ThinApp User's Guide

ThinApp 4.5

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

EN-000219-00

vmware®

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

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

The VMware Web site also provides the latest product updates.

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

docfeedback@vmware.com

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

Contents

About This Book

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

Intended Audience

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

Document Feedback

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

Technical Support and Education Resources

The following sections describe the technical support resources available to you. To access the current version of this book and other books, go to http://www.vmware.com/support/pubs.

Online and Telephone Support

To use online support to submit technical support requests, view your product and contract information, and register your products, go to http://www.vmware.com/support.

Customers with appropriate support contracts should use telephone support for the fastest response on priority 1 issues. Go to http://www.vmware.com/support/phone_support.

Support Offerings

To find out how VMware support offerings can help meet your business needs, go to http://www.vmware.com/support/services.

VMware Professional Services

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

1 Installing ThinApp

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

This information includes the following topics:

ThinApp Requirements

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

Operating Systems, Applications, and Systems That ThinApp Supports

ThinApp supports the following operating systems, applications, and systems:

- 32-bit platforms include Windows NT, Windows 2000, Windows XP, Windows XPE, Windows 2003 Server, Windows Vista, Windows Server 2008, Windows 7
- 64-bit platforms include Windows XP 64 bit, Windows 2003 64 bit, Windows Vista 64 bit, Windows Server 2008 64 bit, Windows Server 2008 R2 64 bit, Windows 7 64 bit
- 16-bit applications running on 32-bit Windows operating systems
- 32-bit applications running on 32-bit and 64-bit Windows operating systems
- Terminal Server and Citrix Xenapp

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

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

Device Drivers

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

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

Shell Integration

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

DCOM Services that are Accessible on a Network

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

Global Hook Dynamic Link Libraries

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

Recommendations for Installing ThinApp

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

Using a Clean Computer

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

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

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

Using Virtual Machines for Clean Computers

The easiest way to set up a clean computer is to create a virtual machine. You can install Windows on the virtual machine and take a snapshot of the entire virtual machine in its clean state. After you capture an application, you can restore the snapshot and revert it to a clean virtual machine state that is ready for the next application capture.

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

Using the Earliest Operating System Required For Users

Install ThinApp on a clean computer with the earliest version of the operating system you plan to support. In most cases, the earliest platform is Windows 2000 or Windows XP. Most packages captured on Windows XP work on Windows 2000. In some cases, Windows XP includes some DLLs that Windows 2000 lacks. ThinApp excludes these DLLS from the captured application package if the application typically installs these DLLs.

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

Install ThinApp

Use the ThinApp executable file to install ThinApp.

To install ThinApp software

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

2 Double-click the ThinApp executable file.

3 In the **Patent Lists** dialog box, click **Next**.

4 Accept the license, type the serial number, and type a license display name that appears when you open applications that ThinApp captures.

5 Click **Install**.

ThinApp is installed.

Checking ThinApp Installation Files

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

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

- **AppSync.exe** – Keeps captured applications up to date with the latest available version.
- **logging.dll** – Generates `.trace` files.
- **dll_dump.exe** – Lists all captured applications that are currently running on a system.
- **log_monitor.exe** – Displays the execution history and errors of an application.
- **relink.exe** – Updates existing packages to the latest ThinApp version installed on the system.
- **sbmerge.exe** – Merges runtime changes recorded in the application sandbox with the ThinApp project and updates the captured application.
- **Setup Capture.exe** – Captures and configures applications through a wizard.
- **snapshot.exe** – Compares the preinstallation environment and postinstallation environment during the application capture process.

  ThinApp starts this utility during the setup capture process.

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

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

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

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

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

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

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

Capturing Applications 2

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

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

This information includes the following topics:

Phases of the Capture Process

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

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

Preparing to Capture Applications

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

For target applications that have dependencies on other applications, libraries, or frameworks, you can capture the dependencies or use the Application Link utility to link separate virtual applications at runtime. For information about the Application Link utility, see "Application Link Updates" on page 46.

For target applications that require locale formats, such as a specific date format, you can capture them in an environment with the required locale setting. ThinApp runs virtual applications according to the regional and language settings on the capture system rather than the settings on the system that runs the application. Although you can modify the default locale setting by commenting out the `LocaleIdentifier` parameter in the `Package.ini` file and rebuilding the application, you can avoid complications in the capture environment. For information about the `LocaleIdentifier` parameter, see "LocaleIdentifier" on page 77.

Capturing Applications with the Setup Capture Wizard

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

This information uses Mozilla Firefox as a key example for application capture. For information about Microsoft Office 2007 that extends beyond the basic capture process, see "Guidelines for Packaging Microsoft Office 2007" on page 24.

Create a System Image Before the Application Installation

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

To create a system image before the application installation

1 Download the applications to capture.

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

2 Close any applications, such as virus scans, that might change the file system during the capture process.

3 From the desktop, select **Start** > **Programs** > **VMware** > **ThinApp Setup Capture**.

4 (Optional) In the **Ready to Prescan** dialog box, click **Advanced Scan Locations** to select the drives and registry hives to scan.

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

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

The scanning process takes about 10 seconds for Windows XP.

Rescan the System with the Installed Application

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

To install the application and rescan the system

1 When the **Install Application** page appears, minimize the Setup Capture wizard and install the applications to capture.

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

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

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

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

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

4 Close the application.

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

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

Defining Entry Points as Shortcuts into the Virtual Environment

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

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

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

Entry Points for Troubleshooting

ThinApp provides entry points to troubleshoot your environment.

Debugging an application might involve the following entry points:

- `cmd.exe` – Starts a command prompt in a virtual context that allows you to view the virtual file system.
- `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 executable files in a virtual context. Entry points do not create virtual packages of `cmd.exe`, `regedit.exe`, or `iexplore.exe`.

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

Set Entry Points

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

To set entry points in the Setup Capture wizard

1 On the **Entry Points** page, select the check boxes for user-accessible entry points.

The wizard displays the executable files that were directly accessible through the desktop or **Start** menu shortcuts.

2 (Optional) To debug your environment, select the **Show entry points used for debugging** check box to display the `iexplore.exe`, `regedit.exe`, and `cmd.exe` troubleshooting options.

Set User Groups

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

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

To set user groups in the Setup Capture wizard

1 On the **Groups** page, limit the user access to the application.

   a Select **Only the following Active Directory groups**.

   b Click **Add** to specify Active Directory object and location information.

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

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

Defining Isolation Modes for the Physical File System

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

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

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

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

Applying Merged Isolation Mode for Modifications Outside the Package

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

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

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

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

- %SystemRoot%
- %SystemSystem%

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

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

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

Applying WriteCopy Isolation Mode To Prevent Modifications Outside of the Package

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

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

When you select WriteCopy isolation in the Setup Capture wizard, ThinApp completes the following operations:

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

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

Set File System Isolation Modes

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

For information about Full isolation and registry isolation that are available only outside of the Setup Capture wizard, see "DirectoryIsolationMode" on page 58 and "RegistryIsolationMode" on page 59.

To set file system isolation modes in the Setup Capture wizard

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

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

Storing Application Changes in the Sandbox

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

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

Customize the Sandbox Location

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

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

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

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

To customize the sandbox location in the Setup Capture wizard

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

Send Anonymous Statistics to VMware

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

To send anonymous statistics to VMware

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

Customize ThinApp Project Settings

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

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

To customize project settings in the Setup Capture wizard

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

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

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

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

Defining Package Settings

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

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

Defining the Primary Data Container

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

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

Generating MSI Packages in the Capture Process

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

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

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

Compressing Packages in the Capture Process

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

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

Customize Package Settings

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

To customize package settings in the Setup Capture wizard

1 On the **Package Settings** page, select the primary data container from the list that is based on your executable file entry points.

- If the size of the primary container is smaller than 200MB, ThinApp creates a `.exe` file as the primary container. For a small application such as Firefox, any `.exe` file can serve as the main data container.
- If the size of the primary container is larger than 200MB, ThinApp creates a separate `.dat` file as the primary container because Windows XP and Windows 2000 cannot show shortcut icons for large `.exe` files. Generating separate small `.exe` files along with the `.dat` file fixes the problem.
- If the size of the primary container is between 200MB and 1.5GB, ThinApp creates the default `.dat` file unless you select a `.exe` file to override the default `.dat` file.

2 (Optional) If you select a `.exe` file to override the default `.dat` file when the size of the primary container is between 200MB and 1.5GB, ignore the generated warning.

Selecting a `.exe` file allows all applications to work properly but might prevent the proper display of icons.

3 (Optional) If you cannot select a primary data container, type a primary data container name to generate a `.dat` file.

If you plan to use the Application Sync utility to update a captured application, ThinApp uses the primary data container name during the process. You must use the same name across multiple versions of the application. You might not be able to select the same primary data container name from the list. For example, Microsoft Office 2003 and Microsoft Office 2007 do not have common entry point names.

4 (Optional) Select the **Generate MSI package** check box and change the MSI filename.

5 (Optional) To create a smaller executable package for locations such as a USB device, select the **Compress virtual package** check box.

6 Click **Save**.

Opening Project and Parameter Files

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

For example, if you capture Firefox 2.0.0.3, you might browse the `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3` directory to update a setting, such as an Active Directory specification, in the `Package.ini` file that contains the parameters set during the capture process. For information about updating settings, see "Advanced Package Configuration" on page 23.

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

Build Virtual Applications

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

To build virtual applications in the Setup Capture wizard

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

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

2 (Optional) To prevent a build from taking place, select the **Skip the build process** check box.

You can build the package at a later time with the `build.bat` file in the virtual application folder. For example, a Firefox 2.0.0.3 path to the `build.bat` file might be `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3\build.bat`.

3 Click **Build** to build an executable package or MSI package containing the files you installed during the capture process.

4 (Optional) Deselect the **Open folder containing project executables after clicking Finish** check box to view the executable files and MSI files at a later time.

5 Click **Finish**.

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

Advanced Package Configuration

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

Modifying Settings in the Package.ini File

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

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

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

- `DirectoryIsolationMode` – Sets the isolation mode to `Merged`, `WriteCopy`, or `Full`.
- `PermittedGroups` – Restricts use of an application package to a specific set of Active Directory users.
- `SandboxName` – Identifies the sandbox.

  You might keep the name for incremental application updates and change the name for major updates.
- `SandboxPath` – Sets the sandbox location.
- `SandboxNetworkDrives` – Specifies whether to direct write operations on the network share to the sandbox.
- `RequiredAppLinks` – Specifies a list of external ThinApp packages to import to the current package at runtime.
- `OptionalAppLinks` – Specifies a list of external ThinApp packages to import to the current package at runtime.

For information about all `Package.ini` parameters, see Chapter 5, "Configuring Package Parameters," on page 55.

Modify the Package.ini File

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

To modify the Package.ini file

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

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

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

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

  ```
  RemoveSandboxOnExit=1
  ```

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

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

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

Modifying Settings in the ##Attributes.ini File

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

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

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

Modify the ##Attributes.ini File

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

To modify the ##Attributes.ini file

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

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

Guidelines for Packaging Microsoft Office 2007

Although the process to capture Microsoft Office 2007 might depend on your environment, you can refer to basic guidelines.

For more information about the capture and configure process for Microsoft Office and environment needs, see the VMware ThinApp community blog.

Requirements for Packaging Microsoft Office 2007

The process of capturing Microsoft Office 2007 assumes several requirements beyond the standard requirements for creating ThinApp packages.

The following requirements involve Microsoft software or printer dependencies:

- A clean virtual machine with a supported Windows operating system.
- A licensed copy of Microsoft Office 2007.
- A Microsoft Office 2007 volume license key.

  You must activate a retail license key on each machine the package runs on.
- Windows Installer 4.5.
- Microsoft .NET Framework 2.0.

  You must install Microsoft .NET after the prescan operation in the Setup Capture wizard and before the installation of Microsoft Office 2007.
- Any required printer, such as a corporate printer, installed before the prescan operation in the Setup Capture wizard.

Capturing Microsoft Office 2007

The key differences between capturing Microsoft Office 2007 and capturing basic applications involve modifying Microsoft Office and blocking child processes.

The capture process for Microsoft Office 2007 involves the following steps:

1 "Customize Microsoft Office 2007 Installation Options" on page 25

2 "Disable Child Processes for Microsoft Office 2007" on page 25

3 "Set Capture Options for Microsoft Office 2007" on page 26

This process assumes that you are familiar with the Setup Capture wizard. You can modify the process according to your environment.

Customize Microsoft Office 2007 Installation Options

Starting the capture process for Microsoft Office 2007 involves customizing the Microsoft Office installation.

To customize the installation of Microsoft Office 2007

1 Copy Microsoft .NET 2.0, Windows Installer 4.5, and ThinApp installation files to the virtual machine.

2 Copy the Microsoft Office 2007 installation files to the virtual machine or load the Microsoft Office 2007 ISO to the virtual machine.

3 Install Windows Installer 4.5 and reboot the system.

4 Install ThinApp.

5 (Optional) Install a required printer, such as a corporate printer.

6 Run the Setup Capture wizard until you complete the prescan operation.

7 On the **Install Application** page of the Setup Capture wizard, minimize the wizard and install Microsoft .NET 2.0.

8 (Optional) If the Microsoft .NET installation generates the `mscorsvw.exe` process that continues for an extended period, stop the process with the `ngen.exe` tool.

```
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\ngen.exe executequeueditems
```

9 Start the Microsoft Office 2007 setup wizard.

The installation setup involves typing the Microsoft Office 2007 license key and accepting the license agreement.

10 (Optional) On the **File Location** tab, install the software in a fixed directory, such as `C:\Office`, rather than a default directory to prevent possible access errors for online help in Microsoft Office applications.

The access errors apply only to Office applications deployed on 64-bit operating systems.

11 On the **Choose the installation you want** dialog box, click the **Customize** button.

12 On the **Installation Options** tab, customize the options to capture the appropriate files and prevent a Microsoft Office printer error.

a Select **Microsoft Office** > **Run all from My Computer**.

b Click the plus (+) symbol next to the **Office Tools** menu.

c Click the plus (+) symbol next to the **Microsoft Office Document Imaging** menu.

d In the **Microsoft Office Document Image Writer** drop-down menu, select **Not Available**.

e Click **Install Now** to install Microsoft Office 2007.

f Close the installation setup.

Disable Child Processes for Microsoft Office 2007

Capturing Microsoft Office involves disabling child processes before the postscan stage of the Setup Capture wizard.

Some child processes prevent you from closing the sandbox.

To disable child processes for Microsoft Office 2007

1 Disable the `ctfmon.exe` process to prevent child processes from preventing you from closing the sandbox.

a In the Windows control panel, select **Regional and Language Options**.

b On the **Languages** tab, click **Details**.

c On the **Advanced** tab, select the **Turn off advanced text services** check box.

d From the desktop, select **Start** > **Run** and run the `Regsvr32.exe /u msimtf.dll` command.

e From the desktop, select **Start** > **Run** and run the `Regsvr32.exe /u Msctf.dll` command.

2 Disable the `mdm.exe` process to prevent an issue with child processes that prevent you from closing the sandbox.

a In Internet Explorer, select **Tools** > **Internet Options**.

b On the **Advanced** tab, select the **Disable Script Debugging (Other)** check box and the **Disable Script Debugging (Internet Explorer)** check box.

3 (Optional) Use VMware Workstation to take a snapshot of the virtual machine.

This feature creates an image that you can revert to when you add plug-ins or updates.

Set Capture Options for Microsoft Office 2007

The final phase of the capture process for Microsoft Office 2007 involves the ThinApp postscan operation and Setup Capture wizard options to create the package.

The following examples for options might apply to Microsoft Office 2007:

- The location of the entry points is `%ProgramFilesDir%\Microsoft Office\Office12`.
- The primary data container name is `Microsoft Office 2007.dat`.
- The isolation mode is Merged mode.

  WriteCopy isolation mode is appropriate when you do not want to leave traces of files outside of the sandbox.
- The deployment of Microsoft Office 2007 uses an MSI package.

To set capture options for Microsoft Office 2007

Maximize the Setup Capture wizard and complete the capture process.

Configure Microsoft Office 2007

Configuring Microsoft Office 2007 outside of the capture process involves deleting directories and updating project files.

You can make configuration changes to the Microsoft Office 2007 package when the changes are appropriate for your environment.

To configure Microsoft Office 2007

- (Optional) To save space, delete the following directories and files that Microsoft Office does not require:
  - `%COOKIES%`
  - `%HISTORY%`
  - `%INTERNET CACHE%`
  - `%PROFILE%`
  - `%COMMON APPDATA%\VMware`
  - `.msi` and `.msp` files in `%SystemRoot%\Installer`
- (Optional) If you do not want to modify your Office 2007 package with customizations involving user names or company names, delete the contents of the `%APPDATA%` directory except the `##Attributes.ini` file.

  This deletion enforces a clean configuration for the user.

- (Optional) In the HKEY_CURRENT_USER.txt file in the project files, add the isolation_full HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Outlook\Security entry anywhere in the file if it does not exist.
- (Optional) Add the following entries below the isolation_full HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Outlook\Security entry.

  ```
  Value=OutlookSecureTempFolder
  REG_SZ~%Profile%\Local Settings\OutlookTemp#2300
  ```

- (Optional) If you activate any Application Sync parameter and the CompressionType parameter, deactivate compression in the ##Attributes.ini file of the %Program Files Common%\Microsoft Shared\OFFICE12\ directory.

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

  Without this modification, an error might affect the %Program Files Common%\Microsoft Shared\OFFICE12\ODSERV.EXE file.

Deploying Applications 3

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

This information includes the following topics:

ThinApp Deployment Options

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

Deploying ThinApp With Deployment Tools

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

When you use any of these tools, you can create MSI files for the captured applications and follow the same process you use to deploy native MSI files. See deployment instructions from the tool vendors. For information about MSI files, see "Building an MSI Database" on page 33.

Deploying ThinApp in the VMware View Environment

You can use VMware View to distribute ThinApp packages.

The workflow for deploying packages might involve the following tasks:

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

- Creating a login script that queries applications entitled to the user and runs the `thinreg.exe` utility with the option that registers the applications on the local machine. Login scripts are useful for nonpersistent desktops. See "Establishing File Type Associations with the thinreg.exe Utility" on page 30.
- Controlling user access to fileshares. IT administrators might control access by organizing network shares based on function and associating permissions with network shares based on those functional boundaries.

Deploying ThinApp on Network Shares

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

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

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

Deploying ThinApp Using Executable Files

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

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

Establishing File Type Associations with the thinreg.exe Utility

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

The `thinreg.exe` utility 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, the `thinreg.exe` utility allows you to see the same menu options for a `.doc` file in a native environment.

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

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

Application Sync Effect on the thinreg.exe Utility

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

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

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

Run the thinreg.exe Utility

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

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

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

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

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

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

For information about `thinreg.exe` parameters, see "Optional thinreg.exe Parameters" on page 31.

To run the thinreg.exe utility

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

2 From the command line, type the `thinreg.exe` command.

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

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

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

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

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

Optional thinreg.exe Parameters

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

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

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

For information about the `PermittedGroups` setting and support for Active Directory groups, see "PermittedGroups" on page 63.

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

Table 3-1. Optional thinreg.exe parameters

Parameter	Purpose	Sample Usage
/a, /allusers	Registers a package for all users. If an unauthorized user attempts to run the application, a message informs the user that he or she cannot run the application.	`thinreg.exe /a "\\<server>\<share>\Microsoft Office 2007 Word.exe"`
/q, /quiet	Prevents the display of an error message for an unrecognized command-line parameter.	`thinreg.exe /q <unknown_option>`

/u, /unregister, /uninstall	Unregisters a package.
	This command removes the software from the Add/Remove Programs control panel applet.
		Unregister Microsoft Word for the current user.
		`thinreg.exe /u "\\<server>\<share>\Microsoft Office 2007 Word.exe"`
		Unregister all Microsoft Office applications for the current user and remove the Add/Remove Programs entry.
		`thinreg.exe /u "\\server\share\Microsoft Office *.exe"`
		If a user registers the package with the /a parameter, you must use the /a parameter when unregistering the package.
		`thinreg.exe /u /a *.exe`

/r, /reregister	Reregisters a package.
	Under typical circumstances, the thinreg.exe utility can detect whether a package is already registered and skips it. The /r option forces the thinreg.exe utility to reregister the package.
		`thinreg.exe /r "\\<server>\<share>\Microsoft Office 2007 Word.exe"`
		If a user registers the package with the /a parameter, you must use the /a when reregistering the package.
		`thinreg.exe /r /a *.exe`

/k, /keepunauthorized, /keep	Prevents the removal of registration information even if you are no longer authorized to access an application package. Without this option, the thinreg.exe utility removes the registration information for that package if it detects you are no longer authorized to access the package. ThinApp stores authorization information in the PermittedGroups parameter of the Package.ini file.	`thinreg.exe /k "\\<server>\<share>\Microsoft Office 2007 Word.exe"`
/noarp	Prevents the creation of an entry in the Add/Remove Programs control panel applet.	`thinreg.exe /q /noarp "\\<server>\<share>\Microsoft Office 2007 Word.exe"`
/norelaunch	Starts the thinreg.exe utility on Microsoft Vista without elevated privileges. Standard users can start the utility without a user account control (UAC) pop-up window. When the thinreg.exe utility detects a need for more privileges, such as the privileges required for the /allusers parameter, the utility restarts itself as a privileged process and generates a UAC pop-up window. The /norelaunch option blocks this restart process and causes the registration to fail.	`thinreg.exe /q /norelaunch "\\<server>\<share>\Microsoft Office 2007 Word.exe"`

Building an MSI Database

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

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

Customizing MSI Files with Package.ini Parameters

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

The following parameters can affect MSI configuration:

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

  For example, include `MSIInstallDirectory=C:\Program Files\` in the `Package.ini` file.

- The `MSIDefaultInstallAllUsers` parameter sets the installation of the package for individual users. ThinApp installs the package in the `%AppData%` user directory.

  For example, include `MSIDefaultInstallAllUsers=0` in the `Package.ini` file.

  For more information about this parameter, see "Specifying a Database Installation for Individual Users and Machines" on page 34.

- The `MSIFileName` parameter names the package.

  For example, include `MSIFilename=Firefox30.msi` in the `Package.ini` file.

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

  For example, include `MSIRequireElevatedPrivileges=1` in the `Package.ini` file.

- The `MSIProductCode` parameter makes it easier to install a new version of the application. An MSI database contains a product code and an upgrade code. When you update a package, keep the original value of the `MSIUpgradeCode` parameter.

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

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

Regardless of the parameter values specified at build time, you can override the settings at deployment time. See "Force MSI Deployments for Each User or Each Machine" on page 34. For more information about MSI parameters, see "Configuring MSI Files" on page 86.

Modify the Package.ini File to Create MSI Files

You must add an entry for the `MSIFilename` parameter to generate MSI files.

For more information about MSI parameters, see "Customizing MSI Files with Package.ini Parameters" on page 33 and "Configuring MSI Files" on page 86.

To modify the MSI parameters

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

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

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

2 (Optional) Update other MSI parameters.

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

Specifying a Database Installation for Individual Users and Machines

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

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

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

For more information about the `MSIDefaultInstallAllUsers` parameter, see "MSIDefaultInstallAllUsers" on page 86.

Force MSI Deployments for Each User or Each Machine

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

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

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

To force MSI deployments for individual users

From the command line, type the `msiexec /i <database>.msi ALLUSERS=""` command.

To force MSI deployments for all users on a machine

From the command line, type the `msiexec /i <database>.msi ALLUSERS=1` command.

Override the MSI Installation Directory

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

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

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

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

To override the MSI installation directory

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

Deploying MSI Files on Microsoft Vista

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

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

Controlling Application Access with Active Directory

You can control access to applications using Active Directory groups.

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

- You must be connected to your Active Directory domain during the build process and the groups you specify must exist. ThinApp looks up the SID value during the build.
- If you delete a group and 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 users can log into their machines, authentication still works. Use a group policy to set the period when cached credentials are valid.
- Cached credentials might not refresh on clients until the next Active Directory refresh cycle. You can force a group policy on a client by using the `gpupdate` command. This command refreshes local group policy, group policy, and security settings stored in Active Directory. You might log out before Active Directory credentials are recached.
- Certain groups, such as the Administrators group and Everyone group, have the same SID on every Active Directory domain and workgroup. Other groups you create have a domain-specific SID. Users cannot create their own local group with the same name to bypass authentication.
- Active Directory Domain Services define security groups and distribution groups. If you use nested groups, ThinApp can only support nested security groups.

Package.ini Entries for Active Directory Access Control

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

When you start a captured application, the `PermittedGroups` parameter checks whether a user is a member of a specified Active Directory group. If the user is not a member of the Active Directory group, Thinapp does not start the application. For information about restricting packages to Active Directory groups, see "PermittedGroups" on page 63.

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

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

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

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

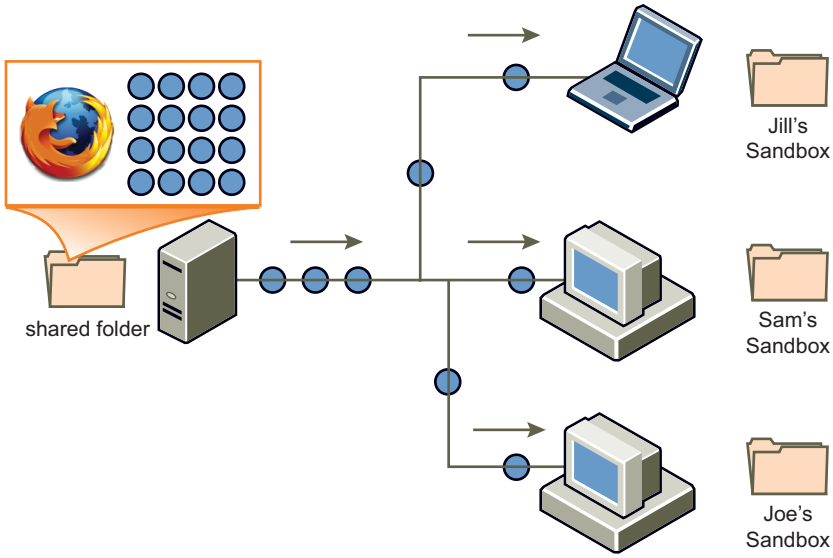
Starting and Stopping Virtual Services

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

Using ThinApp Packages Streamed from the Network

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

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



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

How ThinApp Application Streaming Works

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

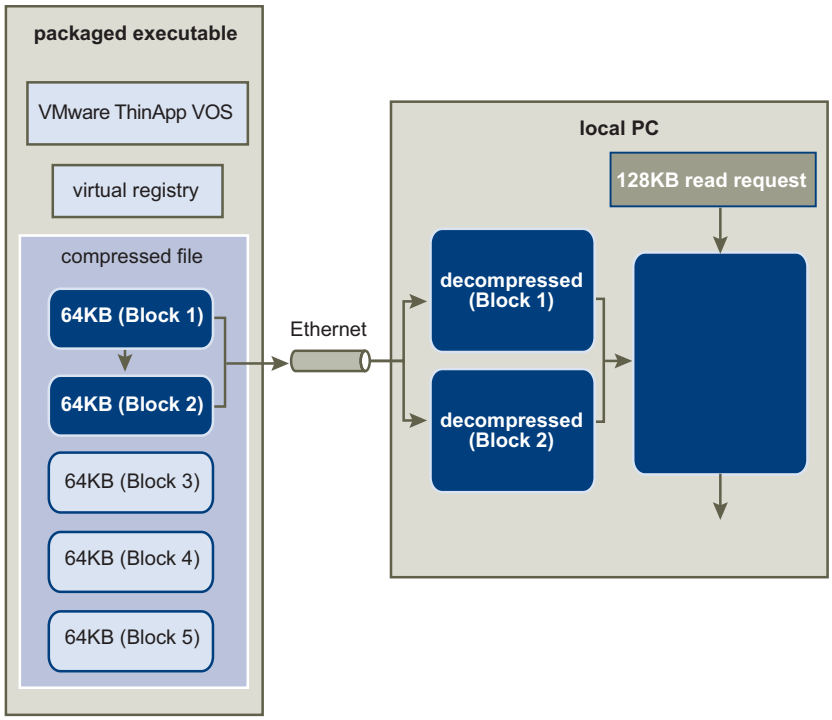
After a client computer receives data, ThinApp decompresses the data directly to memory. Because ThinApp does not write data to the disk, the process is fast. A large package does not necessarily take a long time to load over the network and the package size does not affect the startup time of an application.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

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

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

When the application makes subsequent read requests for the same data, the Windows disk cache provides data without requiring a network read operation. If the client computer runs low on memory, Windows discards some of its disk cache and provides the memory resource to other applications.

Figure 3-2. Application Streaming



Requirements and Recommendations for Streaming Packages

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

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

- Use a URL to deploy the applications.
- Use a desktop deployment solution to push the package to the background. Allow the application to run only after the entire package downloads.

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

Security Recommendations for Streaming Packages

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

A common configuration is to place the user sandbox on another central storage device. The user can use any computer and 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 ThinApp Packages from the Network

Users can access packaged applications through the network.

To stream packages from the network

1 Place the ThinApp package in a location accessible to client computers.

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

Using Captured Applications with Other System Components

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

Performing Paste Operations

Review the following paste operations and limitations with ThinApp:

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

Accessing Printers

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

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

Accessing Drivers

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

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 the printer driver installation. Other applications do not detect a new printer driver unless the driver is installed.

Accessing the Local Disk, the Removable Disk, and Network Shares

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

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

Accessing the System Registry

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

Accessing Networking and Sockets

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

Using Shared Memory and Named Pipes

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

ThinApp can isolate shared memory objects and synchronization objects. This isolation makes them invisible to other applications, and other application objects are invisible to a captured application.

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

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

Starting Services

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

Using File Type Associations

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

Sample Isolation Mode Configuration Depending on Deployment Context

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

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

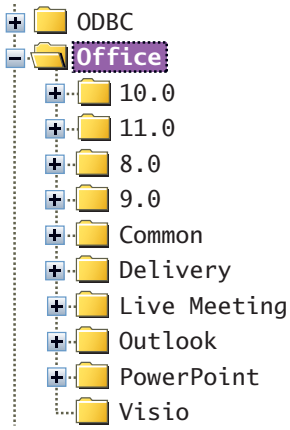
Table 3-2. Sample Problems and Solutions That Use Isolation Modes

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

View of Isolation Mode Effect on the Windows Registry

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

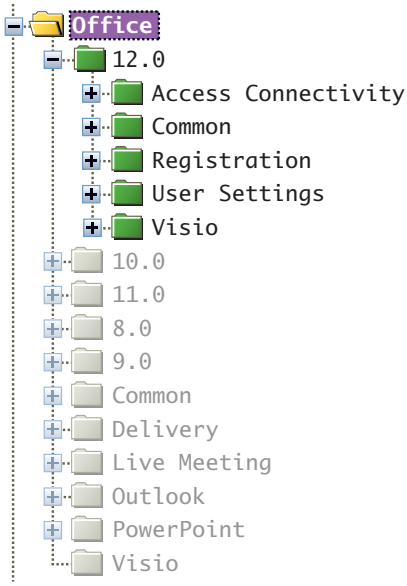
Figure 3-3. Windows Registry as seen by Windows Regedit



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

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

Figure 3-4. Windows Registry as seen by the captured Microsoft Visio 2007



Updating and Linking Applications 4

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

This information includes the following topics:

Application Updates That the End User Triggers

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

Application Sync Updates

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

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

Using Application Sync in a Managed or Unmanaged Environment

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

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

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

Update Firefox 2.0.0.3 to Firefox 3 with Application Sync

This example shows the Application Sync update process for Firefox.

The update process involves modifying the `Package.ini` file. The `AppSyncURL` parameter requires a URL path. ThinApp supports HTTP, HTTPS, and file protocols. For information about all Application Sync parameters, see "Configuring Application Updates with Application Sync" on page 83.

To update Firefox 2.0.0.3 to Firefox 3

1 Capture Firefox 2.0.0.3 and Firefox 3 into separate packages.

2 Verify that the primary data container name is the same for both packages.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Fix an Incorrect Update with Application Sync

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

To fix an incorrect update

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

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

Application Sync Effect on Entry Point Executable Files

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

Updating thinreg.exe Registrations with Application Sync

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

Maintaining the Primary Data Container Name with Application Sync

The Application Sync utility requires that the name of the primary data container, the file that stores virtual files and registry information, is the same for the old and new versions of an application. For example, you cannot have an old version with `Microsoft Office Excel 2003.exe` as the primary data container name while the new version has `Microsoft Office 2007.dat` as the primary data container name. To verify the name of the primary data container, see the `ReadOnlyData` parameter in the `Package.ini` file. For more information about the primary data container, see "Defining Entry Points as Shortcuts into the Virtual Environment" on page 17.

Completing the Application Sync Process When Applications Create Child Processes

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

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

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

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

- Log out and log in to the machine to stop the child processes.
- Create a script to end the child processes.

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

Prevent the startup of the ctfmon.exe process for Microsoft Office and Internet Explorer

Preventing the startup of the `ctfmon.exe` process requires knowledge of the ThinApp sandbox and `sbmerge.exe` utility. For information about the `sbmerge.exe` utility, see "Updating Applications with Runtime Changes" on page 51.

To prevent the startup of the ctfmon.exe process

1 If you did not activate the `cmd.exe` entry point during the capture process, set the `Disabled` parameter for the `cmd.exe` entry in the `Package.ini` file to 0 and rebuild the package with the `build.bat` utility.

This generates an executable file for the `cmd.exe` entry point in the `/bin` directory.

2 Copy the `/bin` directory in the captured application directory to a clean virtual machine or delete the sandbox for the Microsoft Office package.

3 Double-click the `cmd.exe` entry point.

4 In the Windows command processor, run the `INTL.CPL` command.

5 In the **Languages** tab of the **Regional and Languages** dialog box, click **Details**.

6 In the **Advanced** tab of the **Text Services and Input Languages** dialog box, select the **Turn off advanced text services** check box.

7 Click **OK** in all the open dialog boxes and leave the Windows command processor open.

8 Unregister the `MSIMTF.dll` and `MSCTF.dll` files with the `REGSVR32.EXE/U <DLL_file>` command.

See knowledge base article 282599 in the Microsoft Web site.

9 Close the Windows command processor.

10 If the virtual machine does not reside on the same machine where ThinApp is installed, copy the sandbox from the package to the packaging system.

The default sandbox location is `%APPDATA%\Thinstall`.

11 From the standard command prompt on the packaging system, use the `sbmerge.exe` utility to merge the updated sandbox with the package.

A sample command is `SBMERGE APPLY -ProjectDir "C:\Program Files\VMware \VMware ThinApp\Captures\Microsoft Office Professional 2007" -SandboxDir "%APPDATA%\Thinstall\Microsoft Office Pro 2007"`.

12 Rebuild the package and test the package on a clean virtual machine to confirm that the `ctfmon.exe` process no longer exists.

Application Link Updates

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

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

The Application Link utility is useful for the following objects:

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

  For example, you can link .NET to an application even if the local machine for the application does not allow the installation of .NET or already has a different version of .NET.

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

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

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

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

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

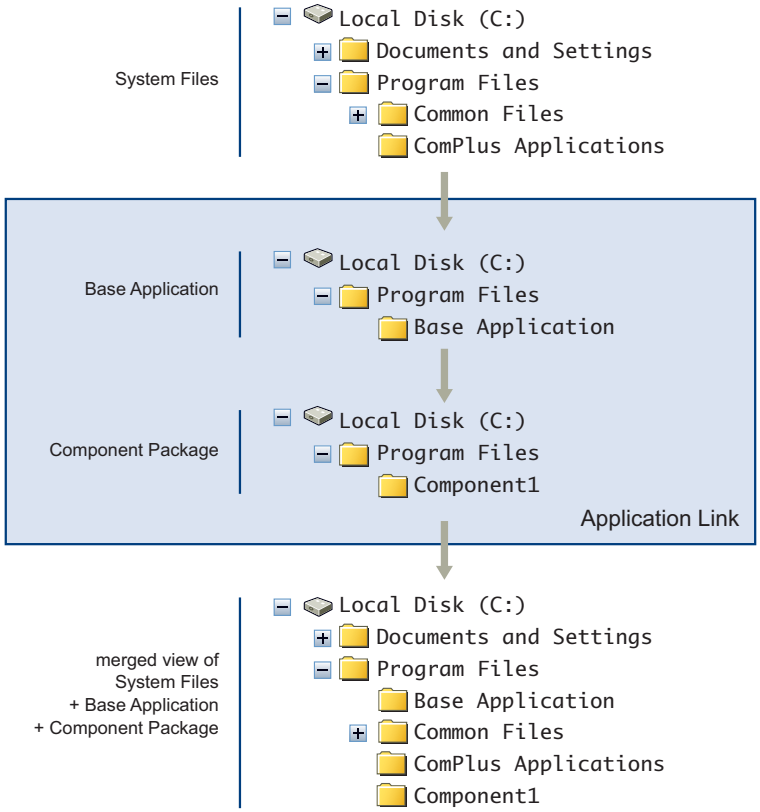
- **Hot fixes and service packs** – Link updates to an application and roll back to a previous version if users experience significant issues with the new version. You can deploy minor patches to applications as a single file and reduce the need for rollbacks.

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

View of the Application using Application Link

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

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



Link a Base Application to the Microsoft .NET Framework

Review this sample workflow to link a base application, `MyApp.exe`, to a separate package that contains the Microsoft .NET 2.0 Framework. Make sure that the base application capture process does not include the Microsoft .NET 2.0 Framework. For information about the process of capturing an application, see Chapter 2, "Capturing Applications," on page 15.

For information about required and optional Application Link parameters and formats in the `Package.ini` file, see "Configuring Dependent Applications with Application Link" on page 81.

To link an application to Microsoft .NET

1 Capture the installation of the .NET 2.0 Framework.

During the capture process, you must select at least one user-accessible entry point.

2 Rename the `.exe` file that ThinApp produces to a `.dat` file.

This renaming prevents users from accidentally running the application.

The name of the `.dat` file you select does not matter because users do not run the file directly. For example, use `dotnet.dat`.

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

4 Capture the base application by using the same physical system or virtual machine with the .NET framework already installed.

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

6 Open the `Package.ini` file in the captured application folder for the base application.

7 Enable the `RequiredAppLinks` parameter for the base application by adding the following line after the `[BuildOptions]` entry.

```
RequiredAppLinks=dotnet.dat
```

Application Link parameters must reference the primary data container of the application you want to link to. You cannot reference shortcut `.exe` files because these files do not contain any applications, files, or registry keys.

8 Rebuild the .NET 2.0 and base application packages.

a Double-click the `build.bat` file in `C:\Captures\MyApp`.

b Double-click the `build.bat` file in `C:\Captures\dotnet`.

Running these batch files builds separate ThinApp packages.

9 Deploy the applications to an end-user desktop in `C:\Program Files\MyApp`.

a Copy `C:\Captures\MyApp\bin\MyApp.exe` to `\\<end_user_desktop>\<Program_Files_share>\MyApp\MyApp.exe`.

b Copy `C:\Captures\dotnet\bin\cmd.exe` to `\\<end_user_desktop>\<Program_Files_share>\MyApp\dotnet.dat`.

Set up Nested Links with Application Link

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

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

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

For information about setting up required and optional Application Link parameters in this procedure, see "Configuring Dependent Applications with Application Link" on page 81.

To set up nested links

1 Capture Application A.

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

For example, add `RequiredLinks=\AppFolder\AppB\AppB.exe` to the file.

3 Capture Application B.

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

For example, add `RequiredLinks=\AppFolder\AppC\AppC.exe` to the file.

5 Capture Application C.

If you start Application A, it can access the files and registry keys of Application B and Application B can access the files and registry keys of Application C.

Affecting Isolation Modes with Application Link

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

PermittedGroups Effect on Linked Packages

If you link two applications and you specify a value for the `PermittedGroups` parameter, the user account used for starting the application must be a member of at least one of the Active Directory groups for this parameter in the `Package.ini` files of both applications. For information about the `PermittedGroups` parameter, see "Configuring Permissions" on page 62.

Sandbox Changes for Standalone and Linked Packages

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

Import Order for Linked Packages

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

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

Using `a.exe` and `b.exe` as sample executable files, ThinApp imports linked packages in the following 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 the first set of plug-ins in this list

For information about nested links, see "Set up Nested Links with Application Link" on page 48.

File and Registry Collisions in Linked Packages

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

VBScript Collisions in Linked Packages

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

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

VBScript Function Order in Linked Packages

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

Storing Multiple Versions of a Linked Application in the Same Directory

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

Using Application Sync For a Base Application and Linked Packages

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

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

Application Updates That the Administrator Triggers

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

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

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

Forcing an Application Sync Update on Client Machines

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

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

Updating Applications with Runtime Changes

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

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

Merge Sandbox Changes with Firefox

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

To merge sandbox changes with Firefox 2.0.0.3

1 Capture Firefox 2.0.0.3.

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

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

3 Create a `Thinstall` directory in the `bin` directory for the sandbox location.

4 Start Firefox and make a change to the settings.

For example, change the home page.

5 From the command line, navigate to the directory where the ThinApp project folder resides.

For example, navigate to `C:\Program Files\VMware\VMware ThinApp\Captures\Mozilla Firefox 2.0.0.3`.

6 From the command line, type the `"C:\Program Files\VMware\VMware ThinApp\sbmerge" Print` command.

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

7 From the command line, type the `"C:\Program Files\VMware\VMware ThinApp\sbmerge" Apply` command.

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

sbmerge.exe Commands

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

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

Usage

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

Optional Parameters

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

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 "Search Order for the Sandbox" on page 93. 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 the additional exclusion of content.

Automatic Application Updates

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

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

To merge changes that an auto-update mechanism makes with the original package to build an updated executable file, use the `sbmerge.exe` utility. See "Application Updates That the Administrator Triggers" on page 50.

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 43.

Dynamic Updates Without Administrator Rights

You can update applications dynamically without requiring administrator rights. For example, .NET-based applications that download new DLL files from the Internet as part of their update process must run the `ngen.exe` file to generate native image assemblies for startup performance. In typical circumstances, the `ngen.exe` file writes to `HKLM` and `C:\WINDOWS`, both of which are only accessible with administrator accounts. With ThinApp, the `ngen.exe` file can install native image assemblies on guest user accounts but stores changes in a user-specific directory.

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

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

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

  If you apply patches in the virtual environment on a central packaging machine, you can use the `sbmerge.exe` utility to merge sandbox changes made by the update with the application. See "Application Updates That the Administrator Triggers" on page 50.
- In the captured project.

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

Upgrading Running Applications on a Network Share

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

File Locks

Starting an application locks the executable file package. You cannot replace, delete, or move the application. This file lock ensures that any computer or user who accesses a specific version of an application continues to have that version available as long as the application processes and subprocesses are running.

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

Upgrade a Running Application

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

You do not have to update shortcuts.

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 file location at \\<server>\<share>\Firefox.exe.

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

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

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

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

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

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

Sandbox Considerations for Upgraded Applications

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

Updating the ThinApp Version of Packages

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

relink Examples

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

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

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

```
relink AdobeReader.exe
```

The relink.exe utility can use a wildcard pattern.

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

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

```
relink C:MyPackages
```

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

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

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

Configuring Package Parameters

5

Advanced users can customize the parameters of the virtual application outside of the capture process. Parameters can affect the configuration of build options that include MSI, update, and entry point settings.

The `Package.ini` file is located in the project folder and contains parameters that configure a captured application during the build process. The Setup Capture wizard sets the initial values of some of the `Package.ini` parameters. You can save the `Package.ini` file and build the project to have the parameter changes take effect.

This information includes the following topics:

Package.ini File Structure

The structure of the `Package.ini` file includes sections that apply to all applications or individual applications.

Most parameters must appear under a specific section heading. The `Package.ini` file contains the following headings:

- `[BuildOptions]`
- `[<application>.exe]`
- `[FileList]`
- `[Compression]`
- `[Isolation]`

The `[BuildOptions]` section of the `Package.ini` file applies to all applications. Individual applications inherit these parameters unless the entries specific to applications override these settings. For example, the `[Adobe Reader 8.exe]` section of the `Package.ini` file for an Adobe Reader application might have settings that override the larger `[BuildOptions]` parameters. The application-specific parameters show the application entry points that you create during the build process.

The `[FileList]`, `[Compression]`, and `[Isolation]` parameters act as `[BuildOptions]` parameters but are grouped separately for backward compatibility reasons. You can add the `[FileList]` heading manually to the file when you add the `ExcludePattern` parameter.

Parameters that do not apply to the standard sections can reside under any heading. Parameters do not have to appear in alphabetical order.

Parameters that Apply to Package.ini or ##Attributes.ini Files

You can apply certain parameters to the `Package.ini` file or the `##Attributes.ini` file depending on the requirements to override the `Package.ini` settings at the directory level.

You can use the `DirectoryIsolationMode`, `CompressionType`, and `ExcludePattern` parameters in an `##Attributes.ini` file. The `##Attributes.ini` file exists in the folder macros of the project folder.

For more information about the `##Attributes.ini` file, see "Modifying Settings in the ##Attributes.ini File" on page 23.

Configuring the ThinApp Runtime

You can modify ThinApp parameters for runtime configuration tasks that affect application startup performance and virtual computer names.

NetRelaunch

The `NetRelaunch` parameter determines whether to restart an application from the local disk when you run the application from a network share or removable disk to address the slow startup of applications.

ThinApp sets an initial value of the `NetRelaunch` parameter that detects whether an application runs from a network drive or a removable disk, and uses a stub executable file on the local hard disk to restart the application. This process addresses performance problems that Symantec AntiVirus generates when it tries to perform a complete scan of executable files that start from a network share or removable disk and on executable files that make the initial network connections. The scan can affect start times for large executable files.

Because a large number of desktops have Symantec AntiVirus, ThinApp allows applications to start from a network share without incurring lengthy scan times. When the application runs from a network share or removable disk, ThinApp creates a stub executable file in the directory that the `CachePath` parameter sets on the local disk and restarts the application from this stub executable file. The stub executable file can load the runtime from the large package and read the rest of the application from its original network location. Symantec AntiVirus perceives that the application is local and does not scan the larger executable file on the network share or removable disk.

Examples

If your application is small or you know that Symantec AntiVirus is not installed on the desktops to which you are deploying the application, you can modify the `NetRelaunch` parameter for stronger initial startup performance.

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

RuntimeEULA

The `RuntimeEULA` parameter controls the End User License Agreement (EULA) display for the package. This parameter addresses legacy EULA requirements. VMware does not require a runtime EULA for ThinApp packages.

Do not modify the value of this parameter.

Examples

The `RuntimeEULA` parameter prevents the display of the EULA.

```
[BuildOptions]
;Default: do not show an Eula
RuntimeEULA=0
```

VirtualComputerName

The `VirtualComputerName` parameter determines whether to virtualize the computer name to avoid naming conflicts between the deployment system and the capture system.

Applications can use the name of the computer on which they are installed or connect to a database and use the name of the computer in the connection string. Because the capture system is not the same as the deployment system, captured applications that require a computer name must virtualize the computer name to ensure that the application can run on any machine.

ThinApp comments out the initial setting of the `VirtualComputerName` parameter. This parameter uses a string that `GetComputerName` and `GetComputerNameEx` API functions return in a virtual application.

Examples

If the capture system does not have the LOCALHOST name, ThinApp comments out the `VirtualComputerName` parameter.

```
;VirtualComputerName=<original_machine_name>
```

If you rename a clean machine as `LOCALHOST` before performing the capture process, the `Package.ini` file activates the `VirtualComputerName` entry. The virtual application works with this `LOCALHOST` name because any computer that the application runs on gets this value as the computer name.

```
VirtualComputerName=LOCALHOST
```

If you type a `GetComputerName` or `GetComputerNameEx` command, the machine returns `LOCALHOST`. If the Windows system requires the `GetComputerName` and `GetComputerNameEx` API functions to operate in a standard way and return the actual name of the computer where the application runs, do not rename the machine as `LOCALHOST`.

Besides specifying a literal string, such as LOCALHOST, you can specify an environment variable.

```
VirtualComputerName=%VCOMPNAME%
```

When you specify an environment variable, the value returned is the value of the environment variable. If the value of the `VirtualComputerName` parameter is %VCOMPNAME%, and the %VCOMPNAME% environment variable is set to `EnvCompName`, the `GetComputerName` API returns `EnvCompName`.

Wow64

The `Wow64` parameter simulates a 32-bit environment for 32-bit applications that cannot run on a 64-bit Windows operating system.

If a 32-bit application tries to handle its own 64-bit registry redirection, you can activate this parameter before building a project. ThinApp comments out the initial setting to prevent Windows on Windows 64-bit (WOW64) emulation.

Examples

You can uncomment the `Wow64` parameter to simulate a 32-bit environment for 32-bit applications on a 64-bit operating system. For example, a virtualized 32-bit Oracle application might not work on a 64-bit operating system.

```
[BuildOptions]
Wow64=0
```

QualityReportingEnabled

The `QualityReportingEnabled` parameter specifies whether VMware can collect anonymous data on a package to improve ThinApp application support. VMware collects application information such as the version and number of application failures.

If you agree to anonymous data collection during the capture process, the `QualityReportingEnabled` parameter uploads data every 10 days.

Examples

You can modify the `QualityReportingEnabled` parameter to turn off ThinApp data collection.

```
[BuildOptions]
QualityReportingEnabled=0
```

Configuring Isolation

You can modify ThinApp parameters to determine the read and write access to the file system and registry keys.

DirectoryIsolationMode

The `DirectoryIsolationMode` parameter specifies the level of read and write access for directories to the physical file system.

The capture process sets the initial value of the `DirectoryIsolationMode` parameter in the `Package.ini` file. This parameter controls the default isolation mode for the files created by the virtual application except when you specify a different isolation mode in the `##Attributes.ini` file for an individual directory.
Any unspecified directories, such as `C:\myfolder`, inherit the isolation mode from the `Package.ini` file.

ThinApp provides only the Merged and WriteCopy isolation mode options in the capture process. You can use the Full isolation mode outside the wizard to secure the virtual environment.

With Merged isolation mode, applications can read and modify elements on the physical file system outside of the virtual package. Some applications rely on reading DLLs and registry information in the local system image. The advantage of using Merged mode is that documents that users save appear on the physical system in the location that users expect, instead of in the sandbox. The disadvantage is that this mode might clutter the system image. An example of the clutter might be first-execution markers by shareware applications written to random computer locations as part of the licensing process.

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

With Full isolation mode, ThinApp blocks visibility to system elements outside the virtual application package. This mode restricts any changes to files or registry keys to the sandbox and ensures that no interaction exists with the environment outside the virtual application package. Full isolation prevents application conflict between the virtual application and applications installed on the physical system. Do not use the Full isolation mode in the `Package.ini` file because that mode blocks the ability to detect and load system DLLs. You can use Full isolation mode as an override mechanism in the `##Attributes.ini` files.

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 delete the sandbox for the change to take effect.

For more information about the definitions and effect of isolation modes, see "Defining Isolation Modes for the Physical File System" on page 18.

Examples

You can modify the `DirectoryIsolationMode` parameter with WriteCopy isolation to ensure that the application can read resources on the local machine but not write to the host computer. This is the default setting for the `snapshot.exe` utility. You must place the parameter under an `[Isolation]` heading.

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

You can assign Merged isolation mode to ensure that the application to read resources on and write to any location on the computer except where the package specifies otherwise. This is the default setting for the Setup Capture wizard.

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

RegistryIsolationMode

The `RegistryIsolationMode` parameter controls the isolation mode for registry keys in the package. This setting applies to the registry keys that do not have explicit settings.

The capture process does not set the value of this parameter. You can configure the registry isolation mode only in the `Package.ini` file. ThinApp sets the initial registry isolation mode to WriteCopy. For information about isolation mode options, see "DirectoryIsolationMode" on page 58.

Do not use the Full isolation mode in the `Package.ini` file because that mode blocks the ability to detect and load system DLLs. You can use Full isolation mode as an override mechanism. You can place exceptions to the configured `RegistryIsolationMode` parameter in the registry key text files in the project directory. An exception might appear in a file, such as `HKEY_CURRENT_USER.txt`, as `isolation_full HKEY_CURRENT_USER\Software\Macromedia`.

All runtime modifications to virtual files in the captured application are stored in the sandbox, regardless of the isolation mode setting. At runtime, virtual and physical registry files are indistinguishable to an application, but virtual registry files always supersede physical registry files when both exist in the same location. If virtual and physical entries exist at the same location, isolation modes do not affect access to these entries because the application always interacts with virtual elements.

If external group policy updates occur separately from the package through the physical registry, you might remove virtual registry files from a package and verify that the parent file of these virtual registry files does not use Full isolation. Because child files inherit isolation modes from parent elements, Full isolation in a parent file can block the visibility of physical child files to an application.

Examples

You can modify the `RegistryIsolationMode` parameter to ensure that the application can read keys from the host computer but not write to the host computer.

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

You can ensure that the application can write to any key on the computer, except where the package specifies otherwise.

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

Configuring File and Protocol Associations

You can modify ThinApp parameters to associate file extensions with applications and to specify protocols that are visible to the physical environment.

FileTypes

The `FileTypes` parameter lists file extensions that the `thinreg.exe` utility associates with an executable file.

The capture process generates initial values that you cannot add to. You can remove extensions that you do not want to associate with the virtual package. Do not use separators between the file extensions in the list.

Examples

A Microsoft Word 2007 package specifies `.doc.docx` as the values of the `FileTypes` parameter. If you capture Microsoft Office 2007 and have Microsoft Office 2003 installed in the physical environment, you can remove the `.doc` extension from the `FileTypes` parameter and leave the `.docx` extension to ensure that Microsoft Word 2003 opens `.doc` files and Microsoft Word 2007 opens `.docx` files.

```
[Microsoft Office Word 2007.exe]
FileTypes=.docx
```

The capture process can create file type associations for `.doc` and `.dot` extensions and link them to Microsoft Word.

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

Protocols

The `Protocols` parameter specifies the protocols, such as HTTP, that are visible to applications in the physical environment. This parameter is similar to the `FileTypes` parameter but deals with applications that handle protocols rather than file types.

The capture process generates initial values that you cannot add to. You can remove entries for browsers or other applications.

Examples

The capture process can specify protocols, such as the `mailto` protocol for a Microsoft Outlook package, in the `Protocols` parameter.

```
[Microsoft Office Outlook 2007.exe]
Protocols=feed;feeds;mailto;Outlook.URL.mailto;stssync;webcal;webcals
```

Configuring Build Output

You can modify ThinApp parameters to specify the location of the build output and the files in the package.

ExcludePattern

The `ExcludePattern` parameter excludes files or directories during the application build process. You must add a `[FileList]` heading before this parameter entry.

You can use a comma to separate patterns in the list. Wildcards (*) match none of the characters or at least one of the characters and question marks (?) match exactly one character. The syntax is similar to the DOS `dir` command, but you can apply wildcard characters to directory names and filenames.

You can specify the `ExcludePattern` parameter in the `Package.ini` file, where the pattern exclusion applies to the entire directory structure, and the `##Attributes.ini` file, where ThinApp adds the pattern exclusion to the current list of exclusions but applies settings only to the specific directory and subdirectories. You can create a different exclusion list for different directories in your project.

Examples

If you store packages in a version control system and you want to exclude version control info from the virtual file system, you can exclude any directories called `.svn` or `.cvs` and all the subdirectories.

```
[FileList]
ExcludePattern=\.svn,\.cvs
```

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

You can exclude any path that ends with `.bak` or `.msi`.

```
[FileList]
ExcludePattern=*.bak,*.msi
```

Icon

The `Icon` parameter specifies the icon file to associate with the generated executable file. This icon appears in the application, such as Microsoft Word, and in the files associated with the application, such as `.doc` files.

Each application comes with its own icon stored as a `.ico` file, within the `.exe` file of the application, or within a `.dll` file. The capture process attaches the icons to the executable files. The application uses the main group icon from the executable file in the `Source` parameter and the individual icon resource that the group icon points to.

Examples

You can modify the `Icon` parameter to use an alternate icon by specifying an executable file that is different from the executable file in the `Source` parameter. Alternate icons might be useful for third party companies.

```
[<my_app>.exe]
Source=%ProgramFilesDir%\<my_app>\app.exe
Icon=%ProgramFilesDir%\<my_app>\app2.exe
```

You can specify an icon set by appending `,1 ,2` to the end of the icon path.

```
[<my_app>.exe]
Source=%ProgramFilesDir%\<my_app>\<app>.exe
Icon=%ProgramFilesDir%\<my_app>\<app2>.exe,1
```

You can use a `.ico` file to specify the application icon.

```
[<my_app>.exe]
Source=%ProgramFilesDir%\<my_app>\<app>.exe
Icon=%ProgramFilesDir%\<my_app>\<my_icon>.ico
```

OutDir

The `OutDir` parameter specifies the directory that stores the `build.bat` output.

Do not modify the value of this parameter.

Examples

The static value of the `OutDir` parameter specifies the `bin` directory of the project.

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

RetainAllIcons

The `RetainAllIcons` parameter keeps all of the original icons of the executable file listed in the `Source` parameter in the application.

The icons that are not assigned to application executable files reside in the virtual file system of the package. The `RetainAllIcons` parameter determines whether to copy the unused icons from the virtual file system to the executable file. To save disk space, ThinApp sets an initial value that removes unused icons from the portion of the executable file that is visible to the physical environment.

Examples

You can modify the `RetainAllIcons` parameter to keep all of the original icons of the application.

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

Configuring Permissions

You can modify ThinApp parameters for security tasks that define user access to packages and change Data Execution Prevention (DEP) protection.

AccessDeniedMsg

The `AccessDeniedMsg` parameter contains an error message to display to users who do not have permission to run a package.

ThinApp sets an initial message that notifies the user to contact the administrator.

Examples

You can modify the `AccessDeniedMsg` parameter to add a technical support number.

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

AddPageExecutePermission

The `AddPageExecutePermission` parameter supports applications that do not work in a Data Execution Prevention (DEP) environment.

The DEP feature of Windows XP SP2, Windows Server 2003, and later operating system versions protects against some security exploits that occur with buffer overflow. This feature creates compatibility issues. Windows turns off the feature by default on Windows XP SP2 and you can use a machine-specific opt-in or opt-out list of the applications to which to apply DEP protection. Opt-in and opt-out policies can be difficult to manage when a large number of machines and applications are involved. The `AddPageExecutePermission` parameter instructs ThinApp to add execution permission to pages that an application allocates. The application can run on machines that have DEP protection enabled without modifying the opt-out list.

ThinApp sets an initial value of the `AddPageExecutePermission` parameter that prevents any change to the DEP protections.

Examples

You can modify the `AddPageExecutePermission` parameter to add execution permission to pages that an application allocates. ThinApp executes code from memory pages that the application specifies. This is useful for applications that combine the program and its data into one area of memory.

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

PermittedGroups

The PermittedGroups parameter restricts a package to a specific set of Active Directory users.

You can specify group names, SID strings, or a mix of group names and SID strings in the same line of the PermittedGroups parameter. If you use a domain-based group name, you must connect to that domain when you build the application package. If you add a SID in the parameter value, you are not required to connect to the domain where the SID is defined.

Active Directory Domain Services define security groups and distribution groups. This parameter can only support nested security groups. For example, if a user is a member of security group A, and security group A is a member of security group B, ThinApp can detect the user as a member of security group A and security group B.

When ThinApp builds an application, ThinApp assumes any specified group names are valid and converts the names to SID values. ThinApp can resolve group ownership at runtime using cached credentials. You can continue to authenticate laptop users even when they are offline. If the user does not have access to run the package, you can customize the AccessDeniedMsg parameter to instruct the user.

You can place the PermittedGroups parameter under the [BuildOptions] heading to affect the package or under the [<application>.exe] heading to affect a specific application. The [<application>.exe] value overrides the default [BuildOptions] value for the specific application.

Examples

You can modify the PermittedGroups parameter to specify a list of Active Directory user group names separated by semicolons. The [BuildOptions] parameters set global settings for the entire project.

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

You can specify a user group setting for a specific application that overwrites the global PermittedGroups setting.

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

If you do not specify a PermittedGroups setting for an application, the application inherits the global PermittedGroups value in the [BuildOptions] section.

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

You can mix group names and SID strings in the same entry for the PermittedGroups parameter.

```
PermittedGroups=S-1-5-32-544;Office Users
```

UACRequestedPrivilegesLevel

The UACRequestedPrivilegesLevel parameter specifies privileges for programs requiring User Account Control (UAC) information. This parameter affects users working on Windows Vista or later operating system versions.

You can use the following values to specify privileges:

- asInvoker

  This value uses the profile in Vista.

- requireAdministrator
- highestAvailable

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

If you do not specify privileges, ThinApp does not assign a default value but operates according to the asInvoker setting.

Examples

You can modify the `UACRequestedPrivilegesLevel` parameter to specify administrator privileges for a program.

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

UACRequestedPrivilegesUIAccess

The `UACRequestedPrivilegesUIAccess` parameter specifies user interface access on Windows Vista or later operating system versions. These operating systems protect some elements of the user interface.

ThinApp assigns an initial value of the `UACRequestedPrivilegesUIAccess` parameter to block application access to protected elements. Although you can assign a `true` or `false` value to the `UACRequestedPrivilegesUIAccess` parameter to specify user interface access, the parameter exists to support Microsoft settings.

Examples

You can keep the initial value of the `UACRequestedPrivilegesUIAccess` parameter to ensure that the virtual application cannot access protected elements.

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

Configuring Objects and DLL Files

You can modify ThinApp parameters to specify COM object access and DLL loading requirements.

ExternalCOMObjects

The `ExternalCOMObjects` parameter determines whether Windows creates and runs COM objects in the physical environment rather than the virtual environment to facilitate application compatibility with ThinApp. COM objects that are external to the virtual environment always run in the physical environment.

ThinApp sets an initial value of the `ExternalCOMObjects` parameter that creates and runs the COM objects in the virtual environment.

COM supports out-of-process executable servers and service-based COM objects. If an application can create COM objects that generate modifications on the host computer, the integrity of the host computer is at risk. If ThinApp runs out-of-process and service-based COM objects in the virtual environment, ThinApp stores in the sandbox all changes that the COM objects make.

The capture process does not generate this parameter. You can add this parameter to the `Package.ini` file.

Examples

When you troubleshoot a problem with VMware support and determine that an application implements COM objects that is not compatible with ThinApp, you can modify the `ExternalCOMObjects` parameter to run the COM objects outside of the virtual environment. You can list the CLSID keys.

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

ExternalDLLs

The `ExternalDLLs` parameter can force Windows to load specific DLL files from the virtual file system.

ThinApp sets an initial value that loads DLL files from the virtual file system and passes the loading process to Windows for DLL files on the physical file system. In some circumstances, Windows must load a DLL file in the virtual file system. You might have a DLL file that inserts itself into other processes using Windows hooks. The DLL file that implements the hook must be available on the host file system and Windows must load that file. When you specify a DLL file in the `ExternalDLLs` parameter, ThinApp extracts the file from the virtual file system to the sandbox and instructs Windows to load it.

Virtual dictation software is a type of software that might interface with native applications that pass information between DLLs. ThinApp can pass the loading of DLLs in the virtual environment to Windows to ensure that local applications can interface with the DLLs.

The `ExternalDLLs` parameter does not support a DLL file that depends on other DLL files inside the virtual file system. In this case, Windows cannot load the DLL file.

Examples

You can modify the `ExternalDLLs` parameter to force Windows to load the `inject.dll` and `injectme2.dll` files from the virtual file system.

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

ForcedVirtualLoadPaths

The `ForcedVirtualLoadPaths` parameter instructs ThinApp to load DLL files as virtual DLL files even if the files reside outside the package. This parameter is useful when the application must load external system DLL files that depend on DLL files located inside the package.

The DLL paths can contain macros. Use semicolons to separate multiple paths.

This parameter achieves the same result as the `AddForcedVirtualLoadPath` API function. See "AddForcedVirtualLoadPath" on page 109.

Examples

You can modify the `ForcedVirtualLoadPaths` parameter when you have an application that depends on external DLL files. When you capture Microsoft Office without Microsoft Outlook and a native version of Microsoft Outlook exists on the local system, you cannot send email from the virtual version of Microsoft Excel because the native `envelope.dll` file that is installed with Microsoft Outlook depends on the `mso.dll` file that ThinApp loads in the virtual environment. You can force ThinApp to load the `envelope.dll` file in the virtual environment instead of in the native environment.

```
[BuildOptions]
ForcedVirtualLoadPaths=%ProgramFilesDir%\Microsoft Office\Office10\envelope.dll
```

IsolatedMemoryObjects

The `IsolatedMemoryObjects` parameter lists the shared memory objects to isolate from other applications or from system objects.

Applications that use `CreateFileMapping` and `OpenFileMapping` Windows functions create shared memory objects. When you do not isolate memory objects, conflicts can occur between virtual applications and native applications sharing those objects. For example, you might have a two versions of an application with one version in the native environment and one version in the virtual environment. When these application versions use information in the same memory object, the applications can interfere with each other and fail. You might want to isolate shared memory objects to ensure that virtual applications and system objects cannot detect each other.

The IsolatedMemoryObjects parameter does not appear in the Package.ini file but you can add the parameter. ThinApp sets an initial value that isolates the memory objects that a native version of Internet Explorer uses in the virtual environment. The value addresses a conflict between the explorer.exe and iexplore.exe utilities when the utilities map sandbox files. You can use the IsolatedMemoryObjects parameter to isolate additional named shared memory objects to ensure that the objects are visible only to other virtual applications using the same sandbox.

The IsolatedMemoryObjects parameter accepts a list of entries that are separated by the semicolon (;). Each entry can contain asterisk (*) and question mark (?) wildcard characters to match variable patterns.

Examples

You can modify the IsolatedMemoryObjects parameter to isolate the memory object with the My Shared Object name and any memory object with outlook in the name.

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

IsolatedSynchronizationObjects

The IsolatedSynchronizationObjects parameter lists the synchronization objects to isolate from other applications.

Synchronization objects coordinate actions between applications. The following Windows synchronization objects might appear in logs for application errors:

- OpenMutex
- CreateMutex
- OpenSemaphore
- CreateSemaphore
- OpenEvent
- CreateEvent

If these objects appear in log files, you might isolate the objects in the virtual environment to avoid a collision with synchronization objects that native applications create. You can isolate synchronization objects from applications that do not run in the same virtual namespace. If two applications share the same sandbox path, the applications have the same namespace for isolated synchronization objects. If two applications have the same sandbox name but different sandbox paths, the applications have separate namespaces.

The IsolatedSynchronizationObjects parameter does not appear in the Package.ini file but you can add the parameter. ThinApp sets an initial value that makes synchronization objects accessible to other applications. Virtual applications with different sandboxes can detect the synchronization objects.

The IsolatedSynchronizationObjects parameter accepts a list of entries that are separated by the semicolon (;). Each entry can use the asterisk (*) and question mark (?) as wildcard characters to match variable patterns.

Examples

You can modify the IsolatedSynchronizationObjects parameter to isolate the synchronization object with the My Shared Object name and the synchronization object with outlook in the name.

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

NotificationDLLs

The NotificationDLLs parameter makes calls to third-party DLL files to provide notification of events, such as application startup or shutdown. The DLLs can reside on the physical file system or the virtual package. If ThinApp cannot load a DLL file, the package generates errors.

This parameter does not appear in the Package.ini file. ThinApp SDK users can add this parameter to the file.

Examples

You can modify the NotificationDLLs parameter to make calls to the First.dll and Second.dll files.

```
[BuildOptions]
NotificationDLLs=First.dll;Second.dll
```

NotificationDLLSignature

The NotificationDLLSignature parameter works with the NotificationDLLs parameter and verifies that a specified DLL file has a signature. If the DLL does not have a signature, ThinApp does not load the file.

Example

You can modify the NotificationDLLSignature parameter with an asterisk (*) to ensure that the DLL file is authenticode-signed.

```
[BuildOptions]
NotificationDLLSignature=*
```

You can set an entity to ensure that the DLL is signed by that entity.

```
[BuildOptions]
NotificationDLLSignature=VMware, Inc.
```

ObjectTypes

The ObjectTypes parameter specifies a list of virtual COM object types that are visible to other applications in the physical environment. You can use scripts, such as VBScripts, to call objects that start captured applications.

An object type is registered to only one native or virtual application at a time. If you install Office 2003 on the native machine and want to use a virtual Office 2007 package, you must determine whether to have the virtual or native application handle the object types.

If you want the virtual Office 2007 to handle the object types, you can leave the ObjectTypes setting in the Package.ini file, build the package, and register it using the thinreg.exe utility. If you want the native Office 2003 to handle the object types, you must remove the ObjectTypes setting from the Package.ini file before building and registering the package. You cannot add random entries to the ObjectTypes parameter. You can only remove entries generated by the capture process.

Examples

If a script or a native application creates an Excel.Application COM object or other COM objects listed in the ObjectTypes parameter, ThinApp starts the virtual package.

```
[Microsoft Office Excel 2007.exe]
ObjectTypes=Excel.Application;Excel.Application.12;Excel.Chart;
               Excel.Macrosheet;Excel.Sheet; Excel.Workspace
```

SandboxCOMObjects

The SandboxCOMObjects parameter indicates whether applications in the physical environment can access COM objects that the virtual application registers at runtime.

ThinApp sets an initial value that prevents native applications in the physical environment from accessing COM objects that the virtual application registers. ThinApp places in the sandbox the COM objects that the virtual application registers.

Examples

You can modify the SandboxCOMObjects parameter to make visible the COM objects that the virtual application registers outside the sandbox. For example, if you install native Microsoft Office 2003 and virtual Microsoft Office 2007, and you run a custom mail merge program in the native environment that starts Microsoft Word and instructs it to open, change, and save a document, you can generate Microsoft Word 2007 documents when virtual Microsoft Word is running. The native application can access COM objects from the virtual application.

```
SandboxCOMObjects=0
```

VirtualizeExternalOutOfProcessCOM

The VirtualizeExternalOutOfProcessCOM parameter controls whether out-of-process COM objects can run in the virtual environment. COM objects that are external to the virtual environment always run in the physical environment.

This parameter addressed out-of-process COM objects that are not part of a ThinApp package and are not registered in the virtual registry. ThinApp sets an initial value of the VirtualizeExternalOutOfProcessCOM parameter to run external out-of-process COM objects in the virtual environment to ensure that COM objects cannot modify the host computer. If a compatibility problem exists with an external COM object running in the virtual environment, you can create and run COM objects on the host system. To run only specific COM objects outside of the virtual environment, you can use the ExternalCOMObjects parameter to list the CLSID of each COM object.

Examples

You can modify the VirtualizeExternalOutOfProcessCOM parameter to run all external out-of-process COM objects in the physical environment rather than the virtual environment. For example, you might use virtual Microsoft Access 2003 to send email through a native IBM Lotus Notes session.

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

Configuring File Storage

You can modify ThinApp parameters to configure file storage and set up virtual drives.

For information about storage related to sandbox configuration, see "Configuring Sandbox Storage and Inventory Names" on page 90.

CachePath

The CachePath parameter sets the deployment system path to a cache directory for font files and stub executable files.

Because of the frequent use of font and stub executable files, ThinApp must extract files in the cache quickly and place them on the physical disk. If you delete the cache, ThinApp can reconstruct the cache.

You can use the THINSTALL_CACHE_DIR environment variable to override the CachePath parameter at runtime. If you do not set the THINSTALL_CACHE_DIR environment variable or the CachePath parameter, ThinApp sets an initial value of the CachePath parameter based on the SandboxPath parameter according to the following rules:

- If the SandboxPath parameter is present in the Package.ini file and uses a relative path, the CachePath parameter uses the sandbox path and stores the cache at the same directory level as the sandbox.
- If the SandboxPath parameter is present in the Package.ini file and uses an absolute path, or if the SandboxPath parameter does not exist in the Package.ini file, the CachePath parameter uses the %Local AppData%\Thinstall\Cache location. This places the cache directory on the local machine, regardless of there the user travels with the sandbox. ThinApp creates a Stubs directory within the cache.

Examples

You can modify the CachePath parameter to use an absolute path.

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

You can set a relative path that ThinApp detects as the path relative to the directory where the application executable file resides. If the package resides in C:\VirtApps and the CachePath parameter has a value of Cache, the cache directory is C:\VirtApps\Cache.

```
CachePath=Cache
```

When you use a USB device and move the sandbox on to the USB device, you might move the cache to the USB device to avoid interfering with the local machine. In this example, the cache and sandbox exist in the same directory level.

```
CachePath=<sandbox_path>
```

UpgradePath

The UpgradePath parameter specifies the location of information and files for Application Sync and integer updates.

ThinApp sets an initial value that causes the Application Sync utility to place log and cache files in the same location as the application executable file on the local machine. Integer updates operate in the same way with files.

When the Application Sync utility downloads an update from a server, it stores the update with a temporary name in the UpgradePath location. The next time the application starts, ThinApp renames the temporary file with a .1 extension or a .2 extension depending on whether .1 already exists. ThinApp attempts to change the name with the .1 extension to the original name of the file that might reside in another directory. If ThinApp cannot make this change, the file keeps the .1 extension in the UpgradePath location. Running the original application accesses that file.

For information about the Application Sync utility, see "Application Sync Updates" on page 43.

Examples

When the default location has limited space, such as a USB device, or you want to isolate upgrades from the application executable file, you can modify the UpgradePath parameter to specify another location to detect update files. The parameter can include environment variables in the path but cannot support folder macros.

```
[BuildOptions]
UpgradePath=C:\Program Files\<my_app_upgrades>
```

VirtualDrives

The VirtualDrives parameter specifies additional drive letters that are available to the application at runtime.

ThinApp makes the virtual environment resemble the physical capture environment and mimics the physical drives that are available on the capture system. ThinApp represents virtual drives through the VirtualDrives parameter and a project folder, such as %drive_<drive_letter>%, that contains the virtual files on the drive. This project folder can reside in the read-only file system of the package and in the sandbox when write operations cannot occur on the physical drive.

The VirtualDrives parameter presents the drives to the application at runtime. The VirtualDrives parameter displays metadata about the drive, such as a the serial number and type of drive. For example, ThinApp detects the physical C: drive on the capture system and enters it into the parameter as a FIXED type of drive with the serial number.

The VirtualDrives parameter includes the following information:

- Drive – Single character between A and Z.
- Serial – 8 digit hex number.

- Type – FIXED, REMOVABLE, CD-ROM, or RAMDISK.
  - FIXED – Indicates fixed media.

    For example, a hard drive or internal Flash drive.
  - REMOVABLE – Indicates removable media.

    For example, a disk drive, thumb drive, or flash card reader.
  - CD-ROM – Indicates a CD-ROM drive.
  - `RAMDISK`—Indicates a RAM disk.

Virtual drives are useful when applications rely on hard-coded paths to drive letters that might not be available on the deployment systems. For example, legacy applications might expect that the `D:` drive is a CD-ROM and that the data files are available at `D:\media`.

Virtual drive settings override the physical properties of the drive on the physical deployment system. If the `VirtualDrives` parameter defines a drive type as CD-ROM, and the physical drive is a hard disk, the application on the deployment system detects that drive as a CD-ROM drive.

Isolation Modes for Virtual Drives

Virtual drives are visible only to applications running in the virtual environment. Virtual drives do not affect the physical Windows environment. Virtual drives inherit isolation modes from the default isolation mode of the project unless you override the mode with a `##Attributes.ini` file in the drive folder within the project directory.

If you copy files to the `%drive_D%` folder before building the application, you can use Full isolation mode for this drive. The application always reads from the virtual drive and does not try to read from any corresponding physical CD-ROM drive on the deployment system.

If you do not copy files to the `%drive_D%` folder before building the application, you can use Merged or WriteCopy isolation modes for virtual drive folders depending on whether you want to read from and write to the physical drive on the deployment system.

If you assign Merged isolation mode to your virtual drive, any write operations to that drive fail if that drive does not exist on the physical deployment system. ThinApp does not direct changes to the sandbox because Merged isolation mode instructs ThinApp to write to the physical drive. When the application cannot write to the physical drive as directed, the write operations fail.

The `VirtualDrives` parameter does not override isolation mode settings. A virtual application might not detect files on a physical drive because of isolation mode settings.

Modify Virtual Drive Isolation Modes

You might modify isolation modes for virtual drives when you want to override the default isolation mode of the project.

To modify virtual drive isolation modes

1 Add the `%Drive_<letter>%` directory to your ThinApp project.

2 Create a `##Attributes.ini` file that includes an isolation mode entry for the drive letter.

```
[Isolation]
DirectoryIsolationMode=<isolation_mode>
```

3 Place the `##Attributes.ini` file in the `%Drive_<letter>%` directory.

Examples

The `VirtualDrives` parameter is a single string that can hold information about multiple drive letters, and optional parameters for those drive letters. The parameter uses semicolons to separate information assigned to different drive letters and commas to separate parameters for individual drive letters. ThinApp assigns a serial number and the `FIXED` type to the drive.

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

You can specify the X, D, and Z virtual drive letters.

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

Drive X is a removable disk with the ff797828 serial number.

Drive D is a CD-ROM drive with an assigned serial number,

Drive Z is a FIXED disk with an assigned serial number.

Configuring Processes and Services

You can modify ThinApp parameters to configure processes and services that might specify write access to a native process or the startup and shutdown of virtual services.

AllowExternalKernelModeServices

The AllowExternalKernelModeServices parameter controls whether applications can create and run native kernel driver services. The service executable file must exist on the physical file system.

ThinApp does not display the default parameter in the Package.ini file but assigns an initial value that prevents the application from starting a native Windows kernel driver service.

Examples

You can add the AllowExternalKernelModeServices parameter to the Package.ini file and modify the default value of 0 to 1 to allow the application to create or open a native Windows kernel driver service.

```
[BuildOptions]
AllowExternalKernelModeServices=1
```

AllowExternalProcessModifications

The AllowExternalProcessModifications parameter determines whether captured applications can write to a native process. Some virtualized applications require a method to interact with native applications.

ThinApp blocks any attempt by the captured application to inject itself into a native application. The captured application can still inject itself into virtual applications running in the same sandbox. ThinApp does not display the default parameter in the Package.ini file.

When ThinApp blocks a captured application from injecting itself into a native application, Log Monitor generates trace logs that refer to the AllowExternalProcessModifications parameter.

Examples

You can add the AllowExternalProcessModifications parameter to the Package.ini file to support write operations from virtual processes to native processes. For example, a speech recognition application must inject itself into native applications to voice the text.

```
[BuildOptions]
AllowExternalProcessModifications=1
```

AllowUnsupportedExternalChildProcesses

The AllowUnsupportedExternalChildProcesses parameter specifies whether to run 64-bit child processes in the physical environment. ThinApp runs 64-bit applications in the physical environment because ThinApp does not support 64-bit processes and cannot virtualize a 64-bit application.

ThinApp sets an initial value of the AllowUnsupportedExternalChildProcesses parameter that runs 64-bit applications in the physical environment. You can run 64-bit child process tasks on applications that run on 64-bit systems. Running the print spooler is an example of a 64-bit child process task.

Examples

If you want to protect the physical file system from any changes, you can modify the `AllowUnsupportedExternalChildProcesses` parameter and block ThinApp from generating 64-bit child processes outside of the virtual environment. ThinApp cannot run any 64-bit processes because ThinApp does not support the processes in the virtual environment.

```
[BuildOptions]
AllowUnsupportedExternalChildProcesses=0
```

AutoShutdownServices

The `AutoShutdownServices` parameter controls whether to shut down virtual services when the last nonservice process exits.

ThinApp sets an initial value to stop virtual services when the last nonservice process exits. The parameter does not affect services outside the virtual context.

Examples

You can modify the `AutoShutdownServices` parameter when you run Apache Web Server and want to keep the virtual service running after the application that starts the service exits.

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

AutoStartServices

The `AutoStartServices` parameter controls whether to start the virtual services when the first virtual application starts.

ThinApp sets an initial value that starts the virtual services that are installed with the startup type of Automatic. The virtual services start when the user runs the first parent process.

Examples

When applications install a service but do not use it, you can modify the `AutoStartServices` parameter to prevent the start of the virtual service and save time.

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

ChildProcessEnvironmentDefault

The `ChildProcessEnvironmentDefault` parameter determines whether ThinApp runs all child processes in the virtual environment.

ThinApp creates all child processes in the virtual environment. If the processes are slow, you might want to move child processes to the physical environment. As a child process, Microsoft Outlook might affect performance when it copies the whole mailbox to the virtual environment.

You can create specific exceptions with the `ChildProcessEnvironmentExceptions` parameter.
See "ChildProcessEnvironmentExceptions" on page 73.

Examples

If you do not want the child process to operate in or slow down the virtual environment, you can modify the `ChildProcessEnvironmentDefault` parameter to create child processes in the physical environment.

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

ChildProcessEnvironmentExceptions

The `ChildProcessEnvironmentExceptions` parameter notes exceptions to the `ChildProcessEnvironmentDefault` parameter when you want to specify child processes.

When you set the `ChildProcessEnvironmentDefault` parameter to `Virtual`, the `ChildProcessEnvironmentExceptions` parameter lists the applications that run outside of the virtual environment. When you set the `ChildProcessEnvironmentDefault` parameter to `External`, the `ChildProcessEnvironmentExceptions` parameter lists the applications that run inside the virtual environment.

Examples

You can specify exceptions to running child processes in the virtual environment. When the virtual application starts a `notepad.exe` child process, the child process runs outside the virtual environment.

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

Configuring Sizes

You can modify ThinApp parameters to compress file and block sizes for applications.

BlockSize

The `BlockSize` parameter controls the size of compression blocks only when ThinApp compresses files for a build.

A larger block size can achieve higher compression. Larger block sizes might slow the performance because of the following reasons:

- The build process slows down with larger block sizes.
- The startup time and read operations for applications slow down with large block sizes.
- More memory is required at runtime when you use larger block sizes.

You can specify the `BlockSize` parameter in the `Package.ini` file and in the `##Attributes.ini` file. You can use different block sizes for different directories within a single project.

Examples

You can increase the default size of 64KB in the `BlockSize` parameter to a supported block size of 128KB, 256KB, 512KB, or 1MB. You can add `k` after the number to indicate kilobytes or `m` to indicate megabytes.

```
[Compression]
BlockSize=128k
```

CompressionType

The `CompressionType` parameter can compress all files in a package except for Portable Executable files.

You can compress files when you have a large package and disk space is a top priority. Compression has a quick rate of decompression and little effect on most application startup times and memory consumption at runtime. Compression achieves similar compression ratios as the ZIP algorithm.

Table 5-1 lists sample compression ratios and startup times for a Microsoft Office 2003 package that runs from a local hard drive.

Table 5-1. Sample Compression Ratios and Startup Times

Compression Type	None	Fast
Size	448,616KB	257,373KB
Compression ratio	100%	57%
Startup time (first run)	6 seconds	6 seconds
Startup time (second run)	0.1 seconds	1 seconds
Build time (first build)	3 minutes	19 minutes
Build time (second build)	2 minutes	1.2 minutes

Compression has some performance consequences and can affect the startup time on older computers or in circumstances where you start the application multiple times and depend on the Windows disk cache to provide data for each start.

The `CompressionType` parameter does not affect MSI files. For information about compressing MSI files, see "MSICompressionType" on page 74.

Examples

ThinApp sets default values for the `OptimizeFor` parameter and the `CompressionType` parameter that work together to achieve maximum memory performance and startup time. ThinApp stores all data in uncompressed format.

```
[Compression]
CompressionType=None

[BuildOptions]
OptimizeFor=Memory
```

You can use this configuration when disk space is moderately important. Thinapp stores executable files in uncompressed format but compresses all the other data.

```
[Compression]
CompressionType=Fast

[BuildOptions]
OptimizeFor=Memory
```

You can use this configuration when disk space is the top priority. ThinApp compresses all files.

```
[Compression]
CompressionType=Fast

[BuildOptions]
OptimizeFor=Disk
```

MSICompressionType

The `MSICompressionType` parameter determines whether to compress MSI files for package distribution. Compression improves performance when opening MSI files and using the ThinApp SDK.

If you create an MSI file during the capture process, ThinApp adds the `MSICompressionType` parameter to the `Package.ini` file and sets the initial value of `Fast` to compress the file. Decompression occurs at the time of installation.

You do not have to set the `MSICompressionType` parameter to `Fast` when you set the `CompressionType` parameter to `Fast`. Setting both parameters does not increase the amount of compression.

Examples

If you are working with large builds and performance is not a priority, you can modify the `MSICompressionType` parameter to prevent MSI file compression.

```
[Compression]
MSICompressionType=none
```

OptimizeFor

The `OptimizeFor` parameter controls whether to compress executable files or to reduce memory consumption and page file usage on the hard drive to improve startup performance. You can use this parameter with the `CompressionType` parameter to customize the package size, memory allocation, and application startup time.

VMware recommends leaving the default setting of the `OptimizeFor` parameter and the `CompressionType` parameter to improve startup performance and memory consumption. You can change the parameters to favor smaller package sizes when disk size is the priority concern. ThinApp compresses executable files only when you set the `OptimizeFor` parameter to `Disk` and the `CompressionType` parameter is `Fast`. Executable files stored in compressed format inside a package can adversely affect performance and memory consumption. When ThinApp loads executable files from compressed format, ThinApp cannot share the file memory across similar suites of applications or with other users in a multiuser environment such as Terminal Server.

When you want to compress all the package files except for Portable Executable files, you can leave the default `OptimizeFor` parameter and only set the `CompressionType` parameter is `Fast`. When you want to compress only MSI files, use the `MSICompressionType` parameter.

Examples

VMware recommends the default configuration of the `OptimizeFor` parameter and the `CompressionType` parameter to maximize performance. ThinApp stores all data in uncompressed format.

```
[Compression]
CompressionType=None

[BuildOptions]
OptimizeFor=Memory
```

VMware recommends this configuration when disk space is moderately important. Thinapp stores executable files in uncompressed format but compresses all the other data.

```
[Compression]
CompressionType=Fast

[BuildOptions]
OptimizeFor=Memory
```

VMware recommends this configuration when disk space is the top priority. ThinApp compresses all files.

```
[Compression]
CompressionType=Fast

[BuildOptions]
OptimizeFor=Disk
```

Configuring Logging

You can modify ThinApp parameters to prevent logging activity or customize the location of the log files.

DisableTracing

The `DisableTracing` parameter prevents `.trace` file generation when you run Log Monitor for security and resource purposes.

You might block standard `.trace` file generation to hide the application history from a user. In a testing environment, you might turn off tracing for applications that you know work properly. Producing extra `.trace` files wastes disk space and CPU time.

Examples

You can set the `DisableTracing` parameter to prevent the generation of `.trace` files in Log Monitor.

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

LogPath

The `LogPath` parameter sets the location to store `.trace` files during logging activity.

The default location is the same directory that stores the application executable file. You might change the default location to find a directory with more space or to redirect the logs from a USB device to the client computer. Unlike most paths in ThinApp, the log path cannot contain macros such as `%AppData%` or `%Temp%`.

Examples

You can set the `LogPath` parameter to store log files in `C:\ThinappLogs`.

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

Configuring Versions

ThinApp parameters provide information about the versions of application executable files and ThinApp.

CapturedUsingVersion

The `CapturedUsingVersion` parameter displays the version of ThinApp for the capture process and determines the file system macros that ThinApp must expand.

Do not modify or delete this parameter from the `Package.ini` file. ThinApp uses this parameter for backward compatibility and technical support.

Examples

The `CapturedUsingVersion` parameter might display ThinApp version 4.0.0-2200.

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

StripVersionInfo

The `StripVersionInfo` parameter determines whether to remove all version information from the source executable file when ThinApp builds the application. The source executable file is the file listed in the `Source` parameter.

Version information for executable files appears in Windows properties. Properties information includes the copyright, trademark, and version number. The `StripVersionInfo` parameter can remove the **Version** tab of Windows properties.

ThinApp sets an initial value of the `StripVersionInfo` parameter that copies all version information from the source executable file.

Examples

In rare cases, you can modify the `StripVersionInfo` parameter to generate an application without version information. For example, you might want to circumvent version detection scans that compare versions against a database of outdated software.

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

Version.XXXX

The `Version.XXXX` parameter overrides application version strings or adds new version strings in the **Version** tab of Windows properties.

The capture process does not generate this parameter. You can add this parameter to the `Package.ini` file.

Examples

You can set a new product name with the `Version.XXXX` parameter. You might want `ThinApp Office` rather than `Office` as the product name. Use the `Version.<string_name>=<string_value>` format.

```
[<app>.exe]
Version.ProductName=ThinApp Office
Version.Description=This Product is great!
```

Configuring Locales

You can use ThinApp parameters to verify locale information.

AnsiCodePage

The `AnsiCodePage` parameter displays a numerical value that represents the language of the operating system on which you capture the application. ThinApp uses the value to manage multibyte strings.

This parameter does not perform language translation. The value that affects the display of text strings and use of the strings within the application.

Examples

When the operating systems of the deploy and capture computers have different languages, you might check the `AnsiCodePage` parameter.

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

LocaleIdentifier

The `LocaleIdentifier` parameter displays a numeric ID for the locale that affects layout and formatting. The value locates the correct language resources from the application.

ThinApp runs packages according to the regional and language settings of the capture system rather than the settings of the system that runs packages. If you capture an application that requires a locale format, such as a date format, on a system that does not have the required format, you can comment out this parameter to ensure that the application can run on a system that has the supported format.

Examples

When the regional language of the operating system is U.S. English, the capture process sets the `LocaleIdentifier` parameter to 1033.

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

LocaleName

The `LocaleName` parameter displays the name of the locale when you capture an application on Microsoft Vista.

Examples

The LocaleName parameter can display a Japanese locale name.

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

Configuring Individual Applications

You can modify ThinApp parameters to configure specific applications.

Parameters specific to entry points fall under the [<application>.exe] sections of the Package.ini file. For example, the entries under [Adobe Reader 8.exe] for an Adobe Reader application might affect command-line arguments and application shortcuts.

CommandLine

The CommandLine parameter specifies the command-line arguments that start a shortcut executable file. While the Source parameter specifies the path to the shortcut executable file, the CommandLine parameter specifies the file with the required options or parameters to start it.

If the **Start** menu shortcut for the application has command-line options, the capture process sets the initial value of the CommandLine parameter based on those options. In rare troubleshooting cases with technical support, you might alter this parameter.

The options and parameters follow the base application name. Depending on the application, use / or – before the option or parameter. Use folder macros for the path name conventions.

Examples

You can modify the CommandLine parameter with an entry based on the "C:\Program Files\Mozilla Firefox\firefox.exe" –safe-mode **Start** menu shortcut.

```
CommandLine="C:\Program Files\Mozilla Firefox\firefox.exe" –safe-mode
```

Command-line arguments can use the /<option> <parameter> format.

```
[<app>.exe]
Source=%ProgramFilesDir%\<base_app>\<app>.exe
Shortcut=<primary_data_container>.exe
CommandLine="%ProgramFilesDir%\<base_app>\<app>.exe" /<option> <parameter>
```

Disabled

The Disabled parameter determines whether the application build target is just a placeholder and prevents ThinApp from generating the executable file in the /bin directory.

ThinApp enables entry points when the application installer has shortcuts on the desktop and in the **Star**t menu. When you do not select an entry point that appears in the Setup Capture wizard, ThinApp sets an initial value of the Disabled parameter that prevents the generation of that application executable file during the build process.

Examples

If you do not select the cmd.exe, regedit.exe, or iexplore.exe troubleshooting entry points during the capture process, and you develop a need to debug the environment, you can modify the Disabled parameter to generate these entry points.

```
[app.exe]
Source=%ProgramFilesDir%\<my_app>\<app>.exe
Disabled=0
```

ReadOnlyData

The ReadOnlyData parameter specifies the name of the read-only virtual registry file created during the application build and designates the primary data container for an application.

Do not modify this parameter. The Package.ini file displays this parameter in case you want to locate the primary data container.

When the primary data container is less than 200MB, ThinApp stores the container within an entry point executable file. When the primary data container is more than 200MB, ThinApp stores the container as a .dat file that cannot serve as an entry point for the application.

Examples

ThinApp sets the required value of the ReadOnlyData parameter that specifies Package.ro.tvr as the name of the virtual registry file.

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

ReserveExtraAddressSpace

The ReserveExtraAddressSpace parameter indicates the amount of extra address space to reserve for the captured executable file.

ThinApp uses the executable file listed in the Source parameter to determine the amount of memory address space to allocate for an application. When you build a package that includes a source executable file that is not included in the package, such as a cmd.exe entry point, or you activate automatic updates for an application that directs the new version of the application to the sandbox, ThinApp adjusts to the need for more memory but does not reserve a set amount of extra address space. In rare cases that might involve technical support, you can use the ReserveExtraAddressSpace parameter to add memory space.

Examples

You can instruct the Windows loader to reserve extra address space. Add K after the number to indicate kilobytes or M to indicate megabytes.

```
[<app>.exe]
Source=%ProgramFilesDir%\<my_app>\<app>.exe
ReserveExtraAddressSpace=512K
```

Shortcut

The Shortcut parameter points a shortcut executable file to the primary data container that contains the virtual file system and virtual registry. You can distinguish a primary data container from other entry points in the Package.ini file because the primary data container contains the ReadOnlyData entry and the other entry points contain the Shortcut entry.

To ensure that the application can start, the shortcut executable file must reside in the directory that stores the primary data container file. For information about the primary data container, see "ReadOnlyData" on page 79.

Do not modify the value of the Shortcut parameter. ThinApp detects the primary data container during the capture process.

Examples

ThinApp can point AcroRd32.exe, the shortcut executable file, to Adobe Reader 8.exe, the primary data container file.

```
[AcroRd32.exe]
Shortcut=Adobe Reader 8.exe
Source=%ProgramFilesDir%\Adobe\Reader 8.0\Reader\AcroRd32.exe
```

ThinApp can point Microsoft Office Word 2007.exe, the shortcut executable file, to Microsoft Office Enterprise 2007.dat, the primary data container file.

```
[Microsoft Office Word 2007.exe]
Source=%ProgramFilesDir%\Microsoft Office\Office12\WINWORD.EXE
Shortcut=Microsoft Office Enterprise 2007.dat
```

Shortcuts

The `Shortcuts` parameter lists the locations where the `thinreg.exe` utility creates a shortcut to a virtual application.

The capture process determines `Shortcuts` entries based on the shortcuts the application installer implements. MSI files use the `Shortcuts` parameter to determine the shortcuts to create.

Examples

You can modify the `Shortcuts` parameter to create a shortcut in the Microsoft Office folder of the **Start** menu to the Microsoft Word 2003 application. If you add shortcut locations, use semicolons to separate the entries. Each entry can contain folder macros.

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

Source

The `Source` parameter specifies the executable file that ThinApp loads when you use a shortcut executable file. The parameter provides the path to the executable file in the virtual or physical file system.

ThinApp specifies the source for each executable file. If an application suite has three user entry points, such as `Winword.exe`, `Powerpnt.exe`, and `Excel.exe`, the `Package.ini` file lists three application entries. Each entry has a unique source entry.

If ThinApp cannot locate the source executable file in the virtual file system, ThinApp searches the physical file system. For example, if you use native Internet Explorer from the virtual environment, ThinApp loads the source executable file from the physical file system.

The `Source` parameter and the `/bin` directory in the project are not related to each other. The `/bin` directory stores the generated executable file and the `Source` path points to the installed executable file stored in the read-only virtual file system.

Do not modify the `Source` path. The capture process determines the path based on where the application installer places the executable file in the physical file system of the capture machine. ThinApp creates a virtual file system path based on the physical file system path.

Examples

The `Source` parameter can point to an entry point in `C:\Program Files\<base_app>\<app>.exe`.

```
[<app>.exe]
Source=%ProgramFilesDir%\<base_app>\<app>.exe
```

WorkingDirectory

The `WorkingDirectory` parameter determines the first location in which an application looks for files and places files.

ThinApp does not include this parameter by default in the `Package.ini` file because Thinapp assumes the working directory is the directory where the executable file resides. The typical location in a ThinApp environment is on the desktop of the deployment machine.

You can set the working directory for individual applications. The working directory can exist in the virtual file system, the sandbox, or the physical system depending on the isolation mode setting. You can use folder macros for the path name conventions.

The `WorkingDirectory` parameter sets the initial value of the working directory but the directory is dynamic as you navigate to other locations.

Examples

If you have an application on a USB drive, you can modify the `WorkingDirectory` value from the default USB location to the `My Documents` directory on the desktop.

```
[<app>.exe]
WorkingDirectory=%Personal%
```

The location of the `My Documents` directory depends on the isolation mode setting. If you want to map the working directory to the `My Documents` directory on the physical system, use the Merged isolation mode setting. If you want to map the working directory to the sandbox on the local machine, use the WriteCopy or Full isolation mode setting.

Configuring Dependent Applications with Application Link

The Application Link utility keeps shared components or dependent applications in separate packages. In the `Package.ini` file, you can use the `OptionalAppLinks` and `RequiredAppLinks` entries to dynamically combine ThinApp packages at runtime on end-user computers. This process enables you to package, deploy, and update component pieces separately and retain the benefits of application virtualization.

ThinApp can link up to 250 packages at a time. Each package can be any size and the links must point to the primary data container of a package.

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

For more information about the Application Link utility, see "Application Link Updates" on page 46, "OptionalAppLinks" on page 83, and "RequiredAppLinks" on page 82.

Application Link Path Name Formats

The Application Link utility supports the following path name formats:

- Path names can be relative to the base executable file. For example, `RequiredAppLinks=..\SomeDirectory` results in `C:\MyDir\SomeDirectory` when you deploy the base executable file to `c:\MyDir\SubDir\ Dependency.exe`.
- Path names can be absolute path names. An example is `RequiredAppLinks=C:\SomeDirectory`.
- Path names can use a network share or a UNC path. An example is `RequiredAppLinks=\\share\somedir\Dependency.exe`.
- Path names can contain environment variables and dynamically expand to any of the preceding path names. An example is `RequiredAppLinks=%MyEnvironmentVariable%\Package.dat`.

  The risk of using environment variables is that a user might change the values before starting the application and create an Application Link dependency other than the one that the administrator set up.
- Path names can contain ThinApp folder macros. An example is `RequiredAppLinks=%SystemSystem%\Package.dat`.
- Path names can specify multiple links or dependencies with a semicolon that separates individual filenames. An example is `RequiredAppLinks=Dependency1.exe; Dependency2.exe;`.

RequiredAppLinks

The `RequiredAppLinks` parameter specifies a list of required packages to import to the base package at runtime. You can configure this parameter in the `Package.ini` file of the base package.

If the import operation for any dependent package fails, an error message appears and the base executable file exits. You can use the `OptionalAppLinks` parameter instead to continue even when load errors occur. If you use a wildcard pattern to specify a package and files do not match the wildcard pattern, ThinApp does not generate an error message.

Importing packages involves the following operations:

- Running VBScripts from imported packages
- Starting autostart services from imported packages
- Registering fonts from imported packages
- Relocating SxS DLL files from Windows XP to Windows Vista

You must create a link to the primary data container of a package. You cannot link to other shortcut packages.

Path names are on the deployment machine because the linking takes effect at runtime on the client machine. Use semicolons to separate the linked packages. For information about path name formats, see "Application Link Path Name Formats" on page 81.

Examples

If you package the .NET framework in the `dotnet.exe` package and you have a .NET application, you can specify that the application needs to link to the `dotnet.exe` file before it can start.

```
RequiredAppLinks=C:\abs\path\dotnet.exe
```

You can specify a relative path.

```
RequiredAppLinks=<relative_path>\dotnet.exe
```

You can specify a UNC path.

```
RequiredAppLinks=\\server\share\dotnet.exe
```

You can use ThinApp folder macros in the path value.

```
RequiredAppLinks=%SystemSystem%\Package.dat
```

You can use environment variables in the path value. The risk of using environment variables is that a user might change the values before starting the application and create an Application Link dependency other than the one that the administrator set up.

```
RequiredAppLinks=%MyEnvironmentVariable%\Package.dat
```

You can import a single package located in the same directory as the base executable file.

```
RequiredAppLinks=Plugin.exe
```

You can import a single package located in a subdirectory of the base executable file.

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

You can import all executable files located in the directory for plug-in files. If ThinApp cannot import any executable file because the file is not a valid Thinapp package or because a security problem exists, the base executable file fails to load.

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

You can import all executable files located at the `n:\plugins` absolute path.

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

You can expand the PLUGINS environment variable and import all executable files at this location.

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

You can load two specified plug-in files and a list of executable files located under the plug-in location.

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

OptionalAppLinks

The OptionalAppLinks parameter is similar to the RequireAppLinks parameter but ignores errors and starts the main application even when an import operation fails.

You must create a link to the primary data container of a package. You cannot link to other shortcut packages.

Path names are on the deployment machine because the linking takes effect at runtime on the client machine. You can specify absolute paths, such as C:\abs\path\dotnet.exe, relative paths, such as relpath\dotnet.exe, and UNC paths, such as \\server\share\dotnet.exe.

RequiredAppLinks and OptionalAppLinks parameters use the same syntax. For information about the RequireAppLinks parameter and examples, see "RequiredAppLinks" on page 82.

Configuring Application Updates with Application Sync

The Application Sync utility keeps deployed virtual applications up to date. When an application starts, Application Sync can query a Web server to determine if an updated version of the package is available. If an update is available, ThinApp downloads the differences between the existing package and the new package and constructs an updated version of the package.

The Application Sync utility downloads updates in the background. You can continue to use an old version of the application. If the user quits the application before the download is complete, the download resumes when the virtual application starts again. When the download is finished, ThinApp activates the new version the next time the application starts.

You must uncomment the AppSyncURL parameter to activate all Application Sync parameters. The following entries are the default settings for Application Sync parameters:

```
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
```

AppSyncClearSandboxOnUpdate

The AppSyncClearSandboxOnUpdate parameter determines whether to clear the sandbox after an update.

ThinApp sets an initial value of the AppSyncClearSandboxOnUpdate parameter that keeps the contents of the sandbox.

Examples

You can modify the AppSyncClearSandboxOnUpdate parameter to clear the sandbox after application updates.

```
AppSyncClearSandboxOnUpdate=1
```

AppSyncExpireMessage

The AppSyncExpireMessage parameter sets the message that appears when the connection to the Web server fails after the expiration period ends and a virtual application starts. The application quits when the message appears.

Examples

ThinApp provides a default message for the AppSyncExpireMessage parameter.

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

If the value of the AppSyncExpirePeriod parameter is in hours or minutes, change the message to indicate hours or minutes rather than days.

AppSyncExpirePeriod

The AppSyncExpirePeriod parameter sets the expiration of the package in minutes (m), hours (h), or days (d). If ThinApp cannot reach the Web server to check for updates, the package continues to work until the expiration period ends and the user closes it. Even after the expiration period ends, ThinApp tries to reach the Web server at each subsequent startup attempt.

Examples

You can prevent the package from expiring with the default never value.

```
AppSyncExpirePeriod=never
```

AppSyncURL

The AppSyncURL parameter sets the Web server URL or fileshare location that stores the updated version of an application. ThinApp checks this location and downloads the updated package.

Application Sync works over the HTTP (unsecure), HTTPS (secure), and File protocols. Part of the HTTPS protocol involves checking the identity of the Web server. You can include a user name and a password in the AppSyncURL parameter for basic authentication. ThinApp adheres to the standard Internet Explorer proxy setting.

You must uncomment the AppSyncURL parameter to activate all Application Sync parameters.

Examples

You can assign an HTTP or HTTPS value to the AppSyncURL parameter according to the following format.

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

You can specify local and network drive paths.

```
file:///C:/<path>/<package_name>.exe
```

You can use a UNC path and access locations of network resources.

```
file://<server>/<share>/<path>/<package_name>.exe
```

AppSyncUpdateFrequency

The AppSyncUpdateFrequency parameter specifies how often ThinApp checks the Web server for application updates. You can set the update frequency in minutes (m), hours (h), or days (d).

ThinApp sets an initial value of 1d that connects a package to the Web server once a day to check for updates. ThinApp does not check for an update when another running application shares the same sandbox.

Examples

You can modify the AppSyncUpdateFrequency parameter with a value of 0 to set the application to check for updates every time you start it.

```
AppSyncUpdateFrequency=0
```

AppSyncUpdatedMessage

The AppSyncUpdatedMessage parameter sets the message that appears when an updated package first starts.

Examples

You can use the AppSyncUpdatedMessage parameter to confirm that the application is updated.

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

AppSyncWarningFrequency

The AppSyncWarningFrequency parameter specifies how often a warning appears before the package expires. You can specify minutes (m), hours (h), or days (d).

ThinApp sets an initial value of 1d that sets the warning message to appear once a day.

Examples

You can modify the AppSyncWarningFrequency parameter to configure the warning to appear each time the application starts.

```
AppSyncWarningFrequency=0
```

AppSyncWarningMessage

The AppSyncWarningMessage parameter sets the message that appears when the warning period starts. The first time you start the application in the warning period, a warning message appears and ThinApp tries to access the update from the server. If ThinApp cannot update the package, ThinApp tries again every time the application starts. The warning message appears only after each AppSyncWarningFrequency period expires.

Examples

ThinApp includes a default message for the Application Sync utility warning.

```
AppSyncWarningMessage=This application will become unavailable for use in %%remaining_days%%
day(s) if it cannot contact its update server. Check your network connection to ensure
uninterrupted service.
```

The %%remaining_days%% variable is the number of days remaining until the expiration of the package.

If the value of the AppSyncWarningPeriod parameter is in hours or minutes, change the message to indicate hours or minutes rather than days.

AppSyncWarningPeriod

The AppSyncWarningPeriod parameter sets the start of the warning period before a package expires. You can specify minutes (m), hours (h), or days (d). When the warning period starts, ThinApp checks the Web server every time an application starts and sets the value of the AppSyncUpdateFrequency parameter to 0.

Examples

The default period of the `AppSyncWarningPeriod` parameter is five days.

```
AppSyncWarningPeriod=5d
```

Configuring MSI Files

You can modify ThinApp parameters to configure MSI files for deployment through desktop management systems.

For information about working with MSI files, see "Building an MSI Database" on page 33.

Information about compression of MSI files appears with other parameters that control file sizes. See "MSICompressionType" on page 74.

MSIArpProductIcon

The `MSIArpProductIcon` parameter specifies the icons to represent the application in the Windows **Add or Remove Programs** dialog box. The icon can reside in ICO, DLL, or executable files.

Do not modify this parameter. If an MSI package does not have an application icon, the application appears with a generic icon.

Examples

The `MSIArpProductIcon` parameter can specify an icon for Microsoft Office 2007. This example uses an index number to point to the first icon inside a DLL file.

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

The `<icon_index_number>` entry in this `MSIArpProductIcon=<path_to_icon_file>[,<icon_index_number>]` format is applicable only when multiple icons are available in a DLL file or executable file.

MSIDefaultInstallAllUsers

The `MSIDefaultInstallAllUsers` parameter sets the installation mode of the MSI database. You can install a `.msi` file for all users on a computer and for individual users.

For information about forcing an MSI installation for each user or each machine, see "Force MSI Deployments for Each User or Each Machine" on page 34.

The parameter works only when the `MSIFilename` parameter requests the generation of a Windows Installer database.

Examples

ThinApp sets an initial value for the `MSIDefaultInstallAllUsers` parameter that installs the MSI database with shortcuts and file type associations for all users who log in to the computer. The user who installs the database must have administrator rights. You can use this approach to push the application to desktops for all users.

```
[BuildOptions]
MSIFilename=<my_msi>.msi
MSIDefaultInstallAllUsers=1
```

An individual user can install the MSI database with shortcuts and file type associations for only that user. You do not need administrator rights for an individual user installation. Use this approach when you want each user to deploy the application separately.

```
[BuildOptions]
MSIFilename=<my_msi>.msi
MSIDefaultInstallAllUsers=0
```

An administrator can install the MSI database for all users on a machine or an individual user without administrator rights can install the database for only that user.

```
[BuildOptions]
MSIFilename=<my_msi>.msi
MSIDefaultInstallAllUsers=2
```

MSIFilename

The `MSIFilename` parameter triggers the generation of an MSI database and specifies its filename. Other MSI parameters can work only when you uncomment the `MSIFilename` parameter.

This parameter produces a Windows Installer with the specified filename in the output directory. You can create an MSI file when you want to deliver packages to remote locations through desktop management systems. Unlike executable files that require the manual use of the `thinreg.exe` utility, MSI files automate the creation of shortcuts and file type associations for each user.

ThinApp comments out the `MSIFilename` parameter unless you specify MSI generation during the capture process.

Examples

The inventory name is the default name in the `MSIFilename` parameter.

```
[BuildOptions]
;MSIFilename=<inventory_name>.msi
```

You can generate an MSI file during the build process and replace the filename with your own filename.

```
[BuildOptions]
MSIFilename=<my_msi>.msi
```

MSIInstallDirectory

The `MSIInstallDirectory` parameter specifies the relative path of the MSI installation directory. The path is relative to `%ProgramFilesDir%` for installations on each machine and relative to `%AppData%` for installations for each user.

When you install the MSI database for all users, ThinApp places applications in the `C:\%ProgramFilesDir%\<InventoryName> (VMware ThinApp)` directory during the installation on each machine.

When you install the MSI database for individual users, ThinApp places applications in the `C:\%AppData%\<InventoryName> (VMware ThinApp)` directory.

The parameter works only when the `MSIFilename` parameter requests the generation of a Windows Installer database.

Examples

If you do not want the `MSIInstallDirectory` parameter to use a location based on the inventory name, you can install a `.msi` file in the `C:\Program Files\<my_application>` directory.

```
[BuildOptions]
MSIFilename=<my_msi>.msi
MSIInstallDirectory=<my_application>
```

MSIManufacturer

The `MSIManufacturer` parameter specifies the manufacturer or packaging company of the MSI database and displays the value in the Windows **Add or Remove Programs** dialog box.

ThinApp sets the initial value of the `MSIManufacturer` parameter to the name of the company that your copy of Windows is registered to.

The parameter works only when the `MSIFilename` parameter requests the generation of a Windows Installer database.

Examples

You can modify the `MSIManufacturer` parameter to display the name of a specific department. For example, users can see a department name in the Windows **Add or Remove Programs** dialog box and contact the help desk for that department.

```
[BuildOptions]
MSIFilename=<my_msi>.msi
MSIManufacturer=<department_or_company_name>
```

MSIProductCode

The `MSIProductCode` parameter specifies a product code for the MSI database. Windows Installer uses the code to identify MSI packages.

The capture process generates a random and unique product code that is not taken from the application. The value must be a valid Globally Unique Identifier (GUID).

The parameter works only when the `MSIFilename` parameter requests the generation of a Windows Installer database.

Do not modify the `MSIProductCode` parameter.

Examples

The capture process can create an `MSI` file with 590810CE-65E6-3E0B-08EF-9CCF8AE20D0E as the product code.

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

MSIProductVersion

The `MSIProductVersion` parameter specifies a product version number for the MSI database to facilitate version control. This version number is unrelated to the application version or the ThinApp version.

ThinApp assigns an initial version of 1.0. This version appears in 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 blocks the installation of an older version over an updated version. In this situation, you must uninstall the new version.

The `MSIProductVersion` parameter works only when the `MSIFilename` parameter requests the generation of a Windows Installer database.

Examples

You can change the value of the `MSIProductVersion` parameter when you change the MSI package. A value of 2.0 causes ThinApp to uninstall a 1.0 version of the package and install the 2.0 version of the package.

```
[BuildOptions]
MSIFilename=<my_msi>.msi
MSIProductVersion=2.0
```

The format of the `MSIProductVersion` value is `X.Y.Z`. The values of `X` and `Y` range from 0 to 255, and the value of `Z` ranges from 0 to 65536.

MSIRequireElevatedPrivileges

The `MSIRequireElevatedPrivileges` parameter applies to Windows Vista and specifies elevated privilege requirements for the MSI database.

Most users who log in to Windows Vista have restricted privileges. To install MSI packages for all users who must have shortcuts and file type associations, the users must have elevated privileges.

ThinApp sets an initial value of the `MSIRequireElevatedPrivileges` parameter that marks the MSI database as requiring elevated privileges. If your system is set up for UAC prompts, a UAC prompt appears when you install an application.

The parameter works only when the `MSIFilename` parameter requests the generation of a Windows Installer database.

Examples

You can modify the `MSIRequireElevatedPrivileges` parameter to block the UAC prompt and the installation across all computers.

```
[BuildOptions]
MSIFilename=<my_msi>.msi
MSIRequireElevatedPrivileges=0
```

MSIUpgradeCode

The `MSIUpgradeCode` parameter specifies a code for the MSI database that facilitates updates. When two packages, such as the version 1.0 package and the version 2.0 package, have the same upgrade code, the MSI installer detects this link, uninstalls the earlier package, and installs the updated package.

The capture process generates a random upgrade code based on the inventory name. To ensure that the MSI database versions have the same upgrade code, keep the same inventory name across versions of the MSI wrapper. For information about the inventory name, see "InventoryName" on page 90.

The parameter works only when the `MSIFilename` parameter requests the generation of a Windows Installer database.

Do not modify the `UpgradeCode` value unless the new value is a valid GUID.

Examples

The capture process can create an MSI file with D89F1994-A24B-3E11-0C94-7FD1E13AB93F as the upgrade code.

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

MSIUseCabs

The `MSIUseCabs` parameter determines the use of `.cab` files that can affect application performance.

ThinApp sets an initial value that compresses the package files in a `.cab` file and makes it easier to move the file. The `.cab` file is in the MSI file.

Examples

You can modify the `MSIUseCabs` parameter to avoid a `.cab` file when it slows down the installation process for applications. You can distribute the MSI file and individual executable files in the `/bin` directory to install the application.

```
[BuildOptions]
MSIUseCabs=0
```

Configuring Sandbox Storage and Inventory Names

You can modify ThinApp parameters to configure the sandbox where all changes that the captured application makes are stored. The ThinApp inventory name might affect the need to change the sandbox name.

For more information about the sandbox placement and structure, see Chapter 6, "Locating the ThinApp Sandbox," on page 93.

InventoryName

The `InventoryName` parameter is a string that inventory tracking utilities use for package identification. This parameter determines the default names of the project folder and sandbox during the application capture process.

The application capture process sets a default value for the `InventoryName` parameter based on new strings created under one of the following locations:

- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall`
- `HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall`

The `thinreg.exe` utility and ThinApp MSI files reference the inventory name to determine the product name for display in the Add or Remove Programs control panel. For example, if the inventory name is `SuperApp` and you install an MSI file or register a package with the `thinreg.exe` utility, the Add or Remove programs list displays an installed application with the `SuperApp (VMware ThinApp)` string. ThinApp appends `VMware ThinApp` to the inventory name to distinguish applications that are virtualized during inventory scans.

You can use the same inventory name across different versions of the same application to ensure that only the most recent version appears in Add or Remove Programs list. The applications overwrite each other in the Add or Remove Programs list and prevent you from uninstalling all of the registered packages. If you want to uninstall more than one version, use a different inventory name for each version. For example, use Microsoft Office 2003 and Microsoft Office 2007 as inventory names rather than just Microsoft Office. When you maintain different versions of a virtual application in the same environment, you might want to change the `SandboxName` parameter to ensure that a new version has isolated user settings in a different sandbox.

If you have a package that includes other applications, you might update the inventory name manually to reflect the true contents of the package. For example, if you capture the `SuperApp` application and the package includes Java Runtime, the `InventoryName` value might appear as `Java Runtime Environment 1.5` instead of `SuperApp`. The Add or Remove Programs list displays the first application installed within the package.

Examples

You can modify the `InventoryName` parameter to Microsoft Office 2003.

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

RemoveSandboxOnExit

The `RemoveSandboxOnExit` parameter deletes the sandbox and resets the application when the last child process exits.

ThinApp stores all application changes to the registry and file system locations with WriteCopy or Full isolation mode in the sandbox. ThinApp sets an initial value of the `RemoveSandboxOnExit` parameter that maintains consistent settings for the sandbox directory across multiple application runs.

If the application creates child processes, ThinApp does not delete the sandbox until all child processes exit. Applications might be designed to leave child processes in place that can block the cleanup operation. For example, Microsoft Office 2003 leaves the `ctfmon.exe` process. You can use a script to end the `ctfmon.exe` process and child processes to force the cleanup operation to occur.

You can decide at runtime whether to use the `RemoveSandboxOnExit` script API function to delete the sandbox on exit.

Examples

You can modify the `RemoveSandboxOnExit` parameter to delete the sandbox when the application exits. When multiple users share an application under one user name, you can delete the sandbox to eliminate the previous user's registry and file system changes.

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

SandboxName

The `SandboxName` parameter specifies the name of the directory that stores the sandbox.

Thinapp sets an initial value that uses the inventory name as the sandbox name.

When you upgrade an application, the sandbox name helps determine whether users retain previous personal settings or require new settings. Changing the sandbox name with new deployments affects the need to create a new sandbox with different settings or retain the same sandbox.

Examples

When you update an application and want to use new user preferences for the application, you can modify the `SandboxName` parameter to reflect the updated version.

```
[BuildOptions]
SandboxName=My Application 2.0
```

SandboxNetworkDrives

The `SandboxNetworkDrives` parameter determines whether ThinApp directs write operations to a network drive or to the sandbox, regardless of isolation mode settings.

When you use this parameter to direct write operations to network drives, the result is the same as setting the isolation mode for the drive to Merged mode.

Examples

When you want to save space or share files for collaborative work, leave the default setting of the `SandboxNetworkDrives` parameter to direct write operations to network drives without storing changes in a sandbox.

```
[BuildOptions]
SandboxNetworkDrives=0
```

You can store changes in the sandbox and prevent the user from making changes to network drives.

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

SandboxPath

The `SandboxPath` parameter determines the path to the sandbox.

The path to the sandbox can be relative or absolute, can include folder macros or environment variables, and can exist on a network drive. For advanced information about how ThinApp sets an initial sandbox location or searches for the sandbox, see "Search Order for the Sandbox" on page 93.

You might work with the `SandboxPath` parameter to address local, USB drive, or network needs, to address space limitations in the initial sandbox location, or to move the sandbox to the desktop for troubleshooting purposes.

The sandbox path does not include the sandbox name because the `SandboxName` parameter determines that name.

Examples

You can modify the SandboxPath parameter to create the sandbox in the same directory as the executable file. If Mozilla Firefox 3.0 is the value of the SandboxName parameter, you can create the Mozilla Firefox 3.0 sandbox in the same directory that Firefox runs from.

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

You can create the sandbox in a subdirectory subordinate to the executable file location.

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

You can create the sandbox in the %AppData% folder of the user under the Thinstall directory.

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

You can store the sandbox on a mapped drive to back up the sandbox or to retain application settings for users who log in to any machine. If Mozilla Firefox 3.0 is the value of the SandboxName parameter, you can create the sandbox in Z:\Sandbox\Mozilla Firefox 3.0.

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

SandboxRemovableDisk

The SandboxRemovableDisk parameter determines whether the application can write removable disk changes to the disks or to the sandbox. Removable disks include USB flash devices and removable hard drives.

ThinApp sets an initial value that instructs the application to write removable disk file changes to the disk.

Examples

To save space, you can modify the SandboxRemovableDisk parameter to direct removable disk changes to the sandbox. Depending on the isolation mode for the removable disk, changes to files stored on the disk can reside in the sandbox or on the removable disk.

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

Locating the ThinApp Sandbox 6

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

This information includes the following topics:

Search Order for the Sandbox

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

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

- `<sandbox_name>` is Mozilla Firefox 3.0

  The `SandboxName` parameter in the `Package.ini` file determines the name. See "SandboxName" on page 91.

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

  The `SandboxPath` parameter in the `Package.ini` file determines the path. See "SandboxPath" on page 91.

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

  The application runs from this location.

- `<computer_name>` is JOHNDOE-COMPUTER

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

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

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

- %<sandbox_name>_SANDBOX_DIR%

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

- %THINSTALL_SANDBOX_DIR%

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

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

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

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

- <exe_directory>\<sandbox_name>

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

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

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

- <exe_directory>\Thinstall\<sandbox_name>

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

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

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

- <sandbox_path>\<sandbox_name>

  For example, Z:\sandboxes\Mozilla Firefox 3.0

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

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

- %AppData%\Thinstall\<sandbox_name>

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

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

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

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

Controlling the Sandbox Location

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

Store the Sandbox on the Network

You can use the SandboxPath parameter to store the sandbox on a mapped drive. A network location is useful for backing up the sandbox and for users who log in to any machine and retain their application settings. For more information about the SandboxPath parameter, see "SandboxPath" on page 91.

To store the sandbox on a mapped drive

1 Open the Package.ini file.

2 Under the SandboxName parameter, set the SandboxPath parameter to the network location.

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

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

Store the Sandbox on a Portable Device

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

For more information about the SandboxPath parameter, see "SandboxPath" on page 91.

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

1 Open the Package.ini file.

2 Under the SandboxName parameter, set the SandboxPath parameter to this value.

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

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

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

1 If the %THINSTALL_SANDBOX_DIR% or %<sandbox_name>_SANDBOX_DIR% environment variables are set, unset the variables.

2 On the portable device, create a Thinstall directory in the same directory as your captured application.

The next time the packaged application starts from the portable device, the application creates a sandbox in the Thinstall directory.

3 If the application and sandbox originally ran from another location, such as a computer, and you need the same sandbox on a portable device, copy the Thinstall directory from %AppData% to the directory where the executable file resides on the device.

ThinApp no longer uses the sandbox in the original location.

Sandbox Structure

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

The sandbox contains the following registry files:

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

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 have to consult the real file system for the real system location and again for the sandbox location. Instead, ThinApp can check for the existence of the file by consulting only the virtual registry. This ability increases the ThinApp runtime performance.

VMware does not support modifying or adding files directly to the sandbox. If you copy files to the sandbox directory, the files are not visible to the application. If the file already exists in the sandbox, you can overwrite and update the file. VMware recommends that you perform all modifications from the application itself.

Listing Virtual Registry Contents with vregtool

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

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

Creating ThinApp Snapshots and Projects from the Command Line 7

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

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

- File information for all local drives

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

- `HKEY_LOCAL_MACHINE` and `HKEY_USERS` registry trees

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

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

This information includes the following topics:

Methods of Using the snapshot.exe Utility

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

For information about the full procedure to create a ThinApp project from the command line, see "Create a Project Without the Setup Capture Wizard" on page 99.

Creating Snapshots of Machine States

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

Usage

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

Examples

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

Options

The options specify the directories or subkeys in the snapshot.

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

Creating the Template Package.ini file from Two Snapshot Files

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

Usage

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

Examples

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

ThinApp requires all of the parameters.

Creating the ThinApp Project from the Template Package.ini File

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

Usage

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

Examples

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

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

Displaying the Contents of a Snapshot File

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

Usage

```
snapshot.exe SnapshotFileName.snapshot –Print
```

Examples

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

ThinApp requires all of the parameters.

Sample snapshot.exe Commands

Table 7-1 describes sample commands for the `snapshot.exe` utility. The parameters are not case-sensitive. The commands are wrapped in the Command column because of space restraints.

Table 7-1. 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 `C:\` and `E:\` drives. ThinApp does not capture registry information.
`snapshot C:\Capture.snapshot C:\data.snapshot C:\ HKEY_LOCAL_MACHINE`	Captures a complete snapshot of the `C:\` drive 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 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`	Generates a ThinApp project by comparing two saved states.

Create a Project Without the Setup Capture Wizard

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

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

To create a project with the snapshot.exe command-line utility

1 Save an initial snapshot of the current machine configuration to disk.

```
snapshot.exe C:\Start.snapshot
```

2 Install the application and make any necessary manual system changes.

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

```
snapshot.exe C:\End.snapshot
```

4 Generate a template `Package.ini` file.

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

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

5 Generate a ThinApp project.

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

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

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

Customizing the snapshot.ini File

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

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

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

If the `snapshot.ini` file excludes the `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Connections` key by default, you can remove this key from the `snapshot.ini` file to ensure that ThinApp captures the key in the capture process.

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

- `Application Data\Thinapp\snapshot.ini`

  This location is the `AppData` directory of the user.

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

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

ThinApp File System Formats and Macros

8

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

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

Virtual File System Formats

ThinApp generates the following virtual file system formats:

- Build

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

- Embedded

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

- Sandbox

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

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

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

ThinApp Folder Macros

ThinApp uses macros to represent file system path locations that might change when virtualized applications run on different Windows operating systems or computers. The use of macros 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 transparently converts C:\WINNT to %SystemRoot% during the capture process for that system and expands %SystemRoot% to C:\Windows during runtime for that system.

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

On Windows Vista, ThinApp moves Windows SxS DLLs and policy information to match Windows Vista instead of using Windows XP file path styles. This feature 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 ThinApp Macros

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

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

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

Table 8-1 lists the available folder macros.

Table 8-1. Folder Macros

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

Table 8-1. Folder Macros (Continued)

Macro Name	Typical Location
%Internet Cache%	C:\Documents and Settings\<user_name>\Local Settings\Temporary Internet Files
%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
%PrintHood%	C:\Documents and Settings\<user_name>\PrintHood
%Profile%	C:\Documents and Settings\<user_name>
%Profiles%	C:\Documents and Settings
%Program Files Common%	C:\Program Files\Common Files
%ProgramFilesDir%	C:\Program Files
%Programs%	C:\Documents and Settings\<user_name>\Start Menu\Programs
%Recent%	C:\Documents and Settings\<user_name>\My Recent Documents
%Resources%	C:\Windows\Resources
%Resources Localized%	C:\Windows\Resources\<language_ID>
%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% in a Terminal Services Environment

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

Creating ThinApp Scripts

9

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

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

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

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

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

This information includes the following topics:

Callback Functions

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

Callback function names include the following names:

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

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

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

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

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

Implement Scripts in a ThinApp Environment

You might implement a script in the following circumstances:

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

To implement a script

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

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

2 Rebuild the application.

.bat Example

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

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

Timeout Example

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

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

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

Modify the Virtual Registry

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

To modify the registry

1 Obtain the path to the package executable files.

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

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

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

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

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

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

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

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

.reg Example

The following script imports the registry values from an external `.reg` file into the virtual registry at runtime.

```
Function OnFirstParentStart
ExecuteVirtualProcess "regedit /s C:\tmp\somereg.reg"
End Function
```

Stopping a Service Example

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

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

Copying a File Example

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

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

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

```
Function OnFirstParentStart
```

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

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

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

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

The source file to copy into the virtual environment is the package path plus config.ini.

```
SourceFile = SourcePath + "Config.ini"
```

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

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

Use the file systemObject parameter to check the source file exists.

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

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

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

Add a Value to the System Registry

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

To add a value to the system registry

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

```
Function OnFirstParentStart
```

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

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

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

3 Construct the .reg file.

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

4 Add the information in the system registry.

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

Wait until the process is complete.

5 Clean the environment.

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

API Functions

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

AddForcedVirtualLoadPath

The `AddForcedVirtualLoadPath(Path)` function instructs ThinApp to load all DLLs from the specified path as virtual DLLs even if they are not located in the package.

Use this function if the application needs to load external DLLs that depend on DLLs located inside the package.

You can use the `ForcedVirtualLoadPaths` parameter in the `Package.ini` file to achieve the same result as this API function. See "ForcedVirtualLoadPaths" on page 65.

Parameters

`Path`

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

Examples

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

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

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

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

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

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

```
AddForcedVirtualLoadPath(SourcePath)
```

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

ExitProcess

The `ExitProcessExitCode` function quits the current process and sets the specified error code.

Parameters

`ExitCode`

[in] The error code to set. This information might be available to a parent process. A value of 0 indicates no error.

Examples

You can exit the process and indicate success.

```
ExitProcess 0
```

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

ExpandPath

The `ExpandPath(InputPath)` function converts a path from macro format to system format.

Parameters

`InputPath`

[in] A path in macro format.

Returns

The expanded macro path in system format.

Examples

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

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

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

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

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

ExecuteExternalProcess

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

Parameters

`CommandLine`

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

Returns

Integer process ID. You can use the process ID with the `WaitForProcess` function. See "WaitForProcess" on page 115.

Examples

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

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

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

ExecuteVirtualProcess

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

Parameters

`CommandLine`

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

Returns

Integer process ID. You can use the process ID with the `WaitForProcess` function. See "WaitForProcess" on page 115.

Examples

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

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

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

GetBuildOption

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

Parameters

`OptionName`

[in] Name of the setting.

Returns

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

Examples

```
Package.ini contains:
[BuildOptions]
CapturedUsingVersion=4.0.1–2866
```

The following line appears in a VBS file:

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

GetFileVersionValue

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

Parameters

`Filename`

[in] The name of the filename whose version information is being retrieved.

Value

[in] The name of the value to retrieve from the version information section of the specified file.

You can retrieve the following values from most DLLs:

- Comments
- InternalName
- ProductName
- CompanyName
- LegalCopyright
- ProductVersion
- FileDescription
- LegalTrademarks
- PrivateBuild
- FileVersion
- OriginalFilename
- SpecialBuild

Returns

This function returns a string value. If the requested filename does not exist, or the function cannot locate the specified value in the file, the function returns an empty string ("").

Examples

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

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

End if
```

GetCommandLine

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

Returns

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

Examples

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

GetCurrentProcessName

The `GetCurrentProcessName` function accesses the full virtual path name of the current process.

Returns

This function returns a string that represents the full executable path name inside of the virtual environment. In most circumstances, this path is `C:\Program Files\...`, even if the package source runs from a network share.

Examples

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

GetOSVersion

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

Parameters

This function has no parameters.

Returns

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

`MAJOR` is one the following values:

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

`MINOR` is one of the following values:

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

`BUILD_NUMBER` is the build number of the operating system.

PLATFORM_ID assigns one of the following values:

- `Value = 1` for Windows Me, Windows 98, or Windows 95 (Windows 95 based OS)
- `Value = 2` for Windows Server 2003, Windows XP, Windows 2000, or Windows NT. (Windows NT based OS)

`OS_STRING` represents information about the operating system such as `Service Pack 2`.

Examples

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

GetEnvironmentVariable

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

Parameters

`Name`

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

Returns

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

Examples

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

RemoveSandboxOnExit

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

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

Parameters

`Yes No`

[in] Do you want to clean up when the last process shuts down? 1=Yes, 0=No

Examples

The following example turns on cleanup.

```
RemoveSandboxOnExit 1
```

The following example turns off cleanup.

```
RemoveSandboxOnExit 0
```

SetEnvironmentVariable

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

Parameters

`Name`

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

`Value`

[in] The value to store.

Examples

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

SetfileSystemIsolation

The `Setfile systemIsolation(Directory, IsolationMode)` function sets the isolation mode of a directory.

Parameters

`Directory`

[in] Full path of the directory whose isolation mode is to be set.

`IsolationMode`

[in] Isolation mode to set.

1 = WriteCopy
2 = Merged
3 = Full

Examples

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

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

SetRegistryIsolation

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

Parameters

`RegistryKey`

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

`IsolationMode`

[in] Isolation mode to set.

1 = WriteCopy
2 = Merged
3 = Full

Examples

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

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

WaitForProcess

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

Parameters

`ProcessID`

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

`TimeOutInMilliSeconds`

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

Returns

This function returns an integer.

0 = Timeout fails
1 = Process exits
2 = Process does not exist or security is denied

Examples

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

Monitoring and Troubleshooting ThinApp

10

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

This information includes the following topics:

Providing Information to Technical Support

VMware technical 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. Specify the Windows operating system, the use of Terminal Server or Citrix Xenapp, and any prerequisite programs that you installed on the native machine.
- Copies of the Log Monitor trace files. See "Log Monitor Operations" on page 117.
- Exact copy of the capture folder and all content. Do not include the compiled executable files from the `/bin` subfolder.
- Description of the expected and accurate behavior of the application.
- (Optional) Copies of the applications that you captured. Include the server components configuration for Oracle Server or Active Directory.
- (Optional) Native or physical files or registry key settings that might be relevant to the problem.
- (Optional) System services or required device drivers.
- (Optional) Virtual machine that reproduces the defect. VMware support might request this if the support contact is unable to reproduce the problem.
- (Optional) One or more WebEx sessions to facilitate debugging in your environment.

Log Monitor Operations

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

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

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

Troubleshoot Activity with Log Monitor

You can use Log Monitor to perform basic troubleshooting.

To troubleshoot ThinApp logs

1 Shut down the captured application to investigate.

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

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

3 Start the captured application.

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

4 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 **Generate text trace report**.

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

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

6 (Optional) Open the `.txt` file with a text editor and scan the information. In some circumstances, the `.txt` file is too large to open with the text editor.

7 Zip the `.txt` files and send the files to VMware support.

Perform Advanced Log Monitor Operations

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

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

To perform advanced Log Monitor operations

1 Shut down the captured application to investigate.

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

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

3 (Optional) Capture logs for a specific duration to troubleshoot an exact issue.

   a Select the **Suspend** check box.

   b Start the captured application and let it run to the point where the error occurs or the performance problem starts.

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

      You can check the application behavior to isolate the issue.

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

4 (Optional) Select a file in the trace file list to delete and click **Delete File**.

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

6 (Optional) Click the **Compress** check box to decrease the size of a trace file.

   This operation slows the performance of the application.

7 (Optional) Generate a trace file report.

   a Select a trace file in the file list, type a trace filename, or click **Browse** to select a trace file on your system.

   b (Optional) Type or change the name of the output report.

   c Click **Generate text trace report** to create a report.

      You can view the file with a text editor that supports UNIX-style line breaks.

Locating Errors

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

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

  Entries might not indicate errors. ThinApp lists each Win32 API call where the Windows error code changed.

- Look at exceptions that the applications generate.

  Exceptions can indicate errors. Exception types include C++ and .NET. The trace file records the exception type and DLL that generates the exception. If the application, such as a .NET or Java application, creates an exception from self-generating code, the trace file indicates an unknown module.

  The following example is a `.trace` entry for an exception.

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

  If you find an exception, scan the earlier part of the trace file for the source of the exception. Ignore the floating point exceptions that Virtual Basic 6 applications generate during typical use.

- 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 circumstances involving out-of-process COM, a parent application uses COM to start a child process, runs a function remotely, and continues to run functions.

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

  ThinApp addresses the slow performance of Symantec antivirus applications by restarting processes.

- Search for the error message displayed in dialog boxes.

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

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

  The log format specifies the DLL and thread that makes a call. You can often ignore the calls from system DLLs.

Log Format

A trace file includes the following sections:

- System configuration

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

  The information starts with a `Dump started on` string and ends with a `Dump ended on` string.

- Header

  This section shows contextual information for the instance of the process that Log Monitor tracks. Some of the displayed attributes show logging options, address ranges when the operating system runtime is loaded, and macro mapping to actual system paths.

  ThinApp marks the beginning of the header section with sequence number 000001. In typical circumstances, ThinApp marks the end of this section with a message about the Application Sync utility.

- Body

  This section includes trace activity as the application starts and performs operations. Each line represents function calls that target executable files or one of the DLLs make.

  The section starts with a `New Modules detected in memory` entry followed by the `SYSTEM_LOADED` modules list. The section ends with a `Modules Loaded` entry.

- Summary

  This section includes modules that the captured application loads, potential errors, and a profile of the 150 slowest calls.

  The section starts with the `Modules Loaded` message.

General API Log Message Format

The following message shows a format example for API calls.

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

This example includes the following entries:

- `000257` indicates the log entry number. Each log entry has a unique number.
- `0a88` indicates the current running thread ID. If the application has one thread, this number does not change. If two or more threads record data to the log file, you might use the thread ID to follow thread-specific sequential actions because ThinApp records log entries in the order in which they occur.
- `mydll.dll` indicates the DLL that makes the API call.
- `4ad0576d` indicates the return address for the API call that `mydll.dll` makes. In typical circumstances, the return address is the address in the code where the call originates.
- `–>` indicates the process of entering the call. For the call entry log element, ThinApp displays the input parameters. These parameters are in and in/out parameters.

- <– indicates the process of the call returning to the original caller. For call exit log entries, ThinApp displays the output parameters. These parameters are out and in/out parameters.
- kernel32.dll indicates the DLL where the API call lands.
- 7c81b1f0 indicates the address of the API inside kernel32 where the call lands. If you disassemble kernel32.dll at the 7c81b1f0 address, you locate the code for the SetConsoleMode function.
- –>BOOL=1h indicates the API returns the value of 1 and the return code has the BOOL type.

Application Startup Information

The following entries shows basic information about the application, such as the module name and process ID (PID), and about Log Monitor, such as the version and options.

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

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

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

List of DLLs Loaded into Memory During Runtime

The Modules loaded section is located near the end of the log file and describes the DLLs that are loaded into memory at runtime and the DLL addresses. The information shows whether Windows or ThinApp loads the DLLs.

This example includes a summary of the length of the longest calls and the following entries:

- SYSTEM_LOADED indicates that Windows loads the DLL. The file must exist on the disk.
- MEMORY_MAPPED_ANON indicates that ThinApp loads the DLL. ThinApp might load the file from the virtual file system.
- 46800000–46873fff indicates the address range in virtual memory where the DLL resides.
- PRELOADED_BY_SYSTEM and PRELOADED_MAP are duplicate entries and refer to the memory address range where the executable image file is mapped into memory.

```
---Modules loaded --
PRELOADED_MAP 00400000–00452fff, C:\Program Files\Adobe\Reader
8.0\Reader\AcroRd32.exe
PRELOADED_BY_SYSTEM 00400000–00452fff, C:\Program Files\Adobe\Reader
8.0\Reader\AcroRd32.exe
SYSTEM_LOADED 00400000–00452fff, C:\test\AcroRd32.exe
MEMORY_MAPPED_ANON 013b0000–020affff, C:\Program Files\Adobe\Reader
8.0\Reader\AcroRd32.dll

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

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

765380728 cycles (274.01 ms) on log entry 21753
428701805 cycles (153.48 ms) on log entry 191955
410404281 cycles (146.93 ms) on log entry 193969
.
```

```
.
... 438 total calls
7847975891 total cycles (2809.64 ms): |sprof| ts_load_internal_module
764794646 cycles (273.80 ms) on log entry 21753
426837866 cycles (152.81 ms) on log entry 191955
408570540 cycles (146.27 ms) on log entry 193969
.
.
... 94 total calls
4451728477 total cycles (1593.76 ms): |sprof| ts_lookup_imports
544327945 cycles (194.87 ms) on log entry 21758
385149968 cycles (137.89 ms) on log entry 193970
187246661 cycles (67.04 ms) on log entry 190210
.
.
... 34 total calls
1099873523 total cycles (393.76 ms): |sprof| new_thread_start
561664565 cycles (201.08 ms) on log entry 151922
531551734 cycles (190.30 ms) on log entry 152733
1619002 cycles (0.58 ms) on log entry 72875
```

Potential Errors

The `Potential Errors Detected` section marks log entries that might post problems with three asterisks (***). For information about interpreting this section, see "Locating Errors" on page 119.

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

006425 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\Microsoft.Windows.Common-Controls.DLL' flags=2 -> 0 (failed ***)
006427 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\Microsoft.Windows.Common-Controls\Microsoft.Windows.Common-Controls.DLL' flags=2
-> 0 (failed ***)
006428 0000089c nview.dll :1005b94b<-kernel32.dll:7c80ae4b *** LoadLibraryW -
>HMODULE=7c800000h () *** GetLastError() returns 2 [0]: The system cannot find the file
specified.
007062 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\en-US\Microsoft.Windows.Common-Controls.DLL' flags=2 -> 0 (failed ***)
010649 0000075c        LoadLibraryExW 'C:\Program Files\Adobe\Reader
8.0\Reader\en-US\Microsoft.Windows.Common-Controls\Microsoft.Windows.Common-Controls.DLL'
flags=2 -> 0 (failed ***)
019127 0000075c MSVCR80.dll :781348cc<-msvcrt.dll :77c10396 *** GetEnvironmentVariableA -
>DWORD=0h (OUT LPSTR lpBuffer=*0h <bad ptr>) *** GetLastError() returns 203 [0]: The system
could not find the environment option that was entered.
019133 0000075c MSVCR80.dll :78133003<-nview.dll :1000058c *** GetProcAddress -
>FARPROC=*0h () *** GetLastError() returns 127 [203]: The specified procedure could not be found.
019435 0000075c MSVCR80.dll :78136e08<-dbghelp.dll :59a60360 *** Getfile type ->DWORD=0h ()
*** GetLastError() returns 6 [0]: The handle is invalid.
019500 0000075c MSVCR80.dll :78134481<-nview.dll :1000058c *** GetProcAddress -
>FARPROC=*0h () *** GetLastError() returns 127 [0]: The specified procedure could not be found.
019530 0000075c MSVCR80.dll :78131dcd<-dbghelp.dll :59a603a1 *** GetModuleHandleA -
>HMODULE=0h () *** GetLastError() returns 126 [0]: The specified module could not be found.
```

Troubleshooting Example for cmd.exe Utility

In the troubleshooting example, ThinApp packages the `cmd.exe` utility with logging turned on. The example shows how you can simulate application failure by running an invalid command. If you request the `cmd.exe` utility to run the `foobar` command, the utility generates the `foobar is not recognized as an internal or external command` message. You can scan the trace file and check the `Potential Errors Detected` section to locate the API functions that modified the `GetLastError` code.

The example shows the `C:\test\cmd_test\bin\foobar.*`, `C:\WINDOWS\system32\foobar.*`, and `C:\WINDOWS\foobar` paths as the locations where the `cmd.exe` utility looks for the `foobar` command.

The example shows the %drive_C%\test\cmd_test\bin, %SystemSystem%\foobar, and %SystemRoot%\foobar paths as the locations in the virtual file system that ThinApp probes.

```
----Potential Errors Detected ---
*** Unable to determine if any services need to be auto-started, error 2
001550 *** FindFirstFileW 'C:\test\cmd_test\bin\foobar.*' -> INVALID_HANDLE_VALUE *** failed
[system probe C:\test\cmd_test\bin\foobar.* -> ffffffffh][no virtual or system matches]
*** FindFirstFileW ->HANDLE=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]
```

Perform Advanced Examination for cmd.exe Log Entries

A more thorough examination of an entry from the Potential Errors section of a trace file might involve searching the full body of the Log Monitor trace file for that specific entry and reviewing the system calls and conditions leading to the potential error.

For example, the following entry for the cmd.exe utility in the Potential Errors section might require a more thorough examination throughout the Log Monitor trace file.

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

To perform an advanced examination of the cmd.exe entry

1 To determine why the cmd.exe utility probes c:\test\cmd_test\bin, scan the log for this log entry number and determine what occurs before this call.

2 To determine the locations where the cmd.exe utility obtains the c:\test\cmd_test path, scan the log for GetCurrentDirectoryW and GetFullPathNameW entries.

```
000861 0a88 cmd.exe  :4ad01580->USERENV.dll :769c0396 GetCurrentDirectoryW (IN DWORD
nBufferLength=104h)
000862 0a88                    GetCurrentDirectoryW -> 0x14 (C:\test\cmd_test\bin)
000863 0a88 cmd.exe  :4ad01580<-USERENV.dll :769c0396 GetCurrentDirectoryW ->DWORD=14h
(OUT LPWSTR lpBuffer=*4AD34400h->L"C:\test\cmd_test\bin")
000864 0a88 cmd.exe  :4ad05b74->ole32.dll :774e03f0 Getfile type (IN HANDLE hFile=7h)
000865 0a88                    Getfile type 7 -> 0x2
000866 0a88 cmd.exe  :4ad05b74<-ole32.dll :774e03f0 Getfile type ->DWORD=2h ()
.
.
001533 0a88 cmd.exe  :4ad01b0d<-kernel32.dll:7c80ac0f SetErrorMode ->UINT=0h ()
001534 0a88 cmd.exe  :4ad01b13->kernel32.dll:7c80ac0f SetErrorMode (IN UINT uMode=1h)
001535 0a88 cmd.exe   :4ad01b13<-kernel32.dll:7c80ac0f SetErrorMode ->UINT=0h ()
001536 0a88 cmd.exe  :4ad01b24->IMM32.DLL :7639039b GetFullPathNameW (IN LPCWSTR
lpFileName=*1638C0h->L"," IN DWORD nBufferLength=208h)
001537 0a88                    GetFullPathNameW . -> 20 (buf=C:\test\cmd_test\bin,
file_part=bin)
001538 0a88 cmd.exe  :4ad01b24<-IMM32.DLL :7639039b GetFullPathNameW ->DWORD=14h
(OUT LPWSTR lpBuffer=*163D60h->L"C:\test\cmd_test\bin," OUT *lpFilePart=*13D8D4h-
>*163D82h->L"bin")
.
.
001549 0a88 cmd.exe   :4ad01b5f->USERENV.dll :769c03fa FindFirstFileW (IN LPCWSTR
lpFileName=*1638C0h->L"C:\test\cmd_test\bin\foobar.*")
001550 0a88                    FindFirstFileW 'C:\test\cmd_test\bin\foobar.*' ->
INVALID_HANDLE_VALUE *** failed [system probe C:\test\cmd_test\bin\foobar.* -> ffffffffh][no
virtual or system matches]
```

The cmd.exe utility obtains the first location by calling GetCurrentDirectoryW and the second location by calling GetFullPathNameW with "." as the path specifies. These calls return the path for the current working directory. The log file shows that the cmd.exe utility creates the C:\test\cmd_test\bin> prompt. The utility queries the PROMPT environment variable that returns PG and uses the WriteConsoleW API function to print the prompt to the screen after internally expanding PG to C:\test\cmd_test\bin>.

Troubleshooting Specific Applications

Troubleshooting tips are available for capturing Microsoft Outlook, Explorer.exe, and Java Runtime Environment.

Troubleshoot Registry Setup for Microsoft Outlook

Microsoft Outlook stores account settings in registry keys and files. When you start Microsoft Outlook for the first time, it checks that the keys exist. If Microsoft Outlook cannot locate the keys, it prompts you to create a new account.

This process works properly in the virtual environment when Microsoft Outlook is not installed on the physical system. If the user already has Microsoft Outlook installed on the physical system, the captured version finds the registry keys in the system registry and uses those settings. You must use Full isolation mode for the registry keys and files where Microsoft Outlook stores its settings.

To set up Full isolation mode for Microsoft Outlook registry keys

1 Add the following entries to the HKEY_CURRENT_USER.txt file:

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

2 Create a ##Attributes.ini file with the following entries:

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

3 Place the ##Attributes.ini file in each of the following subdirectories.

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

4 (Optional) If the subdirectories do not exist, create the directories.

Viewing Attachments in Microsoft Outlook

Microsoft Outlook creates a default directory to store attachments when you open an attachment for viewing. The typical location is C:\Documents and Settings\<user_name>\Local Settings\Temp\Temporary Internet Files\OLK<xxxx>. The last xxxx is replaced by a random entry.

You can view attachments when the viewing application runs in the same virtual sandbox as Microsoft Outlook. External applications might not be able to find the file to display because Microsoft Outlook stores the file in the sandbox. You must use the Merged isolation mode for the directory that stores the attachments.

To set up Merged isolation mode to view Microsoft Outlook attachments

1 Add a value to the HKEY_CURRENT_USER.txt file that sets the name of the attachment directory:

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

In this example, 11.0 in the key name is for Microsoft Outlook 2003.

2 Replace the last four `xxxx` characters with random alphanumeric entries to increase security.

3 Create a directory that is named in the `OutlookSecureTempFolder` registry key in your ThinApp project.

For example, create the `%Profile%\Local Settings\OutlookTempxxxx` directory.

4 In the `%Profile%\Local Settings\OutlookTempxxxx` directory, create a `##Attributes.ini` file with the following entries:

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

Starting Explorer.exe in the Virtual Environment

Running one instance of the `explorer.exe` utility on a Windows operating system makes it difficult to add an entry point to Windows Explorer and start it inside the virtual environment.

You can use the following methods to open a Windows Explorer window inside the virtual environment:

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

  For example, add the following entries to the `Package.ini` file:

  ```
  [iexplore.exe]
  Shortcut=xxxx.exe
  Source=%ProgramFilesDir%\Internet Explorer\iexplore.exe
  CommandLine=%ProgramFilesDir%\Internet Explorer\iexplore.exe –E
  ```

- Add the following virtual registry key:

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

- Add the following entries to the `Package.ini` file:

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

  Use this method to browse the virtual file system with a familiar interface and enable accurate file type associations without system changes, especially when using portable applications. You can access shell-integrated components without system changes.

Troubleshooting Java Runtime Environment Version Conflict

A conflict might occur if one version of Java is installed on the physical system and another version is included in a captured executable file. Updated versions of Java install a plug-in DLL that Internet Explorer loads. This plug-in DLL overwrites virtual registry keys and conflicts with a virtualized copy of older Java runtimes.

To prevent Internet Explorer from loading plug-in DLLs

Add the following entry to the beginning of the `HKEY_LOCAL_MACHINE.txt` file.

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

Glossary

A

Application Link
A utility that links dependent applications to a base application at runtime and starts all the applications together when you start the base application. You can use the utility to deploy and update component packages separately rather than capture all components in the same package.

Application Sync
A utility that updates an application by detecting a new packaged version on a server or network share. You can configure update settings, such as the checking of an update server at certain intervals. ThinApp detects the most recent application executable file and downloads the differences.

attributes.ini
The file that applies configuration settings at the directory level of the package rather than the entire package. The `##Attributes.ini` settings override the overall `Package.ini` settings.

B

build
To convert a ThinApp project into a package. You can build a package with the Setup Capture wizard or with the `build.bat` utility.

C

capture
To package an application into a virtual environment and set initial application parameters. ThinApp provides the Setup Capture wizard or the `snapshot.exe` utility to create a portable application package that is independent of the operating system it runs on.

clean machine
The computer or virtual machine, installed with only the basic Windows operating system, on which you capture the application. The Windows operating system version must be the earliest version of Windows that you expect the application to run on.

E

entry point
An executable file that starts the captured application. An application might have multiple entry points. For example, the `Firefox.exe` file might serve as an entry point for a Mozilla Firefox application. The primary data container file can exist within an entry point or as a `.dat` file.

I

inventory name
A name that ThinApp uses for internal tracking of the application. The inventory name sets the default project directory name and appears in the **Add or Remove Programs** dialog box for Windows.

isolation mode
A package setting that determines the read and write access to the physical environment. ThinApp has WriteCopy, Merged, and Full isolation modes.

L

logging.dll
A utility that generates `.trace` files.

Log Monitor
A utility that captures chronological activity for executable files that the captured application starts.The `log_monitor.exe` file is compatible only with applications captured using the same version of ThinApp.

M

MSI
A Windows Installer container that is useful for application deployment tools. You can deliver the captured application as an MSI file instead of an executable file.

N

native
Refers to the physical environment rather than the virtual environment. *See also* physical.

network streaming
The process of running a package from a central server. ThinApp downloads blocks of the application as needed to ensure quick processing and display.

P

package
The virtual application files that the ThinApp build process generates. The package includes the primary data container file and entry point files to access the application.

package.ini
The file that applies configuration settings to the package and that resides in the captured application folder. The Setup Capture wizard sets the initial values of the configuration settings.

physical
Refers to the computer memory and file system in which all standard Windows processes run. Depending on ThinApp isolation mode settings, processes in the virtual environment can access the physical environment. *See also* native, virtual.

prescan
To establish a baseline image or snapshot of a machine before you install the application you want to capture. The capture process stores in a virtual file system and virtual registry the differences between the prescan and postscan images. *See also* postscan, snapshot.

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

project
The data that the capture process creates before you build a package. The capture process uses the inventory name as the default project directory name. You can customize parameters in the project files before you build an application package. You cannot deploy a captured application until you build a package from the project.

postscan
To establish an image or snapshot of a machine after you install the application you want to capture. The capture process stores in a virtual file system and virtual registry the differences between the prescan and postscan images. *See also* prescan, snapshot.

S

sandbox
The physical system folder that stores runtime user changes to the virtual application. When you start the application, ThinApp incorporates changes from the sandbox. When you delete the sandbox, ThinApp reverts the application to its captured state. The default location of the sandbox is `%APPDATA%\Thinstall\<application_name>`.

sbmerge.exe
A utility that makes incremental updates to applications, such as the incorporation of a plug-in or a change in a browser home page. The `sbmerge.exe` utility merges runtime changes recorded in the sandbox back into a ThinApp project.

snapshot
A recording of the state of the Windows file system and registry during the application capture process. The Setup Capture process uses the `snapshot.exe` utility to take a snapshot before and after the application is installed and stores the differences in a virtual file system and virtual registry. *See also* postscan, prescan.

snapshot.exe
A utility that creates the snapshots of a computer file system and registry and facilitates the prescan and postscan operations during the capture process. Only advanced users who build ThinApp functionality into other platforms might make direct use of this utility. *See also* postscan, prescan, snapshot.

snapshot.ini
A configuration file that specifies the directories and subkeys to exclude from a ThinApp project when you capture an application. You can customize this file for applications.

T

template.msi
A template for MSI files that you can customize to adhere to company deployment procedures and standards. For example, you can add registry settings for ThinApp to add to client computers as part of the installation.

thinreg.exe
A utility that establishes file type associations, sets up **Start** menu and desktop shortcuts, and facilitates the opening of files. You must run the `thinreg.exe` utility to register executable files. MSI files automate the `thinreg.exe` registration process.

tlink.exe
A utility that links key modules during the build process.

V

vftool.exe
A utility that compiles the virtual file system during the build process.

virtual
Refers to the logical file and memory within which a captured application runs. Processes in a physical environment cannot access the virtual environment. *See also* physical.

virtual application
An application that you capture to make it portable and independent of the operating system it runs on.

virtual file system
The file system as the captured application sees it.

virtual registry
The registry as the captured application sees it.

vregtool.exe
A utility that compiles the virtual registry during the build process.

Index

W