of kernel32 where the call is landing. If you disassemble kernel32.dll at address 7c81b1f0, you find the code for the function SetConsoleMode

`–>BOOL=1h` Indicates the API returned the value 1 and the return code has type BOOL.

Application Startup Information

```
000001 0a88 Logging started for Module=C:\test\cmd_test\bin\cmd.exe
Using archive=
PID=0xec
CommandLine = cmd
000002 0a88 Logging options: CAP_LEVEL=9 MAX_CAP_ARY=25 MAX_CAP_STR=150
MAX_NEST=100
VERSION=3.090

000003 0a88 System Current Directory = C:\test\cmd_test\bin Virtual Current
Directory = C:\test\cmd_test\bin

000004 0a88 |start_env_var| =::=::\
000005 0a88 |start_env_var| =C:=C:\test\cmd_test\bin
000006 0a88 |start_env_var| =ExitCode=00000000
000007 0a88 |start_env_var| ALLUSERSPROFILE=C:\Documents and Settings\All
Users.WINDOWS
...
...
...
```

List of DLLs Loaded into Memory During Runtime

The section labeled "---Modules loaded ---" is located near the end of the log and describes all of the DLLs that were loaded into memory at runtime and the addresses they were loaded to. This list also describes whether DLLs were loaded by Windows or by ThinApp.

```
---Modules loaded --
PRELOADED_MAP 00400000-00452fff, C:\Program Files\Adobe\Reader
8.0\Reader\AcroRd32.exe
PRELOADED_BY_SYSTEM 00400000-00452fff, C:\Program Files\Adobe\Reader
8.0\Reader\AcroRd32.exe
SYSTEM_LOADED 00400000-00452fff, C:\test\AcroRd32.exe
SYSTEM_LOADED 00df0000-00df8fff, C:\WINDOWS\system32\Normaliz.dll
MEMORY_MAPPED_ANON 013b0000-020affff, C:\Program Files\Adobe\Reader
8.0\Reader\AcroRd32.dll
SYSTEM_LOADED 035a0000-035b4fff, C:\WINDOWS\system32\nvwddi.dll
SYSTEM_LOADED 035a0000-03698fff, C:\WINDOWS\system32\nvwimg.dll
SYSTEM_LOADED 035e0000-03ba9fff, C:\WINDOWS\system32\ieframe.dll
SYSTEM_LOADED 04730000-04828fff, C:\WINDOWS\system32\nvwimg.dll
MEMORY_MAPPED_ANON 05000000-050a8fff, C:\Program Files\Adobe\Reader
8.0\Reader\ACE.dll
MEMORY_MAPPED_ANON 06000000-064b0fff, C:\Program Files\Adobe\Reader
8.0\Reader\AGM.dll
MEMORY_MAPPED_ANON 07000000-07019fff, C:\Program Files\Adobe\Reader
8.0\Reader\BIB.dll
MEMORY_MAPPED_ANON 08000000-08239fff, C:\Program Files\Adobe\Reader
```

```
8.0\Reader\CoolType.dll
SYSTEM_LOADED 0ffd0000-0fff7fff, C:\WINDOWS\system32\rsaenh.dll
SYSTEM_LOADED 10000000-10165fff, C:\WINDOWS\system32\nview.dll
SYSTEM_LOADED 20000000-202c4fff, C:\WINDOWS\system32\xpsp2res.dll
 MEMORY_MAPPED_ANON 20800000-20fdafff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\AcroForm.api
MEMORY_MAPPED_ANON 22100000-224f2fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Annots.api
MEMORY_MAPPED_ANON 23000000-2311afff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\DigSig.api
MEMORY_MAPPED_ANON 23800000-23951fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\EScript.api
MEMORY_MAPPED_ANON 24000000-24023fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\EWH32.api
MEMORY_MAPPED_ANON 25800000-25817fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\IA32.api
MEMORY_MAPPED_ANON 26800000-2680ffff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\eBook.api
MEMORY_MAPPED_ANON 28000000-285d1fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\PPKLite.api
MEMORY_MAPPED_ANON 28800000-2885dfff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\reflow.api
MEMORY_MAPPED_ANON 29000000-29227fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\MakeAccessible.api
MEMORY_MAPPED_ANON 29800000-2985afff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Accessibility.api
MEMORY_MAPPED_ANON 29a00000-29a1dfff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\ReadOutLoud.api
MEMORY_MAPPED_ANON 2a000000-2a018fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Search5.api
MEMORY_MAPPED_ANON 2a300000-2a359fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Search.api
MEMORY_MAPPED_ANON 2a800000-2a821fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\SendMail.api
MEMORY_MAPPED_ANON 2b000000-2b045fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Spelling.api
MEMORY_MAPPED_ANON 2b800000-2b865fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\PDDom.api
MEMORY_MAPPED_ANON 2d800000-2d94efff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Multimedia.api
MEMORY_MAPPED_ANON 2e000000-2e02ffff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\weblink.api
MEMORY_MAPPED_ANON 30800000-30829fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Updater.api
MEMORY_MAPPED_ANON 31800000-31810fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\HLS.api
MEMORY_MAPPED_ANON 32000000-3204dfff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\SaveAsRTF.api
MEMORY_MAPPED_ANON 40800000-40821fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\DVA.api
```

```
MEMORY_MAPPED_ANON 45800000-458cffff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\Checkers.api
MEMORY_MAPPED_ANON 46800000-46873fff, C:\Program Files\Adobe\Reader
8.0\Reader\plug_ins\ImageViewer.API
SYSTEM_LOADED 59a60000-59b00fff, C:\WINDOWS\system32\dbghelp.dll
SYSTEM_LOADED 5ad70000-5ada7fff, C:\WINDOWS\system32\uxtheme.dll
SYSTEM_LOADED 5d090000-5d129fff, C:\WINDOWS\system32\COMCTL32.dll
SYSTEM_LOADED 61410000-61533fff, C:\WINDOWS\system32\urlmon.dll
SYSTEM_LOADED 629c0000-629c8fff, C:\WINDOWS\system32\LPK.DLL
SYSTEM_LOADED 63380000-633f7fff, C:\WINDOWS\system32\jscript.dll
SYSTEM_LOADED 6e850000-6e894fff, C:\WINDOWS\system32\iertutil.dll
SYSTEM_LOADED 71aa0000-71aa7fff, C:\WINDOWS\system32\WS2HELP.dll
SYSTEM_LOADED 71ab0000-71ac6fff, C:\WINDOWS\system32\ws2_32.dll
SYSTEM_LOADED 71bf0000-71c02fff, C:\WINDOWS\system32\SAMLIB.dll
SYSTEM_LOADED 746c0000-746e8fff, C:\WINDOWS\system32\msls31.dll
SYSTEM_LOADED 746f0000-74719fff, C:\WINDOWS\system32\msimtf.dll
SYSTEM_LOADED 74720000-7476afff, C:\WINDOWS\system32\MSCTF.dll
SYSTEM_LOADED 74d90000-74dfafff, C:\WINDOWS\system32\USP10.dll
SYSTEM_LOADED 755c0000-755edfff, C:\WINDOWS\system32\msctfime.ime
SYSTEM_LOADED 75cf0000-75d80fff, C:\WINDOWS\system32\MLANG.dll
SYSTEM_LOADED 75e90000-75f3ffff, C:\WINDOWS\system32\SXS.DLL
SYSTEM_LOADED 76390000-763acfff, C:\WINDOWS\system32\IMM32.DLL
SYSTEM_LOADED 76780000-76788fff, C:\WINDOWS\system32\shfolder.dll
SYSTEM_LOADED 769c0000-76a72fff, C:\WINDOWS\system32\USERENV.dll
SYSTEM_LOADED 76b40000-76b6cfff, C:\WINDOWS\system32\WINMM.dll
SYSTEM_LOADED 76bf0000-76bfafff, C:\WINDOWS\system32\PSAPI.DLL
SYSTEM_LOADED 76f60000-76f8bfff, C:\WINDOWS\system32\WLDAP32.dll
SYSTEM_LOADED 76fd0000-7704efff, C:\WINDOWS\system32\CLBCATQ.DLL
SYSTEM_LOADED 77050000-77114fff, C:\WINDOWS\system32\COMRes.dll
SYSTEM_LOADED 77120000-771abfff, C:\WINDOWS\system32\OLEAUT32.dll
SYSTEM_LOADED 771b0000-7727efff, C:\WINDOWS\system32\WININET.dll
SYSTEM_LOADED 773d0000-774d2fff,
C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.
2600.2982_x-ww_ac3f9c03\comctl32.dll
SYSTEM_LOADED 774e0000-7761cfff, C:\WINDOWS\system32\ole32.dll
SYSTEM_LOADED 77690000-776b0fff, C:\WINDOWS\system32\NTMARTA.DLL
SYSTEM_LOADED 77920000-77a12fff, C:\WINDOWS\system32\SETUPAPI.dll
SYSTEM_LOADED 77b40000-77b61fff, C:\WINDOWS\system32\appHelp.dll
SYSTEM_LOADED 77c00000-77c07fff, C:\WINDOWS\system32\VERSION.dll
SYSTEM_LOADED 77c10000-77c67fff, C:\WINDOWS\system32\msvcrt.dll
SYSTEM_LOADED 77dd0000-77e6afff, C:\WINDOWS\system32\ADVAPI32.dll
SYSTEM_LOADED 77e70000-77f00fff, C:\WINDOWS\system32\RPCRT4.dll
SYSTEM_LOADED 77f10000-77f56fff, C:\WINDOWS\system32\GDI32.dll
SYSTEM_LOADED 77f60000-77fd5fff, C:\WINDOWS\system32\SHLWAPI.dll
SYSTEM_LOADED 77fe0000-77ff0fff, C:\WINDOWS\system32\Secur32.dll
MEMORY_MAPPED_ANON 78130000-781cafff,
C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.163_x-
ww_681e29fb\msvcr80.dll
MEMORY_MAPPED_ANON 7c420000-7c4a6fff,
```

```
C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.163_x-
ww_681e29fb\msvcp80.dll
MEMORY_MAPPED_ANON 7c4c0000-7c53cfff,
C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.163_x-ww_
681e29fb\msvcm80.dll
SYSTEM_LOADED 7c800000-7c8f3fff, C:\WINDOWS\system32\kernel32.dll
SYSTEM_LOADED 7c900000-7c9affff, C:\WINDOWS\system32\ntdll.dll
SYSTEM_LOADED 7c9c0000-7d1d4fff, C:\WINDOWS\system32\SHELL32.dll
SYSTEM_LOADED 7e410000-7e49ffff, C:\WINDOWS\system32\USER32.dll
SYSTEM_LOADED 7e830000-7eb9efff, C:\WINDOWS\system32\mshtml.dll
```

SYSTEM_LOADED—Indicates the DLL was loaded by Windows, the file must exist on the disk.

MEMORY_MAPPED_ANON—Indicates the DLL was loaded by ThinApp, the file might be loaded from the virtual file system.

46800000-46873fff—Indicates the address ranges in virtual memory where the DLL was loaded.

```
----Timing Report: list of slowest 150 objects profiled ---

8255572220 total cycles (2955.56 ms): |sprof| thinapp_LoadLibrary2

765380728 cycles (274.01 ms) on log entry 21753
428701805 cycles (153.48 ms) on log entry 191955
410404281 cycles (146.93 ms) on log entry 193969
231503734 cycles (82.88 ms) on log entry 188438
227419794 cycles (81.42 ms) on log entry 190209
211952538 cycles (75.88 ms) on log entry 197416
202095103 cycles (72.35 ms) on log entry 189394
200356604 cycles (71.73 ms) on log entry 194646
192420627 cycles (68.89 ms) on log entry 190812
183214731 cycles (65.59 ms) on log entry 195836
... 438 total calls
7847975891 total cycles (2809.64 ms): |sprof| ts_load_internal_module
764794646 cycles (273.80 ms) on log entry 21753
426837866 cycles (152.81 ms) on log entry 191955
408570540 cycles (146.27 ms) on log entry 193969
228790905 cycles (81.91 ms) on log entry 188438
224240114 cycles (80.28 ms) on log entry 190209
209789307 cycles (75.11 ms) on log entry 197416
200287437 cycles (71.70 ms) on log entry 189394
198429210 cycles (71.04 ms) on log entry 194646
190612618 cycles (68.24 ms) on log entry 190812
180322247 cycles (64.56 ms) on log entry 195836
... 94 total calls
4451728477 total cycles (1593.76 ms): |sprof| ts_lookup_imports
544327945 cycles (194.87 ms) on log entry 21758
385149968 cycles (137.89 ms) on log entry 193970
187246661 cycles (67.04 ms) on log entry 190210
```

```
173617241 cycles (62.16 ms) on log entry 194647
173481875 cycles (62.11 ms) on log entry 19065
148587579 cycles (53.20 ms) on log entry 195837
133165053 cycles (47.67 ms) on log entry 189395
126806624 cycles (45.40 ms) on log entry 197417
125894370 cycles (45.07 ms) on log entry 200296
121213253 cycles (43.40 ms) on log entry 200657
... 34 total calls
1099873523 total cycles (393.76 ms): |sprof| new_thread_start
561664565 cycles (201.08 ms) on log entry 151922
531551734 cycles (190.30 ms) on log entry 152733
1619002 cycles (0.58 ms) on log entry 72875
1554448 cycles (0.56 ms) on log entry 637896
1481627 cycles (0.53 ms) on log entry 72881
1091972 cycles (0.39 ms) on log entry 580771
```

Potential Errors

The potential errors list is a collection of all log entries that have *** in their string. ThinApp marks entries that could potentially be a problem by adding *** to the log entry output. See Locating Errors for more tips on interpreting this section.

```
----Potential Errors Detected ---

006425 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\Microsoft.Windows.Common-Controls.DLL' flags=2 -> 0 (failed ***)
006427 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\Microsoft.Windows.Common-Controls\Microsoft.Windows.Common-Control
s.DLL' flags=2
-> 0 (failed ***)
006428 0000089c nview.dll :1005b94b<-kernel32.dll:7c80ae4b *** LoadLibraryW -
>HMODULE=7c800000h () *** GetLastError() returns 2 [0]: The system cannot
find the file specified.
007062 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\en-US\Microsoft.Windows.Common-Controls.DLL' flags=2 -> 0 (failed
***)
010649 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\en-US\Microsoft.Windows.Common-Controls\Microsoft.Windows.Common-C
ontrols.DLL'
flags=2 -> 0 (failed ***)
019127 0000075c MSVCR80.dll :781348cc<-msvcrt.dll :77c10396 ***
GetEnvironmentVariableA -
>DWORD=0h (OUT LPSTR lpBuffer=*0h <bad ptr>) *** GetLastError() returns 203
[0]: The system
could not find the environment option that was entered.
019133 0000075c MSVCR80.dll :78133003<-nview.dll :1000058c *** GetProcAddress
-
>FARPROC=*0h () *** GetLastError() returns 127 [203]: The specified procedure
could not be found.
019435 0000075c MSVCR80.dll :78136e08<-dbghelp.dll :59a60360 *** Getfile type
```

```
->DWORD=0h ()
*** GetLastError() returns 6 [0]: The handle is invalid.
019500 0000075c MSVCR80.dll :78134481<-nview.dll :1000058c *** GetProcAddress
-
>FARPROC=*0h () *** GetLastError() returns 127 [0]: The specified procedure
could not be found.
019530 0000075c MSVCR80.dll :78131dcd<-dbghelp.dll :59a603a1 ***
GetModuleHandleA -
>HMODULE=0h () *** GetLastError() returns 126 [0]: The specified module could
not be found.
```

Troubleshooting Using Log Monitor

Log Monitor can be used to delve into application problems. This section uses an example in which `cmd.exe` is packaged with ThinApp and run with logging recorded.

To simulate an application behaving incorrectly, a simple invalid command is issued. In this case, we have requested `cmd.exe` to execute the command `foobar`, and `cmd.exe` prints the message `foobar is not recognized as an internal or external command`. By viewing the resulting trace file you can dig into what `cmd.exe` is doing in much greater detail and learn how it operates. All applications manifest problems in different ways, so there is more than one method to track issues.

The first place to check in a log file is the section near the end labeled "----Potential Errors Detected ---."

In this section, you can find all the API functions in which the `GetLastError` code was modified. The paths highlighted in bold indicate locations that `cmd.exe` was looking for foobar, and paths in red indicate locations in the virtual file system that were probed for these file system probes.

```
----Potential Errors Detected ---
*** Unable to determine if any services need to be auto-started, error 2
001550 *** FindFirstFileW 'C:\test\cmd_test\bin\foobar.*' ->
INVALID_HANDLE_VALUE *** failed
[system probe C:\test\cmd_test\bin\foobar.* -> ffffffffh][no virtual or
system matches]
*** FindFirstFileW ->HANDLE=ffffffffh .. *** GetLastError() returns 2 [203]:
The system cannot
find the file specified.
*** FindFirstFileW 'C:\test\cmd_test\bin\foobar' -> INVALID_HANDLE_VALUE ***
failed [FS
missing in view 0][fs entry not found %drive_C%\test\cmd_test\bin\foobar][fs
entry not found
%drive_C%\test\cmd_test\bin]
*** FindFirstFileW 'C:\WINDOWS\system32\foobar.*' -> INVALID_HANDLE_VALUE ***
failed [system
probe C:\WINDOWS\system32\foobar.* -> ffffffffh][no virtual or system
matches]
```

```
*** FindFirstFileW 'C:\WINDOWS\system32\foobar' -> INVALID_HANDLE_VALUE ***
failed [FS missing
in view 0][fs entry not found %SystemSystem%\foobar]
*** FindFirstFileW 'C:\WINDOWS\foobar.*' -> INVALID_HANDLE_VALUE *** failed
[system probe
C:\WINDOWS\foobar.* -> ffffffffh][no virtual or system matches]
*** FindFirstFileW 'C:\WINDOWS\foobar' -> INVALID_HANDLE_VALUE *** failed [FS
missing in view
0][fs entry not found %SystemRoot%\foobar]
*** FindFirstFileW 'C:\WINDOWS\System32\Wbem\foobar.*' ->
INVALID_HANDLE_VALUE *** failed
[system probe C:\WINDOWS\System32\Wbem\foobar.* -> ffffffffh][no virtual or
system matches]
*** FindFirstFileW 'C:\WINDOWS\System32\Wbem\foobar' -> INVALID_HANDLE_VALUE
*** failed [FS
missing in view 0][fs entry not found %SystemSystem%\Wbem\foobar]
*** FindFirstFileW 'c:\Program Files\subversion\bin\foobar.*' ->
INVALID_HANDLE_VALUE ***
failed [system probe c:\Program Files\subversion\bin\foobar.* ->
ffffffffh][no virtual or
system matches]
*** FindFirstFileW 'c:\Program Files\subversion\bin\foobar' ->
INVALID_HANDLE_VALUE *** failed
[FS missing in view 0][fs entry not found
%ProgramFilesDir%\subversion\bin\foobar][fs entry
not found %ProgramFilesDir%\subversion\bin]
*** FindFirstFileW 'c:\Program Files\Microsoft SQL
Server\90\Tools\binn\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe c:\Program Files\Microsoft SQL
Server\90\Tools\binn\foobar.* -> ffffffffh][no virtual or system matches]
*** FindFirstFileW 'c:\Program Files\Microsoft SQL
Server\90\Tools\binn\foobar' ->
INVALID_HANDLE_VALUE *** failed [FS missing in view 0][fs entry not found
%ProgramFilesDir%\Microsoft SQL Server\90\Tools\binn\foobar][fs entry not
found
%ProgramFilesDir%\Microsoft SQL Server\90\Tools\binn]
*** FindFirstFileW 'c:\bin\foobar.*' -> INVALID_HANDLE_VALUE *** failed
[system probe
c:\bin\foobar.* -> ffffffffh][no virtual or system matches]
*** FindFirstFileW 'c:\bin\foobar' -> INVALID_HANDLE_VALUE *** failed [FS
missing in view
0][fs entry not found %drive_c%\bin\foobar][fs entry not found %drive_c%\bin]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\Tools\WinNT\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe C:\Program Files\Microsoft
Visual
Studio\Common\Tools\WinNT\foobar.* -> ffffffffh][no virtual or system
matches]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\Tools\WinNT\foobar' ->
```

```
INVALID_HANDLE_VALUE *** failed [FS missing in view 0][fs entry not found
%ProgramFilesDir%\Microsoft Visual Studio\Common\Tools\WinNT\foobar][fs entry
not found
%ProgramFilesDir%\Microsoft Visual Studio\Common\Tools\WinNT]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\MSDev98\Bin\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe C:\Program Files\Microsoft
Visual
Studio\Common\MSDev98\Bin\foobar.* -> ffffffffh][no virtual or system
matches]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\MSDev98\Bin\foobar' ->
INVALID_HANDLE_VALUE *** failed [FS missing in view 0][fs entry not found
%ProgramFilesDir%\Microsoft Visual Studio\Common\MSDev98\Bin\foobar][fs entry
not found
%ProgramFilesDir%\Microsoft Visual Studio\Common\MSDev98\Bin]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\Tools\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe C:\Program Files\Microsoft
Visual
Studio\Common\Tools\foobar.* -> ffffffffh][no virtual or system matches]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\Common\Tools\foobar' ->
INVALID_HANDLE_VALUE *** failed [FS missing in view 0][fs entry not found
%ProgramFilesDir%\Microsoft Visual Studio\Common\Tools\foobar][fs entry not
found
%ProgramFilesDir%\Microsoft Visual Studio\Common\Tools]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual
Studio\VC98\bin\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe C:\Program Files\Microsoft
Visual
Studio\VC98\bin\foobar.* -> ffffffffh][no virtual or system matches]
*** FindFirstFileW 'C:\Program Files\Microsoft Visual Studio\VC98\bin\foobar'
->
INVALID_HANDLE_VALUE *** failed [FS missing in view 0][fs entry not found
%ProgramFilesDir%\Microsoft Visual Studio\VC98\bin\foobar][fs entry not found
%ProgramFilesDir%\Microsoft Visual Studio\VC98\bin]
```

As you can see, the potential errors does a good job of highlighting possible areas where the application is failing.

Deeper Examination

```
001550 *** FindFirstFileW 'C:\test\cmd_test\bin\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe
```

To determine why `cmd.exe` is probing the location `c:\test\cmd_test\bin`, search the log for this line of text using the *log entry number* and find out what is occurring before this call. In the **bold** excerpts below, two possible places are seen where `cmd.exe` obtained the path `c:\test\cmd_test`.

The first is obtained by calling `GetCurrentDirectoryW`, and the second is from calling `GetFullPathNameW` with "." as the path specifies. In both cases, this returns the path for the current working directory, making it clear exactly how `cmd.exe` is obtaining this path.

You can see in the log file how `cmd.exe` creates the `c:\test\cmd_test\bin>` prompt. This is accomplished by querying the environment variable "PROMPT" that returns "PG" and uses the API function `WriteConsoleW` to print the prompt to the screen after internally expanding "PG" to `c:\test\cmd_test\bin>.`

```
000824 0a88 cmd.exe :4ad0697a<-ADVAPI32.dll:77dd038f FormatMessageW
->DWORD=29h
(OUT LPWSTR lpBuffer=*4AD38BA0h->L"(C) Copyright 1985-2001 Microsoft
Corp.\0Dh\0Ah")
000825 0a88 cmd.exe :4ad069d1->ADVAPI32.dll:77dd038f FormatMessageW (IN DWORD
dwFlags=1800h, IN LPCVOID lpSource=*0h, IN DWORD dwMessageId=2334h, IN DWORD
dwLanguageId=0h, IN DWORD nSize=2000h, IN *Arguments=*13DD40h->...)
000826 0a88                        FormatMessageW
FORMAT_MESSAGE_FROM_HMODULE FORMAT_MESSAGE_FROM_SYSTEM
line_width=unlimited lpSource=0x0, dwMessageId=0x2334, dwLanguageId=0x0
                                   -> 0x29 ((C) Copyright 1985-2001 Microsoft
                  Corp.
 )
000827 0a88 cmd.exe  :4ad069d1<-ADVAPI32.dll:77dd038f FormatMessageW
->DWORD=29h
(OUT LPWSTR lpBuffer=*4AD38BA0h->L"(C) Copyright 1985-2001 Microsoft
Corp.\0Dh\0Ah")
000828 0a88 cmd.exe  :4ad08d01->kernel32.dll:7c835484 WriteConsoleW (IN
HANDLE
hConsoleOutput=7h, IN const *lpBuffer=*4AD38BA0h, IN DWORD
nNumberOfCharsToWrite=29h, IN
LPVOID lpReserved=*0h)
000829 0a88 cmd.exe  :4ad08d01<-kernel32.dll:7c835484 WriteConsoleW ->BOOL=1h
(OUT
LPDWORD lpNumberOfCharsWritten=*13DD24h->29h)
000830 0a88 cmd.exe  :4ad048f4->msctfime.ime:755c039b GetModuleHandleW (IN
LPCWSTR
lpModuleName=*4AD0498Ch->L"KERNEL32.DLL")
000831                          0a88 GetModuleHandleW 'KERNEL32.DLL' ->
7c800000
000832 0a88 cmd.exe  :4ad048f4<-msctfime.ime:755c039b GetModuleHandleW -
->HMODULE=7c800000h ()
000833 0a88 cmd.exe  :4ad04907->AcGenral.DLL:6f880364 GetProcAddress (IN
HMODULE hModule=7c800000h, IN LPCSTR lpProcName=*4AD04980h->"CopyFileExW")
000834 0a88                      GetProcAddress
mod=7c800000/C:\WINDOWS\system32\kernel32.dll () 'CopyFileExW' -> 7feb1fcf
000835 0a88 cmd.exe  :4ad04907<-AcGenral.DLL:6f880364 GetProcAddress -
->FARPROC=*7FEB1FCFh ()
000836 0a88 cmd.exe  :4ad04919->AcGenral.DLL:6f880364 GetProcAddress (IN
HMODULE
hModule=7c800000h, IN LPCSTR lpProcName=*4AD0496Ch->"IsDebuggerPresent")
```

```
000837 0a88                      GetProcAddress
mod=7c800000/C:\WINDOWS\system32\kernel32.dll () 'IsDebuggerPresent' ->
7fec0dfa
000838 0a88 cmd.exe :4ad04919<-AcGenral.DLL:6f880364 GetProcAddress -
->FARPROC=*7FEC0DFAh ()
000839 0a88 cmd.exe  :4ad0492b->AcGenral.DLL:6f880364 GetProcAddress (IN
HMODULE
hModule=7c800000h, IN LPCSTR
lpProcName=*4AD04954h->"SetConsoleInputExeNameW")
000840 0a88                       GetProcAddress
mod=7c800000/C:\WINDOWS\system32\kernel32.dll () 'SetConsoleInputExeNameW' ->
7fe90c21
000841 0a88 cmd.exe  :4ad0492b<-AcGenral.DLL:6f880364 GetProcAddress -
>FARPROC=*7FE90C21h ()
000842 0a88 cmd.exe  :4ad02c97->ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=3h)
000843 0a88                     Getfile type 3 -> 0x2
000844 0a88 cmd.exe  :4ad02c97<-ole32.dll :774e03f0 Getfile type ->DWORD=2h
()
000845 0a88 cmd.exe  :4ad02cc0->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF6h)
000846 0a88 cmd.exe  :4ad02cc0<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=3h ()
000847 0a88 cmd.exe  :4ad02ccd->kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=3h)
000848 0a88 cmd.exe  :4ad02ccd<-kernel32.dll:7c81af14 GetConsoleMode
->BOOL=1h (OUT
LPDWORD lpMode=*13DDCCh->A7h)
000849 0a88 cmd.exe  :4ad05b74->ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=7h)
000850 0a88                     Getfile type 7 -> 0x2
000851 0a88 cmd.exe  :4ad05b74<-ole32.dll :774e03f0 Getfile type ->DWORD=2h
()
000852 0a88 cmd.exe  :4ad05b9d->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF5h)
000853 0a88 cmd.exe  :4ad05b9d<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=7h ()
000854 0a88 cmd.exe  :4ad05baa->kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=7h)
000855 0a88 cmd.exe  :4ad05baa<-kernel32.dll:7c81af14 GetConsoleMode
->BOOL=1h (OUT
LPDWORD lpMode=*13DA80h->3h)
000856 0a88 cmd.exe  :4ad05dec->kernel32.dll:7c835484 WriteConsoleW (IN
HANDLE
hConsoleOutput=7h, IN const *lpBuffer=*4AD38BA0h, IN DWORD
nNumberOfCharsToWrite=2h, IN
```

```
LPVOID lpReserved=*0h)
000857 0a88 cmd.exe  :4ad05dec<–kernel32.dll:7c835484 WriteConsoleW –>BOOL=1h
(OUT
LPDWORD lpNumberOfCharsWritten=*13DAACh–>2h)
000858 0a88 cmd.exe  :4ad01ba8–>USERENV.dll :769c03b9 GetEnvironmentVariableW
(IN
LPCWSTR lpName=*4AD34624h–>L"PROMPT," IN DWORD nSize=2000h)
000859 0a88                    GetEnvironmentVariable PROMPT –> $P$G
000860 0a88 cmd.exe  :4ad01ba8<–USERENV.dll :769c03b9 GetEnvironmentVariableW
–
>DWORD=4h (OUT LPWSTR lpBuffer=*4AD2BA20h–>L"$P$G")
000861 0a88 cmd.exe  :4ad01580–>USERENV.dll :769c0396 GetCurrentDirectoryW
(IN DWORD
nBufferLength=104h)
000862 0a88                    GetCurrentDirectoryW –> 0x14
(C:\test\cmd_test\bin)
000863 0a88 cmd.exe  :4ad01580<–USERENV.dll :769c0396 GetCurrentDirectoryW
–>DWORD=14h
(OUT LPWSTR lpBuffer=*4AD34400h–>L"C:\test\cmd_test\bin")
000864 0a88 cmd.exe  :4ad05b74–>ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=7h)
000865 0a88                    Getfile type 7 –> 0x2
000866 0a88 cmd.exe  :4ad05b74<–ole32.dll :774e03f0 Getfile type –>DWORD=2h
()
000867 0a88 cmd.exe  :4ad05b9d–>kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF5h)
000868 0a88 cmd.exe  :4ad05b9d<–kernel32.dll:7c812f39 GetStdHandle
–>HANDLE=7h ()
000869 0a88 cmd.exe  :4ad05baa–>kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=7h)
000870 0a88 cmd.exe  :4ad05baa<–kernel32.dll:7c81af14 GetConsoleMode
–>BOOL=1h (OUT
LPDWORD lpMode=*13DA84h–>3h)
000871 0a88 cmd.exe  :4ad05dec–>kernel32.dll:7c835484 WriteConsoleW (IN
HANDLE
hConsoleOutput=7h, IN const *lpBuffer=*4AD2B1E0h, IN DWORD
nNumberOfCharsToWrite=15h, IN
LPVOID lpReserved=*0h)
000872 0a88 cmd.exe  :4ad05dec<–kernel32.dll:7c835484 WriteConsoleW –>BOOL=1h
(OUT
LPDWORD lpNumberOfCharsWritten=*13DAB0h–>15h)
000873 0a88 cmd.exe  :4ad0bf00–>ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=3h)
000874 0a88                    Getfile type 3 –> 0x2
000875 0a88 cmd.exe  :4ad0bf00<–ole32.dll :774e03f0 Getfile type –>DWORD=2h
()
000876 0a88 cmd.exe  :4ad02c97–>ole32.dll :774e03f0 Getfile type (IN HANDLE
```

```
hFile=3h)
000877 0a88                    Getfile type 3 -> 0x2
000878 0a88 cmd.exe  :4ad02c97<-ole32.dll :774e03f0 Getfile type ->DWORD=2h
()
000879 0a88 cmd.exe   :4ad02cc0->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF6h)
000880 0a88 cmd.exe   :4ad02cc0<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=3h ()
000881 0a88 cmd.exe   :4ad02ccd->kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=3h)
000882 0a88 cmd.exe   :4ad02ccd<-kernel32.dll:7c81af14 GetConsoleMode
->BOOL=1h (OUT
LPDWORD lpMode=*13DD50h->A7h)
000883 0a88 cmd.exe   :4ad02c97->ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=3h)
000884 0a88                    Getfile type 3 -> 0x2
000885 0a88 cmd.exe   :4ad02c97<-ole32.dll :774e03f0 Getfile type ->DWORD=2h
()
000886 0a88 cmd.exe   :4ad02cc0->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF6h)
000887 0a88 cmd.exe   :4ad02cc0<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=3h ()
000888 0a88 cmd.exe   :4ad02ccd->kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=3h)
000889 0a88 cmd.exe   :4ad02ccd<-kernel32.dll:7c81af14 GetConsoleMode
->BOOL=1h (OUT
LPDWORD lpMode=*13DD50h->A7h)
000890 0a88 cmd.exe   :4ad0b9d4->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF5h)
000891 0a88 cmd.exe   :4ad0b9d4<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=7h ()
000892 0a88 cmd.exe   :4ad0ba16->kernel32.dll:7c81bc2b
GetConsoleScreenBufferInfo (IN
HANDLE hConsoleOutput=7h)
000893 0a88 cmd.exe   :4ad0ba16<-kernel32.dll:7c81bc2b
GetConsoleScreenBufferInfo ->BOOL=1h
(OUT PCONSOLE_SCREEN_BUFFER_INFO lpConsoleScreenBufferInfo=*13DD08h->struct
{COORD dwSize=struct {SHORT X=50h, SHORT Y=12Ch}, COORD
dwCursorPosition=struct
{SHORT X=15h, SHORT Y=5h}, WORD wAttributes=7h, SMALL_RECT srWindow=struct
{SHORT
Left=0h, SHORT Top=0h, SHORT Right=4Fh, SHORT Bottom=18h}, COORD
dwMaximumWindowSize=struct {SHORT X=50h, SHORT Y=53h}})
000894 0a88 cmd.exe  :4ad0ba71->kernel32.dll:7c871a6c ReadConsoleW (IN HANDLE
hConsoleInput=3h, IN DWORD nNumberOfCharsToRead=2000h, IN LPVOID
lpReserved=*13DD20h)
001518 0a88 cmd.exe  :4ad0ba71<-kernel32.dll:7c871a6c ReadConsoleW ->BOOL=1h
(OUT
```

```
LPVOID lpBuffer=*4AD2FAE0h, OUT LPDWORD lpNumberOfCharsRead=*13DD70h->8h)
001519 0a88 cmd.exe  :4ad02c97->ole32.dll :774e03f0 Getfile type (IN HANDLE
hFile=3h)
001520 0a88                    Getfile type 3 -> 0x2
001521 0a88 cmd.exe  :4ad02c97<-ole32.dll :774e03f0 Getfile type ->DWORD=2h
()
001522 0a88 cmd.exe  :4ad02cc0->kernel32.dll:7c812f39 GetStdHandle (IN DWORD
nStdHandle=FFFFFFF6h)
001523 0a88 cmd.exe  :4ad02cc0<-kernel32.dll:7c812f39 GetStdHandle
->HANDLE=3h ()
001524 0a88 cmd.exe  :4ad02ccd->kernel32.dll:7c81af14 GetConsoleMode (IN
HANDLE
hConsoleHandle=3h)
001525 0a88 cmd.exe  :4ad02ccd<-kernel32.dll:7c81af14 GetConsoleMode
->BOOL=1h (OUT
LPDWORD lpMode=*13DD50h->A7h)
001526 0a88 cmd.exe  :4ad0bb9c->kernel32.dll:7c81b18f GetConsoleOutputCP ()
001527 0a88 cmd.exe  :4ad0bb9c<-kernel32.dll:7c81b18f GetConsoleOutputCP
->UINT=1B5h ()
001528 0a88 cmd.exe  :4ad0bbad->kernel32.dll:7c812e76 GetCPInfo (IN UINT
CodePage=1B5h)
001529 0a88 cmd.exe  :4ad0bbad<-kernel32.dll:7c812e76 GetCPInfo ->BOOL=1h
(OUT LPCPINFO
lpCPInfo=*4AD33BA0h->struct {UINT MaxCharSize=1h, char[2] DefaultChar=['?',
'\00h'], char[12]
LeadByte=['\00h', '\00h', '\00h', '\00h', '\00h', '\00h', '\00h', '\00h',
'\00h', '\00h', '\00h', '\00h']})
001530 0a88 cmd.exe  :4ad01680->kernel32.dll:7c81b258 SetThreadUILanguage (==
no prototype
available ==)
001531 0a88 cmd.exe  :4ad01680<-kernel32.dll:7c81b258 SetThreadUILanguage (==
no prototype
available ==)
001532 0a88 cmd.exe  :4ad01b0d->kernel32.dll:7c80ac0f SetErrorMode (IN UINT
uMode=0h)
001533 0a88 cmd.exe  :4ad01b0d<-kernel32.dll:7c80ac0f SetErrorMode ->UINT=0h
()
001534 0a88 cmd.exe  :4ad01b13->kernel32.dll:7c80ac0f SetErrorMode (IN UINT
uMode=1h)
001535 0a88 cmd.exe   :4ad01b13<-kernel32.dll:7c80ac0f SetErrorMode ->UINT=0h
()
001536 0a88 cmd.exe  :4ad01b24->IMM32.DLL :7639039b GetFullPathNameW (IN
LPCWSTR
lpFileName=*1638C0h->L"," IN DWORD nBufferLength=208h)
001537 0a88                    GetFullPathNameW . -> 20
(buf=C:\test\cmd_test\bin,
```

```
file_part=bin)
001538 0a88 cmd.exe  :4ad01b24<-IMM32.DLL :7639039b GetFullPathNameW
->DWORD=14h
(OUT LPWSTR lpBuffer=*163D60h->L"C:\test\cmd_test\bin," OUT
*lpFilePart=*13D8D4h-
>*163D82h->L"bin")
001539 0a88 cmd.exe   :4ad01b29->kernel32.dll:7c80ac0f SetErrorMode (IN UINT
uMode=0h)
001540 0a88 cmd.exe   :4ad01b29<-kernel32.dll:7c80ac0f SetErrorMode ->UINT=1h
()
001541 0a88 cmd.exe   :4ad01ba8->USERENV.dll :769c03b9
GetEnvironmentVariableW (IN
LPCWSTR lpName=*4AD34618h->L"PATH," IN DWORD nSize=2000h)
001542 0a88                    GetEnvironmentVariable PATH ->
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;c:\program
files\subversion\bin;c:\Program Files\Microsoft SQL
Server\90\Tools\binn\;c:\bin;C:\Program
Files\Microsoft Visual Studio\Common\Tools\WinNT;C:\Program Files\Microsoft
Visual
Studio\Common\MSDev98\Bin;C:\Program Files\Microsoft Visual
Studio\Common\Tools;C:\Program
Files\Microsoft Visual Studio\VC98\bin
001543 0a88 cmd.exe   :4ad01ba8<-USERENV.dll :769c03b9
GetEnvironmentVariableW -
>DWORD=173h (OUT LPWSTR lpBuffer=*4AD2BA20h-
->L"C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;c:\Program
Files\su.. ")
001544 0a88 cmd.exe   :4ad01ba8->USERENV.dll :769c03b9
GetEnvironmentVariableW (IN
LPCWSTR lpName=*4AD34608h->L"PATHEXT," IN DWORD nSize=2000h)
001545 0a88                    GetEnvironmentVariable PATHEXT ->
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
001546 0a88 cmd.exe   :4ad01ba8<-USERENV.dll :769c03b9
GetEnvironmentVariableW -
->DWORD=30h (OUT LPWSTR lpBuffer=*4AD2BA20h-
-> L."COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH")
001547 0a88 cmd.exe   :4ad02aaa->kernel32.dll:7c80b2d0 GetDriveTypeW (IN
LPCWSTR lpRootPathName=*13D8C4h->L"C:\")
001548 0a88 cmd.exe   :4ad02aaa<-kernel32.dll:7c80b2d0 GetDriveTypeW
->UINT=3h ()
001549 0a88 cmd.exe   :4ad01b5f->USERENV.dll :769c03fa FindFirstFileW (IN
LPCWSTR lpFileName=*1638C0h->L"C:\test\cmd_test\bin\foobar.*")
001550 0a88                    FindFirstFileW 'C:\test\cmd_test\bin\foobar.*'
->
INVALID_HANDLE_VALUE *** failed [system probe C:\test\cmd_test\bin\foobar.* ->
ffffffffh][no virtual or system matches]
```

Troubleshooting Specific Applications

The following sections describe troubleshooting tips that you can use to create packages for certain applications.

Microsoft Outlook

Outlook stores its account settings and other data in a number of registry keys and files. When you start Microsoft Outlook for the first time, it checks if these keys exist and, if not, prompts you to create a new account.

This works fine in the virtual environment when Microsoft Outlook is not installed on the physical system. However, when the user already has Microsoft Outlook installed physically, the captured version finds the registry keys in the system registry and use those settings, which is probably not what is intended. Full Isolation Mode is required for the registry keys and files where Microsoft Outlook stores its settings, so the captured version pretends the system settings do not exist. To set this up, some changes need to be made to the capture. Add the following to the `HKEY_CURRENT_USER.txt` file:

```
isolation_full HKEY_CURRENT_USER\Identities
isolation_full HKEY_CURRENT_USER\Software\Microsoft\Windows
NT\CurrentVersion\Windows Messaging Subsystem\Profiles
```

Next, a file named `##Attributes.ini` with the following contents:

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

This setting must be created in each of the following subdirectories:

```
%AppData%\Microsoft\AddIns
%AppData%\Microsoft\Office
%AppData%\Microsoft\Outlook
%Local AppData%\Microsoft\FORMS
%Local AppData%\Microsoft\Outlook
```

Create the subdirectories as needed.

Attachments

By default, Microsoft Outlook creates a directory where it stores attachments when you open an attachment for viewing. This directory is typically `C:\Documents and Settings\<user_name>\Local Settings\Temp\Temporary Internet Files\OLKxxxx` where the last `xxxx` is replaced by something randomly chosen.

This works fine when the viewing application runs in the same virtual sandbox as Microsoft Outlook. However, external application might not be able to find the file it is supposed to show. This is because Microsoft Outlook stores the file in the sandbox. To solve this, the isolation mode of the directory where the attachments are stored needs to be set to `Merged`.

You need to find out the name of the directory (the last part of the name was randomly chosen). To solve this, add a value to `HKEY_CURRENT_USER.txt` that sets the name of attachment directory:

```
isolation_full
HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Outlook\Security
Value=OutlookSecureTempFolder
REG_SZ~%Profile%\Local Settings\OutlookTempxxxx#2300
```

NOTE The "11.0" in the key name is for Outlook 2003, use the appropriate version number for your version of Outlook.

For slightly increased security, replace the last four "xxxx" characters with something random. Next, create a directory `%Profile%\Local Settings\OutlookTempxxxx` in your project and create an `##Attributes.ini` file there containing:

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

The directory `%Profile%\Local Settings\OutlookTempxxxx` is just an example, you can use whatever you want, as long as you make sure that the directory is named in the OutlookSecureTempFolder registry key and is set to the correct isolation mode.

Explorer.exe

In typical circumstances, Windows operating systems allow you to run one instance of `explorer.exe`. This makes it difficult to add an entry point to Windows Explorer and launch it inside the virtual environment.

You can use the following methods to launch a Windows Explorer windows inside the virtual environment:

- Add an entry point to iExplorer and launch it with the `–E` parameter.

  For example, use these entries in 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 reg key:

```
isolation_full
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer
Value=DesktopProcess
REG_DWORD=#01#00#00#00
```

Use these entries in the `Package.ini` file:

```
[explorer.exe]
Shortcut=xxxxxx.exe
Source=%SystemROOT%\explorer.exe
```

Some advantages with this method include the ability to browse virtual file system with a familiar interface, enabling accurate file type associations without system changes, especially when using portable applications, and accessing shell-integrated components without system changes.

Java Runtime Environment (JRE)

A conflict might occur if one version of Java is installed natively and another one is included in a captured executable. This is because newer versions of Java also install a plug-in DLL that gets loaded by `IE`. This plug-in DLL overwrites virtual registry keys when it is loaded and can conflict with a virtualized copy of older Java runtimes. You can prevent IE from loading plug-ins by adding the following line to beginning of `HKEY_LOCAL_MACHINE.txt`.

```
isolation_full
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser
Helper Objects
```

Package.ini Parameters

A

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

You can set numerous `Package.ini` file settings during the setup capture process. For more information, see Chapter 2, "Capturing Applications," on page 19.

The BuildOptions section applies to all applications. Individual applications inherit any options that you set in this section, unless the application section specifically overrides these settings.

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

This information discusses the following topics:

Isolation and Virtualization Granularity

The following sections describe the isolation and virtualization granularity parameters.

ChildProcessEnvironmentDefault

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

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

To create certain processes outside of the virtual environment, specify them in the `ChildProcessEnvironmentException` attribute.

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

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

Specify that the default behavior is to create child processes outside of the virtual environment:

```
[BuildOptions]
ChildProcessEnvironmentDefault=External
```

ChildProcessEnvironmentExceptions

`ChildProcessEnvironmentExceptions`—Allows for certain processes outside of the virtual environment.

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

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

DirectoryIsolationMode

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

`DirectoryIsolationMode` controls the default isolation mode for the package. This setting applies to any directories that do not have an explicitly specified setting. The default setting depends on the selection made in the setup capture process.

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

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

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

Following are examples of this parameter:

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

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

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

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

ExternalCOMObjects

`ExternalCOMObjects`—Controls whether ThinApp or Windows creates a specific COM object CLSID.

By default, ThinApp creates all COM objects in the virtual environment. COM supports out-of-process (`.exe`) servers as well as service-based COM objects. If an application can create such COM objects on the host computer and cause them to modify the host computer, the integrity of the host computer cannot be assured. However, if ThinApp runs out-of-process and services-based COM objects in the virtual environment, all changes that the COM objects make are stored in the sandbox.

For example, the following code instructs ThinApp to run two COM objects outside of the virtual environment if the application creates them.

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

ExternalDLLs

`ExternalDLLs`—Force windows to load some DLLs.

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

The `ExternalDLLs` option has limited usefulness. If the DLL depends on other DLLs that are located in the virtual file system, Windows cannot load it.

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

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

IsolatedMemoryObjects

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

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

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

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

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

IsolatedSynchronizationObjects

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

Windows has several named Synchronization objects:

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

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

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

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

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

RegistryIsolationMode

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

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

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

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

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

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

SandboxCOMObjects

`SandboxCOMObjects`—Indicates whether COM objects registered at runtime by the application are accessible to native applications.

```
SandboxCOMObjects=1
```

VirtualizeExternalOutOfProcessCOM

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

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

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

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

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

The following example instructs ThinApp to run all external out-of-process COM objects in the virtual environment (default).

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

General Options

The following sections describe the general options parameters.

AddPageExecutePermission

`AddPageExecutePermission`—Fixes applications that do not work in DEP environments.

The Data Execution Prevention (DEP) feature of Windows XP SP2, Windows Server 2003, and later protects against some security exploits that occur with buffer overflows. This feature causes a number of compatibility issues. The feature is turned off by default on XP SP2 and you can use a machine-specific opt-in or opt-out list of the applications to apply DEP protection to. Opt-in and opt-out policies can be difficult to manage when a large number of machines and applications are involved. This option instructs ThinApp to add Execution Permission to pages that an application allocates. The application can then run on machines that have DEP protection enabled, without having to modify the opt-out list.

For example:

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

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

AllowUnsupportedExternalChildProcesses

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

For example:

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

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

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

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

```
AllowUnsupportedExternalChildProcesses=0
```

AnsiCodePage

`AnsiCodePage` – Specifies the country locale where you captured the application in a numerical value.

For example:

```
[BuildOptions]
AnsiCodePage=1252
```

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

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

```
AllowUnsupportedExternalChildProcesses=0
```

AutoShutdownServices

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

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

Following are examples of this parameter:

Keep virtual services running when the application exits.

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

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

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

AutoStartServices

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

By default, ThinApp starts virtual services that are installed with the startup type of Automatic. The virtual services start when the user runs the first parent process. You can use this option to disable auto-starting of virtual services.

Following are examples of `AutoStartServices`:

Do not start virtual services.

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

Start virtual services when first process starts (default).

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

BlockSize

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

Using a larger block size can achieve higher compression. Larger block sizes can negatively effect performance.

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

You can specify `BlockSize` in the following places:

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

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

For example, the default block size is 64K

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

Other block size options follow:

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

CachePath

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

You can use the THINAPP_CACHE_DIR environment variable to override the setting at runtime.

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

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

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

With the package located in C:\VirtApps, the cache is set to C:\VirtApps\Cache:

```
CachePath=Cache
```

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

```
CachePath=Sandbox
```

CapturedUsingVersion

CapturedUsingVersion— Indicates the version number of the Setup Capture wizard used during the capture process. Users do not need to adjust this setting.

```
[BuildOptions]
CapturedUsingVersion=4.0.0–2200
```

CompressionType

CompressionType—Controls the type of compression that is used.

ThinApp supports two compression models: None and Fast.

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

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

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

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

You can specify CompressionType in the following places:

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

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

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

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

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

DisableTracing

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

You can use the ThinApp Log Monitor utility to produce a `.trace` file for applications that ThinApp runs. You can use this option to disable the ability to produce trace files for specific applications.

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

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

The following example allows Log Monitor to create a `.trace` file (default):

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

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

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

ExcludePattern

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

You can use this option to exclude specific files or directories from a project determined dynamically at build time. The list of patterns is specified with a comma separator. The wildcard (*) matches 0 or more following characters and the question mark (?) matches exactly one character. This syntax is similar to the DOS `dir` command. Unlike the DOS `dir` command syntax, you can apply wildcard characters to directory names and filenames. You can specify `ExcludePattern` in the following places:

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

For example:

```
[FileList]
```

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

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

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

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

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

FileTypes

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

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

Normally, Setup Capture places the FileTypes list in the `Package.ini` file that it generates. You can manually remove extensions that you do not want to associate with the virtual package. For example, if you virtualize Microsoft Office 2007 and have Office 2003 installed natively, you can remove the `.doc` extension from the FileTypes list and leave `.docx`. In this way, Word 2003 opens `.doc` files. Word 2007 opens `.docx` files.

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

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

LocaleIdentifier

`LocaleIdentifier`—Displays a numeric ID for the locale.

```
[BuildOptions]
LocaleIdentifier=1033
```

LocaleName

`LocaleName`—Displays the name of the locale when you capture an application on Microsoft Vista.

```
[BuildOptions]
LocaleName=ja-JP
```

LogPath

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

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

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

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

OutDir

`OutDir` – Specifies the directory where the `build.bat` output is stored.

Review this sample entry:

```
[BuildOptions]
OutDir=bin
```

NetRelaunch

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

By default, ThinApp detects if an application is running from a network drive or a removable disk, and uses a local hard disk to restart the application. This process resolves a problem in which Symantec AntiVirus tries to perform a complete scan of an executable file under some conditions. This scan can affect on start times for large executable files located on network shares.

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

Because a large number of desktops have Symantec AntiVirus installed, ThinApp allows applications to start from a network share without incurring lengthy scan times. ThinApp creates a stub executable file in the user's sandbox that is restarted. Because Symantec AntiVirus can scan the stub quickly, it loads the remainder of the application data from the original source location. You can disable this default behavior by adding the `Package.ini` option.

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

For example, disable restart of the application locally on a hard disk.

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

If started from a network drive, restart using a local stub (default).

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

Protocols

`Protocols`—Specifies the protocols, such as HTTP, that are visible to applications on the system.

For example, if you have a Microsoft Outlook package, you can enter the mailto protocol.

Review this sample entry:

```
[BuildOptions]
Protocols=feed;feeds;mailto; Outlook.URL.mailto;stssync;webcal;webcals
```

RuntimeEULA

`RuntimeEULA` – Controls the ThinApp EULA display. By default the EULA does not appear. Depending on licensing terms forThinApp, you might be required to display an end-user license agreement.

Following are examples of the `RuntimeEULA` parameter:

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

Shortcuts

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

For example, use `thinreg.exe` to create a shortcut to the virtual Word 2003 application in the Microsoft Office folder that appears in the **Start** menu:

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

UACRequestedPrivilegesLevel

`UACRequestedPrivilegesLevel`—Specifies privileges for programs requiring this information. This parameter affects users working on Vista or higher operating system versions.

You can use the following values:

- invoker

  This value uses the profile invoked in Vista.

- requireAdministrator
- highestavailable

  This value uses the highest available privilege that can avoid the UAC prompt.

```
[BuildOptions]
UACRequestedPrivilegesLevel=requireAdministrator
```

UACRequestedPrivilegesUiAccess

`UACRequestedPrivilegesUIAccess`—If you work with Vista or a higher operating system version, some elements of the user interface are protected.

You can assign the `UACRequestedPrivilegesUIAccess` parameter a `true` or `false` value.The default value of `false` ensures the virtual application cannot access protected elements. In typical circumstances, applications do not require such access.

Review this sample entry:

```
[BuildOptions]
UACRequestedPrivilegesUiAccess=false
```

UpgradePath

`UpgradePath`—Location to probe for application updates.

By default, ThinApp looks for application upgrades in the same directory as the main executable file. You can use this option to specify an alternative location to look for application upgrades.

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

```
[BuildOptions]
UpgradePath=C:\Program Files\MyAppUpgrades
```

When Application Sync downloads an update from a server, it stores the update with a temporary name in the `UpgradePath` directory. After the download is complete, the temporary file is renamed with a .1 extension (or .2 if .1 already exists) the next time the application starts. No other applications can be using the same sandbox. ThinApp

attempts to change the .1 name (still in the UpgradePath directory) to the original filename (which might be in another directory). If ThinApp cannot make this change, the file keeps the .1 extension in the `UpgradePath` directory. That file is still accessed when you run the original application.

VirtualComputerName

`VirtualComputerName`–A string that `GetComputerName` and `GetComputerNameEx` APIs return in a captured application.

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

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

```
VirtualComputerName=LOCALHOST
```

Because you renamed the machine before you performed the setup capture process, the application uses `LOCALHOST` as its computer name. If you enter a `GetComputerName` or `GetComputerNameEx` command, the machine returns `LOCALHOST`.

If your system requires that the `GetComputerName` and `GetComputerNameEx` APIs behave normally, do not rename the machine `LOCALHOST`. The `VirtualComputerName` parameter is commented out in the `Package.ini` file, as the following example shows. The machine name replaces `OriginalMachineName`.

```
;VirtualComputerName=OriginalMachineName
```

VirtualDrives

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

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

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

Projects that Setup Capture creates list virtual drive information for drives that are present at the time of capture.

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

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

Semicolons separate drive letters and commas separate parameters for individual drive letters.

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

`Serial=`(8-digit hex number)

`Type=`

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

Basic usage is to specify a single virtual drive letter. By default, a drive is assigned a serial number and the `FIXED` type.

```
[BuildOptions]
VirtualDrives=Drive=M
```

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

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

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

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

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

NOTE The semicolon (`;`) separates information assigned to different drive letters.

To change the isolation mode for a virtual drive

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

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

Access Control

The following section describes the access control parameters.

AccessDeniedMsg

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

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

For example:

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

This setting controls the string that appears to the user if they are not authorized to run the application.

PermittedGroups

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

For example, specify a list of Active Directory user group names separated by a semicolon. `[BuildOptions]` can set defaults for applications in a project that individual `[App.exe]` sections can overwrite.

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

Overwrite global PermittedGroups

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

Inherit PermittedGroups from [BuildOptions]

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

During the application build process, ThinApp transforms the specified Group Names into SID values so the Group Names specified must be valid at the time of build. ThinApp can resolve group ownership at runtime using cached credentials. Laptop users can continue to be authenticated even when they are offline.

If you use a domain-based group name, you must be connected to that domain when you build the application package.

If the user does not have access to run the package, you can customize the `AccessDeniedMsg` option to instruct the user.

Application-Specific Parameters

The following sections describe the application-specific parameters.

Disabled

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

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

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

CommandLine

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

For shortcut executable files, you can specify the command line that is passed to the virtual application. Include the application name as the first parameter.

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

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

Use folder macros for your path name conventions. See "Using Folder Macros" on page 158 for more information.

Icon

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

By default, each generated application uses the main group icon from its source executable file and the individual icon resource that the group icon points to. To use an alternative icon, specify an `.ico` file or executable file.

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

NOTE You cannot use NULL if you use the file types directive. The executable image allocates one icon for each file type.

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

Specify the application icon by using an executable file different than the `Source` executable:

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

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

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

Use a `.ico` file to specify the application icon:

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

NoRelocation

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

Windows executable files might contain base relocation information that enables Windows to load the executable image at an alternative starting memory address. If the base address is greater than 0x40000, the executable image is always loaded at its specified base address. Relocation information is unnecessary and can be safely stripped from the resulting image. Relocation information is typically small on disk. Removing this information does not affect the size of the generated executable files.

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

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

ReadOnlyData

`ReadOnlyData`— Specifies the name of the read-only virtual registry file created during the build. If this parameter appears in an application-specific section of the `Package.ini` file, the specified file is the primary data container.

Review this sample entry:

```
ReadOnlyData=bin\Package.ro.tvr
```

ReserveExtraAddressSpace

`ReserveExtraAddressSpace`—Indicates how much extra address space to reserve for the captured executable file.

In typical circumstances, `tlink.exe` sets the `SizeOfImage` field in the generated executable based on the `SizeOfImage` field of the source executable. The Windows loader uses the `SizeOfImage` field to determine how much virtual address space to reserve for the executable file. When you build a package based on a source executable file that is not included in the package, you can reserve virtual address space by specifying the `ReserveExtraAddressSpace` option. The value is the number of bytes to reserve. Optionally, you can follow the number by K to indicate kilobytes or M to indicate megabytes. The default value is `0`, which specifies address space to reserve.

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

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

The following tells the Windows loader not to reserve extra address space (the default).

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

RetainAllIcons

`RetainAllIcons`—Retains all of the `Source` executable file's original icons in the generated captured executable file.

By default, `tlink.exe` constructs a new executable using a source executable. To reduce disk space, the new executable image contains only icons viewable from the system shell. The package contains all other icons. The icons remain accessible to the application while it is running. The icons that the system can access system cannot be compressed, so their disk size is larger. In some cases you might want to have all of the application's original icons visible to the system shell.

Following are examples of the `RetainAllIcons` parameter:

Instruct `tlink.exe` to retain all of the application's original icons.

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

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

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

Shortcut

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

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

The `HostApp.exe` file must be listed in the package. It also needs to be present in the same directory for `MyShortcutApp.exe` to run.

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

Source

`Source`—Points to the executable file that ThinApp initially loads.

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

The following are examples of the `Source` parameter:

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

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

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

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

StripVersionInfo

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

Version information for an executable file is in the Windows Properties dialog box. Typically, this includes copyright, trademark, and version number information. By default, ThinApp copies all version information from the source executable file (specified using `Source`) (See "Source" on page 114). You can use this option to strip version information from the resulting application.

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

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

WorkingDirectory

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

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

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

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

Version.XXXX

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

Version resources are normally copied from the original executable file. To override the version resource strings and add new ones, use a "Version.<string_name>=string_value" setting.

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

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

Application Link

In the `Package.ini` file, you can use the `OptionalAppLinks` and `RequiredAppLinks` features to dynamically combine ThinApp packages at runtime on end-user computers. See "OptionalAppLinks" on page 116 and "RequiredAppLinks" on page 116. This process enables you to package, deploy, and update component pieces separately and retain the benefits of application virtualization.

Application Link Pathname Formats

Application Link supports the following pathname formats:

- Pathnames can be relative to the base .exe file. For example, `RequiredAppLinks=..\SomeDirectory` results in `c:\MyDir\SomeDirectory` if the base EXE is deployed to `c:\MyDir\SubDir\ Dependency.exe`.
- Pathnames can be an absolute pathname. For example, `RequiredAppLinks=c:\SomeDirectory`
- Pathnames can be located on a network share or use a UNC path. For example, `RequiredAppLinks=\\share\somedir\Dependency.exe`

- Pathnames can contain environment variable and dynamically expand to any of the preceding pathnames. For example `RequiredAppLinks=%MYAPP_ADDONS%\Dependency.exe`
- To specify multiple links or dependencies, use the semicolon (;) to separate individual filenames. For example, `RequiredAppLinks=Dependency1.exe; Dependency2.exe;`

RequiredAppLinks

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

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

Importing packages do the following:

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

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

OptionalAppLinks

`OptionalAppLinks` – Operates like `RequireAppLinks` except that the error is ignored and the main application starts running if an import fails to load. See "RequiredAppLinks" on page 116.

Collisions and Order of Import

ThinApp uses a "last import wins" policy to determine what happens when two packages with the same files or registry keys are imported. See "Reviewing File and Registry Collisions in Linked Packages" on page 58.

Security and Authorization

You must be a member of all PermittedGroups sections for all imported packages. Otherwise you is receive an Access Denied message and the application fails to load. The following are limitations of this feature:

- ThinApp supports importing up to 250 packages at a time. Each package can be any arbitrary size.
- Packages that AppSync updates do not have updates visible to the parent executable file.
- Sandbox changes from packages being imported are not be visible to the parent executable file.

The following are examples:

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

```
RequiredAppLinks=Plugin.exe
```

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

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

This imports all executable files located in the plug-ins directory:

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

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

The following imports all EXEs located at the absolute path `n:\plugins`:

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

The following expands the environment variable, `PLUGINS`, and imports all executable files found at this location:

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

The following loads two specified plug-ins and a list of executable files found under plug-ins:

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

Application Sync

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

You can configure Application Sync by editing the `Package.ini` file.

The following entries are the default settings for Application Sync:

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

The following sections describe the Application Sync parameters.

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

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

AppSyncURL

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

For example:

```
AppSyncURL=https://example.com/<path>/<package_name>.exe
```

Another option is to use the `File` protocol. For example:

```
File:///C:/<path>/<package_name>.exe
File:///<server>/<share>/<path>/<package_name>.exe
```

AppSyncUpdateFrequency

`AppSyncUpdateFrequency`–By default, a package connects to the Web server once a day to check if an updated version is available. Modify this frequency to set the frequency.

For example:

```
AppSyncUpdateFrequency=1d
```

You can set the Application Sync update frequency to a number followed by `m` (month), `d` (day), or `y` (year). A value of 0 makes the captured application check for updates every time you start it.

AppSyncExpirePeriod

`AppSyncExpirePeriod`–Sets the update frequency in minutes (m), hours (h), or days (d). If the Web server cannot be reached, the package continues to work until the `AppSyncExpirePeriod` is reached. The default setting is 30 days. Modify the setting to change the update frequency.

For example:

```
AppSyncExpirePeriod=30d
```

To prevent the package from expiring, set the expiration value to `never` as follows:

```
AppSyncExpirePeriod=never
```

AppSyncWarningPeriod

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

For example:

```
AppSyncWarningPeriod=5d
```

AppSyncWarningFrequency

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

With the default of one day, the warning message appears only once a day. To configure the warning to appear each time the application starts, use 0.

After the warning period starts, the Web server is checked every time an application starts, overriding the setting.

If a package is not expired, this parameter checks for new versions and downloads occur in the background. The user can continue to use the old version. If the user quits an application before the download is complete, the download resumes when a virtual application starts again. After the download completes, the new version is activated the next time the application starts.

When the package expires, the version check and download occur in the foreground. A progress bar appears during the download phase.

For example:

```
AppSyncWarningFrequency=1d
```

AppSyncWarningMessage

`AppSyncWarningMessage`–If the connection to the Web server fails, a message box appears. The following example shows the default message.

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

AppSyncExpireMessage

`AppSyncExpireMessage`–After the expiration limit is reached and a virtual application starts, the application tries to connect to the Web server and check for a new version. If the connection fails, a message boxappears and the application quits. The following example shows a default message:

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

AppSyncUpdatedMessage

`AppSyncUpdatedMessage`–When an updated package first starts, an information message can appear.

For example:

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

AppSyncClearSandboxOnUpdate

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

For example:

```
AppSyncClearSandboxOnUpdate=0
```

MSI Generation

The following parameters generate MSI files. For more information about MSI generation, see "Building an MSI Database" on page 37.

MSIArpProductIcon

`MSIArpProductIcon` – Specifies the icons to place in the control panel for Add/Remove programs.

The following example assumes you are using Microsoft Office 2007:

```
MSIArpProductIcon=%Program Files Common%\Microsoft Shared\OFFICE12\
                Office Setup Controller\OSETUP.DLL,1
```

The general format is `MSIArpProductIcon=<filename>[,<icon_index>]`

The `<icon_index>` entry is optional.

MSIDefaultInstallAllUsers

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

This option has an effect only when the `MSIFilename` option requests generation of a Windows Installer database (see "MSIFilename" on page 123).

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

For example, this directs ThinApp to generate an MSI that is installed for each user by default.

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

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

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

For example, this directs ThinApp to generate an MSI that is installed for each machine by default.

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

For example, `msiexec.exe mymsi.msi ALLUSERS=""`

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

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

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

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

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

MSIFilename

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

If set, this option causes the build to produce a Windows Installer with the specified filename in the output directory.

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

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

MSIInstallDirectory

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

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

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

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

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

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

MSIManufacturer

`MSIManufacturer` – Specifies the manufacturer to put in the MSI database. The default setting is the name of the company to which your copy of Windows is registered.

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

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

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

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

MSIProductCode

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

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

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

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

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

MSIProductVersion

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

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

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

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

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

MSIRequireElevatedPrivileges

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

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

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

When you set the option to 0, no UAC prompt is given, but you cannot install machine-wide.

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

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

MSIUpgradeCode

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

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

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

The following creates an MSI file with a specified upgrade code.

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

MSIUseCabs

`MSIUseCabs` – If you set the value to 1, the package files are stored in a `.cab` file. The `.cab` file is in the MSI file.

If you set the value to 0, Thinapp does not use `.cab` files. You might do this when a `.cab` file slows down the installation process for applications. You can distribute the MSI file and individual `.exe` files (executables) in `/bin` to install the application.

```
[BuildOptions]
MSIUseCabs=1
```

Sandbox Control

The following sections describe the sandbox control parameters.

SandboxName

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

For example:

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

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

SandboxPath

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

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

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

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

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

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

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

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

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

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

InventoryName

`InventoryName`—Optional string that inventory control systems use for package identification.

For example:

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

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

Use `InventoryName` as a version-independent string to track a ThinApp package with inventory-scanning software.

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

ThinApp does not use `InventoryName` during build or runtime.

SandboxNetworkDrives

`SandboxNetworkDrives`—Determines if sandboxing is appliede to network-mapped drives.

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

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

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

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

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

SandboxRemovableDisk

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

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

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

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

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

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

RemoveSandboxOnExit

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

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

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

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

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

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

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

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

Using the Sandbox

B

The sandbox is the directory where all changes that the captured application makes are stored. The next time you launch the application, those changes are incorporated from the sandbox. By deleting the sandbox directory, you can instantly revert the application back to its captured state.

This information discusses the following topics:

Reviewing the 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.

The description of the search order uses these examples:

- Sandbox name is Mozilla Firefox 3.0
- Application resides in `C:\Program Files\Firefox`
- User computer name is JOHNDOE-COMPUTER
- Sandbox path is `Z:\sandboxes`
- `%AppData%` is `C:\Documents and Settings\JohnDoe\Application Data`

VMware recommends reviewing the following search order before changing the placement of the sandbox:

- %<sandbox_name>_SANDBOX_DIR.<computer_name>%

  For example, Mozilla Firefox 3.0_SANDBOX_DIR.JOHNDOE-COMPUTER.

- %<sandbox name>_SANDBOX_DIR%

  For example, Mozilla Firefox 3.0_SANDBOX_DIR.

- %THINSTALL_SANDBOX_DIR%\<sandbox_name>.<computer_name>

  For example, %THINSTALL_SANDBOX_DIR%.Mozilla Firefox 3.0.JOHNDOE-COMPUTER.

- %THINSTALL_SANDBOX_DIR%\<sandbox_name>

  For example, %THINSTALL_SANDBOX_DIR%.Mozilla Firefox 3.0.

- <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>.<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 cannot locate an existing sandbox according to the ordered locations, ThinApp creates a new sandbox in these locations using this logic:

- If the SANDBOXNAME_SANDBOX_DIR environment variable is set, create a sandbox at this location.
- If the THINSTALL_SANDBOX_DIR environment variable is set, create a sandbox at this location.
- If the SANDBOXPATH Package.ini parameter is set, create a sandbox at this location.
- Create a sandbox in the %AppData%\Thinstall folder of the user.

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

If you use Active Directory, %AppData% is often mapped to a shared network drive to allow for easy backups. In this situation, users can log in to any machine and retain their application settings.

Place the Sandbox on the Network

ThinApp provides the SandboxPath parameter to store the sandbox on a mapped drive.

To place the sandbox on the network

1 Open the `Package.ini` file.

2 Set the `SandboxPath` parameter under the `SandboxName` parameter to the network location:

```
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`.

Place the Sandbox on a USB Drive

ThinApp provides the `SandboxPath` parameter to set a USB location for the sandbox. The captured application loads and saves changes to the same directory on the USB drive where the executable resides.

To place the sandbox on a USB drive

1 Open the `Package.ini` file.

2 Set the `SandboxPath` parameter under the `SandboxName` parameter to this value:

```
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 where Firefox runs from.

Make a Portable Application

To make a captured application portable for a USB device, iPod, or other similar device, the sandbox must reside in a subdirectory relative to the executable.

To make a portable application

1 Create a `Thinstall` directory in the same directory as your captured application.

2 Move the `Thinstall` directory from `%AppData%` to the application directory.

The next time the application starts, the application operates on this local sandbox.

3 Copy the application and sandbox to a portable device and start the application there.

Reviewing the 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 these 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 from the last successful run if ThinApp detects corruption. On startup, ThinApp restores the `.tvr` file.

Besides 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 need to consult the real file system for the real system location and again for the sandbox location. Instead, ThinApp can check for the file's existence 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 performing 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. Running this command requires 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`.

ThinApp Directory Files

C

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

Review some of the key files and utilities in the `VMware ThinApp` directory:

- **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.
- **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 pre-installation environment and post-installation environment during the application capture process.

  ThinApp starts this utility during the Setup Capture wizard.
- **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 ThinApp does not capture certain entries when creating an application package.

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

  You can customize this template to ensure 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 start 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.

ThinApp Snapshot Files

D

This information discusses advanced usage of the `snapshot.exe` and `snapshot.ini` files and includes these topics:

Using the snapshot.exe utility

You do not need to use `snapshot.exe` directly because the Setup Capture wizard invokes it. Only advanced users and system integrators who are building ThinApp functionality into other platforms might make direct use of this utility.

Use this utility to create a snapshot of a computer file system and registry or create a ThinApp project from two previously captured snapshots.

Creating a Snapshot

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

- The `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.

Sample Commands

Table D-1 describes sample commands for the `snapshot.exe` utility. The parameters are not case sensitive.

Table D-1. snapshot.exe Sample Commands

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

Table D-1. snapshot.exe Sample Commands (Continued)

Command	Description

`snapshot CaptureFile.snapshot`
`[BaseDirectory1][BaseDirectory2]`
`[BaseRegistry1][BaseRegistry2]`
`[–Config ConfigFile.ini]`
`snapshot Original.snapshot –Diff`
`NewEnvironment.snapshot`
`OutputDirectory`
`[–Config ConfigFile.ini][–Quiet]`
`snapshot Original.snapshot`
`–DiffPrint NewEnvironment.snapshot`
`[–Config ConfigFile.ini]`
	`–DiffPrint` is similar to `–Diff` but does not have an output directory. `–DiffPrint` prints all detected changes to the console.

`snapshot c:\start.snapshot`
`echo Please install application and press ENTER when installation is complete`
`pause`
`snapshot C:\end.snapshot`
`snapshot C:\start.snapshot`
`–SuggestProject c:\end.snapshot`
`c:\SuggestedProject.ini`
	Sample `.bat` file that generates a ThinApp project using a starting and ending snapshot.

`snapshot C:\SuggestedProject.ini`
`–GenerateProject c:\ProjectLocation`
`del C:\start.snapshot`
`del C:\end.snapshot`
`del C:\SuggestedProject.ini`
	A GUI application can look at the suggested project and ask the user which executables to make accessible from the `Package.ini` file.

Take a snapshot

This sample procedure describes how to use the `snapshot.exe` utility to manually perform the snapshot steps that automatically occur during the Setup Capture wizard.

To take a snapshot

1 Take a pre-installation snapshot of the computer:

```
snapshot.exe SnapshotFileName.snapshot http://–Config
               ConfigFile.iniBaseDir1BaseDir2BaseReg1
```

2 Take a post installation snapshot:

```
snapshot.exe Snap1.snapshot Snap2.snapshot OutDir http://–Config
               ConfigFile.ini
```

3 Create settings to allow for user or external GUI program modifications:

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

4 Create `Package.ini` settings file from the `SuggestedProject.ini` file:

```
snapshot.exe Template.ini –GenerateProject OutDir http://–Config
             ConfigFile.ini
```

5 (Optional) Print the snapshot file:

```
snapshot.exe SnapshotFileName.snapshot –Print
```

Using the snapshot.ini File

The `snapshot.ini` file configuration file specifies what directories and subkeys 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 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 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.

Using Scripts

E

Scripting allows you to execute custom code before starting an application packaged with ThinApp or after an application exits. Callback functions allow you to specify when blocks of code execute. API functions allow you to execute ThinApp-specific functions and interact with the ThinApp runtime.

To add scripts to your application, you can create an ANSI text file with the `.vbs` file extension in the root project directory for an application (the same directory in which `Package.ini` is located). During the build process, ThinApp adds all of the script files to your executable file and then at runtime, it executes each of these script files.

Many applications create child processes, and scripts execute for child processes as well. To execute code only in the main parent process, use callback functions.

ThinApp uses VB Script to execute script files, so any valid VB Script loads under a ThinApp executable. Refer to Microsoft's VB Script User Guide for more information. VB script can be used to access COM controls registered on the host system, or registered within the virtual package.

This information includes the following topics:

Callback Functions

Callback functions with specific names execute only under certain conditions. For example, callback functions allow script code to execute only when an application starts or quits.

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 is launched using the same sandbox while the first copy is still running. If a first application spawns sub-process and then quits, the sandbox remains locked by the second sub-process so this callback does not execute again until all sub processes have quit and the application is run again.
- `OnFirstParentStart`—Called prior to executing a ThinApp executable file regardless of whether the sandbox is simutaneously owned by another captured executable.
- `OnFirstParentExit`—Called when the first parent process exits. If a parent process executes a child process and quits, this callback is called even if the child process continues to execute.
- `OnLastProcessExit`—Called when the last process owning the sandbox exits. If a parent process executes a child process and quits, this callback is called when the last child process exits.

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

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

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

Example Scripts

The following are example scripts:

- Time out an application on a specific date.
- Run a `.bat` file from a network share inside the virtual environment.
- Modifying the virtual registry.
- Import `.reg` file at runtime.
- Stop a virtual service when the main application quits.
- Copy an external system configuration file into the virtual environment on startup.

To use an example

1 Save contents to a plain text file with the `.vbs` extension in the same directory as your `Package.ini` file (produced by Setup Capture).

It doesn't matter what filename you use, all `.vbs` files are added to the package at build time and run.

2 Rebuild the application.

.bat example

This script executes an external `.bat` file from a network share inside of the virtual environment. The `.bat` file can make modifications to the virtual environment by copying files, deleting files, or apply registry changes using `regedit /s regfile.reg`. Make sure you only execute this for the first parent process otherwise each copy of `cmd.exe` executes the script and an infinite recursion will develop.

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

timeout example

Prevent an application from being used after a specified date.

This check is performed on the parent process startup as well as any child process startup.

```
ExpirationDate = CDate("03-20-07")

if Date >= ExpirationDate then
 msgbox "This application has expired, please contact Administrator"
 ExitProcess 0
end if
```

Registry Modify

This example modifies the virtual registry at runtime so that an external ODBC driver can be loaded from the same directory where the package executable is located.

Get path to package executable files.

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

Find last slash in path and grab just the characters before this.

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

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

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

Modify the virtual registry so that it points to this location. This causes the application to load the DLL from an external location.

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

.reg example

Import the registry values from an external `.reg` file into the virtual registry at runtime.

```
Function OnFirstParentStart
ExecuteVirtualProcess "regedit /s c:\tmp\somereg.reg"
End Function
```

Stopping Service

Stop a virtual or real service when the main application quits.

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

Copyfile Example

The following scripts demonstrate how you can copy a configuration file located in the same directory as your captured executable into the virtual file system each time the application starts. One use for this is to enable you to have an external configuration file that is easy to edit after deployment. Because the copy occurs each time you run the application, any edits to the external version are reflected in the virtual version.

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

By putting this code in the `OnFirstParentStart` function, it is only called once per execution. Otherwise it is executed for every child process as well.

```
Function OnFirstParentStart
```

`TS_ORIGIN` is set by ThinApp to indicate the full path to a captured executable package. A virtual application sets the TS_ORIGIN variable to the physical path of the main data container.

If you have a virtual application consisting of a main.exe and a shortcut.exe, both of them are placed in `C:\VirtApp`. When you run `C:\VirtApp\main.exe`, the `TS_ORIGIN var` will be set to `C:\VirtApp\main.exe`. When you run `C:\VirtApp\shortcut.exe`, the `TS_ORIGIN` environment variable will be set to `C:\VirtApp\main.exe` (it is always set to the main data container, even if you create a shortcut). This happens early during process startup. When you run VBScripts that are included in the package, the variable has already been set and it is available to the VBScripts.

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

Separate the filename from `TS_ORIGIN` by finding the last backslash and remove everything after this.

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

The source file to copy into the virtual environment is the package path plus `config.ini.`

```
SourceFile = SourcePath + "Config.ini"
```

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

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

Use the file `systemObject` to check to make sure 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 virtual directory `%ProgramFilesDir%\MyApplication` is in the package.

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

System Registry Example

This script adds a value to the system registry.

Create a `.reg` file and execute "`regedit /s`" as an external process so it accesses the system registry instead of the virtual registry.

```
Function OnFirstParentStart
```

Create the `.reg` file in a place that has `IsolationMode` set to `Merged`, so it can be easily accessed from both the virtual environment (this script) and the system environment (`regedit /s`). One directory that has a `Merged IsolationMode` by default is the `%Personal%` directory, so create the `.reg` file there.

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

Construct the `.reg` file.

```
RegFile.WriteLine("Windows Registry Editor Version 5.00")
RegFile.WriteBlankLines(1)
RegFile.WriteLine("[HKEY_CURRENT_USER\Software\Thinapp\demo]")
RegFile.WriteLine(chr(34) and "InventoryName" and chr(34) and "=" and chr(34)
and GetBuildOption("InventoryName") and chr(34))
RegFile.Close
```

Enter the information in the system registry, wait until it is done.

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

Clean up.

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

API Reference

This chapter contains API reference information for ThinApp.

AddForcedVirtualLoadPath

Function AddForcedVirtualLoadPath(Path)

This function instructs ThinApp to load all DLLs from the specified path as virtual DLLs even if they are not located in the package. Use this function if the application needs to load external DLLs that have dependencies on DLLs located inside the package.

Parameters

Path

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

Example

Load any DLL located in the same directory as the executable as a virtual DLL.

TS_ORIGIN is the path from which the executable is running.

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

To delete the filename from TS_ORIGIN, so find the last backslash and remove everything after it.

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

Tell ThinApp to load all DLLs in the same directory (or deeper) from where the source executable is.

This allows us to drop additional files in the SourcePath tree and have them resolve imports against virtual DLLs.

```
AddForcedVirtualLoadPath(SourcePath)
```

ExitProcess

Sub ExitProcessExitCode

This function quits the current process and sets the specified Error Code.

Parameters

ExitCode

[in] The error code to set. This information might be available to a parent process. Typically a value of 0 is used to indicate no error.

Example

Exit the process and indicate success.

```
ExitProcess 0
```

NOTE As the process exits, the scripting system receives its `Function OnEndProcess` callback. Also, any DLLs that are loaded run their termination code so they can properly clean up.

ExpandPath

Function ExpandPath(InputPath)

This function converts a path from macro format to system format.

Parameters

InputPath

[in] A path in macro format.

Returns

The expanded macro path in system format.

Example

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

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

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

ExecuteExternalProcess

Function ExecuteExternalProcess(CommandLine)

This function executes a command outside of the virtual environment. It can be used if you need to make real system changes.

Parameters

Command Line

[in] CommandLine represents the application and command line parameters to execute outside of the virtual environment.

Returns

Integer process ID, the process ID can be used with WaitForProcess ("WaitForProcess" on page 156).

Example

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

Execute a command that requires quotes in the command-line.

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

ExecuteVirtualProcess

Function ExecuteVirtualProcess(CommandLine)

This function executes a command inside of the virtual environment. It can be used if you need to make changes to the virtual environment.

Parameters

CommandLine

[in] `CommandLine` represents the application and command line parameters to execute outside of the virtual environment.

Returns

Integer process ID, the process ID can be used with `WaitForProcess` ("WaitForProcess" on page 156).

Example

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

Execute a command that requires quotes in the command-line.

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

GetBuildOption

Function GetBuildOption(OptionName)

This function returns the value of a setting specified in the [BuildOptions] section of the `Package.ini` file used for capturing.

Parameters

Option Name

[in] Name of the setting.

Returns

This function returns a string value. If the requested Option Name does not exist an empty string "" is returned.

Example

```
Package.ini contains:
[BuildOptions]
DemoOption=DemoValue
```

The following line would appear in a VBS file:

```
Value = GetBuildOption("DemoOption")
```

Value = "DemoValue"

GetFileVersionValue

Function GetFileVersionValue(Filename, Value)

This function returns version information value from a specific DLL, executable, OCX, etc. This function can be used to determine the internal version number of a DLL or retrieve information about the copyright owner of a DLL, or product name associated with the DLL.

Parameters

Filename

[in] The name of the `filename` whose version information is being retrieved.

Value

[in] The name of the Value to retrieve from the version information section of the specified file

The following values can be retrieved from most DLLs:

Comments
InternalName
ProductName
CompanyName

LegalCopyright
ProductVersion
FileDescription
LegalTrademarks
PrivateBuild
FileVersion
OriginalFilename
SpecialBuild

Returns

This function returns a string value. If the requested Filename does not exist or the specified Value cannot be located in the file, the value of "" is returned.

Example

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

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

End if
```

GetCommandLine

Function GetCommandLine

This function allows you to access the command-line parameters passed to the current running program

Returns

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

Example

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

GetCurrentProcessName

Function GetCurrentProcessName

This function allows you to access the full virtual path name of the current process.

Returns

This function returns a string that represents the full executable path name inside of the virtual environment. Typically this is `c:\Program Files\...` even though the package source might be running from a network share.

Example

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

GetOSVersion

Function GetOSVersion()

This function returns information about the current version of Windows.

Parameters

This function has no parameters.

Returns

This function returns a string in the following format:

```
MAJOR.MINOR.BUILD_NUMBER.PLATFORM_ID OS_STRING
```

`MAJOR` is one the following values

Windows 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 OS.

PLATFORM_ID is one of the following values.

Value = 1 for Windows Me, Windows 98, or Windows 95 (Windows 95 based OS)

Value = 2 for Windows Server 2003, Windows XP, Windows 2000, or Windows NT. (Windows NT based OS)

OS_STRING Represents additional information about the OS such as "Service Pack 2"

Example

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

GetEnvironmentVariable

Function GetEnvironmentVariable(Name)

This function returns the environment variable associated with variable name Name.

Parameters

Name

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

Returns

This function returns the string value associated with the environment variable "name."

Example

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

RemoveSandboxOnExit

Sub RemoveSandboxOnExit(YesNo)

This function set toggles whether to delete the sandbox when the last child process exits.

If you set the `Package.ini` option `RemoveSandboxOnExit=1`, the default clean up for the package with be "Yes." In this case you, can change the clean up to "No" by calling `RemoveSandboxOnExit` with the value of 0. If you did not modify the `Package.ini` option `RemoveSandboxOnExit=1`, the default clean up for the package with be "No." In this case you, can change the clean up to "Yes" by calling `RemoveSandboxOnExit` with the value of 1.

Parameters

Yes No

[in] Do you want to clean up when the last process shuts down? 1=Yes, 0=No

Example

This turns on cleanup

```
RemoveSandboxOnExit 1
```

This turns off cleanup

```
RemoveSandboxOnExit 0
```

SetEnvironmentVariable

Sub SetEnvironmentVariable(Name, Value)

This function set the value of an environment variable.

Parameters

Name

[in] The name of environment variable where the value is to be stored.

Value

[in] The value to be stored.

Example

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

SetfileSystemIsolation

Sub Setfile systemIsolation(Directory, IsolationMode)

This function sets the isolation mode of a directory.

Parameters

Directory

[in] Full path of the directory whose isolation mode is to be set.

IsolationMode

[in] Isolation mode to be set
1 = WriteCopy
2 = Merged
3 = Full

Example

Set the isolation mode of the temp directory to Merged.

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

SetRegistryIsolation

Sub SetRegistryIsolation(RegistryKey, IsolationMode)

This function sets the isolation mode of a registry key.

Parameters

RegistryKey

[in] The registry key whose isolation mode is to be set. Start with "HKLM" for HKEY_LOCAL_MACHINE, "HKCU" for HKEY_CURRENT_USER of "HKCR" for HKEY_CLASSES_ROOT

IsolationMode

[in] Isolation mode to be set
1 = WriteCopy
2 = Merged
3 = Full

Example

Set isolation of `HKEY_CURRENT_USER\Software\Thinapp\Test to Full`

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

WaitForProcess

Function WaitForProcess(ProcessID, TimeOutInMilliSeconds)

This function waits until the specified `ProcessID` has completed execution.

Parameters

ProcessID

[in] The process ID to wait for completion. The process ID can come from `ExecuteExternalProcess` or `ExecuteVirtualProcess`.

TimeOutInMilliSeconds

[in] The maximum amount of time to wait for the process to end before continuing. If 0 is specified, INFINITE is used.

Returns

This function returns an integer
0 = Timeout failed
1 = Process exited
2 = The process does not exists or security denied

Example

```
id = ExecuteExternalProcess("cmd.exe")
WaitForProcess(id, 0)
```

Virtual File System

F

This information about the virtual files system discusses these topics:

Format Stages of the Virtual File System

The virtual file system uses formats in the following stages:

- Build format.

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

- Embedded format.

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

- Sandbox format.

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

  - Changing the timestamp or attributes of a file
  - Opening a file with write access

- Truncating a file
- Renaming or moving a file

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

Merged and Virtual Views of the File System

Isolation modes specify whether ThinApp presents the application with a merged view of the virtual and physical file system or just a view of virtual files. For information about isolation modes, see "Modifying Isolation Modes" on page 26.

Using Folder Macros

ThinApp uses macros to represent file system path locations that might change when the macros run on different operating systems or computers. The use of macros allows 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 handles the change in Windows directories transparently because ThinApp 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 Vista instead of using XP file path styles. This feature allows 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 Folder Macros

Table F-1 lists the available macros. Macros requiring `shfolder.dll` version 5.0 or higher include `%ProgramFilesDir%`, `%Common AppData%`, `%Local AppData%`, `%My Pictures%`, and `%Profile%`. Macros requiring `shfolder.dll` version 6.0 or higher include `%My Videos%`, `%Personal%`, and `%Profiles%`.

ThinApp uses `shfolder.dll` to obtain the location of shell folders. Older versions of `shfolder.dll` do not support some macro names. ThinApp lists any requirements for specific versions of `shfolder.dll` on the host operating system.

Table F-1. Folder Macros

Macro Name	Typical Location
%AppData%	C:\Documents and Settings\<user_name>\ Application Data
%Common AppData%	C:\Documents and Settings\All Users\ Application Data
%Common Desktop%	C:\Documents and Settings\All Users\Desktop
%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
%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
%Local AppData%	C:\Documents and Settings\<user_name>\ Local Settings\Application Data
%My Pictures%	C:\Documents and Settings\<user_name>\ My Documents\My Pictures
%My Videos%	C:\Documents and Settings\<user_name>\My Documents\ My Videos
%NetHood%	C:\Documents and Settings\<user_name>\NetHood
%Personal%	C:\Documents and Settings\<user_name>\My Documents
%Profile%	C:\Documents and Settings\<user_name>

Table F-1. Folder Macros (Continued)

Macro Name	Typical Location
%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
%Recent%	C:\Documents and Settings\<user_name>\ My Recent Documents
%SendTo%	C:\Documents and Settings\<user_name>\SendTo
%Startup%	C:\Documents and Settings\<user_name>\ Start Menu\Programs\Startup
%SystemRoot%	C:\Windows
%SystemSystem%	C:\Windows\System32
%TEMP%	C:\Documents and Settings\<user_name>\ Local Settings\Temp
%Templates%	C:\Documents and Settings\<user_name>\Templates

Processing %SystemRoot%

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%`.

Virtual Registry

G

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

This information discusses these topics:

Format Stages of the Virtual Registry

By storing settings in the virtual registry, ThinApp ensures that settings required by the application are accessible without actually changing the system registry.

The virtual registry uses formats in the following stages:

- Build format.

  The setup capture process generates this format that involves Unicode text files, such as `HKEY_LOCAL_MACHINE.txt`.

- Embedded format.

  The build process converts the build format into the embedded format. ThinApp compresses and stores the read-only binary format inside the application executable file.

- Sandbox format.

  This read-write format stores the differences from the embedded format as the application writes to the registry.

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

Registry Value Data

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

```
TYPE=VALUE
```

TYPE can be one of the following:

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

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

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

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

Example of Macro Expansion

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

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

Text Format for Virtual Registry

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

The format looks like the following:

```
isolation_mode FULL_SUBKEY_NAME

Value=VALUE_NAME
VALUE_TYPE=VALUE_DATA
```

or

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

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

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

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

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

```
453=#00
```

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

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

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

The following are examples:

```
REG_SZ=Both#00
```

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

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

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

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

This example represents a 4-byte binary value.

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

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

The following are examples:

```
REG_SZ~%AppData%
```

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

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

```
REG_SZ~#23AppData#23
```

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

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

##Attributes.ini

This file controls per-registry subkey settings.

RegistryIsolationMode=WriteCopy | Merged | Full

The following is an example:

```
RegistryIsolation=WriteCopy
```

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

The following is an example of command-line usage:

NOTE All parameters are case insensitive.

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

Performing Registry Operations

You can perform the following registry operations:

- "Importing Registry Data from Regedit Format" on page 165
- "Exporting Registry Data to Regedit Format" on page 166
- "Listing all Registry Keys in a ThinApp .tvr File" on page 167
- "Listing Diagnostic Information About a Thinapp.tvr File" on page 168
- "Comparing Virtual Registry Information with Host Computer Registry Information" on page 168
- "Deleting a Registry Subkey" on page 168

Importing Registry Data from Regedit Format

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

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

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

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

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

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

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

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

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

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

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

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

Exporting Registry Data to Regedit Format

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

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

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

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

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

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

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

The following is an example:

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

Listing all Registry Keys in a ThinApp .tvr File

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

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

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

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

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

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

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

The following is an example:

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

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

Listing Diagnostic Information About a Thinapp.tvr File

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

```
VREGTOOL regfile.tvr PrintStats
```

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

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

The following is an example:

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

Comparing Virtual Registry Information with Host Computer Registry Information

Use the following command to compare virtual registry information with the host computer registry information:

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

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

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

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

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

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

Deleting a Registry Subkey

Use the following command to delete a registry subkey:

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

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

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

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

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

Example:

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

Index

C

D

E

G

I

L

M

N

O

P

R

S

T

U

V

W