ThinApp Package.ini Parameters Reference Guide

ThinApp 5.1

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-000759-03

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 Guide 1

The *ThinApp Package.ini Parameters Reference Guide* provides information on how to configure and customize the ThinApp `Package.ini` parameters. You can refer this guide to customize the parameters of the virtual application outside of the capture process.

Intended Audience

This book is intended for anyone who has to customize the `Package.ini` parameters of the captured applications. Typical users are system administrators responsible for the distribution and maintenance of corporate software packages.

Configuring Package Parameters 2

Advanced users can customize the parameters of a 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 for your virtual application. The file contains parameters that configure a captured application during the build process. The Setup Capture wizard sets the initial values of some `Package.ini` parameters. You can save changes to the `Package.ini` file and build the project to have the parameter changes take effect.

Package.ini File Structure 3

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]

The `[BuildOptions]` section of the `Package.ini` file applies to all applications. Individual applications inherit these parameters unless application-specific entries override the 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.

[<application>.exe]

[FileList]

The `[FileList]` parameters act as `[BuildOptions]` parameters. They are grouped separately from other `[BuildOptions]` parameters for backward compatibility reasons. You can add the `[FileList]` heading manually to the file when you add the `ExcludePattern` parameter.

[Compression]

The `[Compression]` parameters act as `[BuildOptions]` parameters. They are grouped separately from other `[BuildOptions]` parameters for backward compatibility reasons.

[Isolation]

The `[Isolation]` parameters act as `[BuildOptions]` parameters, They are grouped separately from other `[BuildOptions]` parameters for backward compatibility reasons.

Package.ini or ##Attributes.ini Files That Override Package.ini Settings

4

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

You can use the `DirectoryIsolationMode`, `CompressionType`, and `ExcludePattern` parameters in an `##Attributes.ini` file to override directory-level `Package.ini` settings. The `##Attributes.ini` file is located in the folder macros of the project folder.

For more information about modifying the `##Attributes.ini` file settings, see the *ThinApp User's Guide.*

Configuring the ThinApp Runtime 5

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

1 RuntimeEULA Parameter on page 15

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

2 VirtualComputerName Parameter on page 16

The `VirtualComputerName` parameter determines whether to rename the computer name, to avoid naming conflicts between the capture process and the deployment process.

3 QualityReportingEnabled Parameter on page 16

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

RuntimeEULA Parameter

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

NOTE Do not modify the value of this parameter.

Example: Default RuntimeEULA Value

This example shows how the `RuntimeEULA` parameter prevents the display of the End-User License Agreement.

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

VirtualComputerName Parameter

The `VirtualComputerName` parameter determines whether to rename the computer name, to avoid naming conflicts between the capture process and the deployment process.

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 process is different from the deployment process, captured applications that require a computer name must add the computer name to the virtual package 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 the `GetComputerName` and `GetComputerNameEx` API functions return in a virtual application.

Example: Including a Virtual Computer Name

This example shows how the `VirtualComputerName` parameter creates a second name for a computer named LOCALHOST, which will be captured in the virtual application. The application uses the second name to connect to a virtual machine. If the capture system lacks 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 name that the `VirtualComputerName` parameter created. The virtual application works with the renamed LOCALHOST name because any computer that the application runs on receives this value as the computer name.

If you run a `GetComputerName` or `GetComputerNameEx` command, the computer returns *LOCALHOST*. If the Windows system requires the `GetComputerName` and `GetComputerNameEx` commands 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*.

```
VirtualComputerName=LOCALHOST
```

Example: Including an Environment Variable

In addition to specifying a literal string such as *LOCALHOST*, you can include an environment variable.

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*.

```
VirtualComputerName=%VCOMPNAME%
```

QualityReportingEnabled Parameter

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

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

Example: Turning Off Data Collection

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

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

Configuring Isolation 6

ThinApp isolation parameters determine the read and write access to the file system and registry keys

This chapter includes the following topics:

DirectoryIsolationMode Parameter

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 setup capture 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 find user data files that reside in the sandbox instead of the physical system, the 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.

You must place the parameter under an `[Isolation]` heading.

For more information about modifying the `##Attributes.ini` file settings, see the *ThinApp User's Guide.*

Example: Using WriteCopy Isolation

In this example, you modify the `DirectoryIsolationMode` parameter with *WriteCopy* isolation to ensure that the virtual 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.

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

Example: Using Merged Isolation

In this example, you assign *Merged* isolation mode to ensure that the virtual application can read resources on and write to any location on the computer, except where the package specifies otherwise. *Merged* is the default setting for the Setup Capture wizard.

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

RegistryIsolationMode Parameter

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.

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 from an application.

Example: Using WriteCopy Isolation in the Registry

This example shows how 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
```

Example: Using Merge Isolation in the Registry

This example shows how 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 7

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

This chapter includes the following topics:

FileTypes Parameter

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.

Example: Removing a File Extension From a Package to Ensure That it is Opened By a Version of the Application in the Physical Environment

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
```

Example: Creating File Type Extensions and Linking Them to An Application

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 Parameter

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.

Example: Specifying the mailto Protocol for A Microsoft Outlook Package

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 8

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

This chapter includes the following topics:

ExcludePattern Parameter

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 in 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.

Example: Excluding Version Control Information From the Virtual File System

If you store packages in a version control system and you want to exclude version control information 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.

Example: Excluding Paths That End With .bak or .msi

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

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

Icon Parameter

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

Each application includes 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.

Example: Using an Alternative Icon

You can modify the `Icon` parameter to use an alternative icon by specifying an executable file that differs from the executable file in the `Source` parameter.

Alternative icons might be useful for third-party companies.

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

Example: Specifying an Icon

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
```

Example: Using an ICO File

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 Parameter

The `OutDir` parameter specifies the directory that stores the `build.bat` output. Do not modify the value of this parameter.

Example: Specifying the BIN Directory for the Project

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

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

RetainAllIcons Parameter

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

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.

Example: Keeping All Original Icons of an Application

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 9

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

Permissions parameters are generally specified under the `[build options]` section.

This chapter includes the following topics:

AccessDeniedMsg Parameter

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

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

Example: Adding a Contact Number

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
```

PermittedGroups Parameter

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 only supports 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 that 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.

Example: Specifying a List of Active Directory Members

You can modify the `PermittedGroups` parameter to specify a list of Active Directory user group names, separated by semicolons. The parameters in the `[BuildOptions]` section 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
```

Example: Overwriting Global PermittedGroups Setting

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
```

Example: Inheriting the Global PermittedGroups Setting

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

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

Example: Mixing Group Names and SID Strings

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 Parameter

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.

Example: Specifying Administrator Privileges for an Application

You can modify the `UACRequestedPrivilegesLevel` parameter to specify administrator privileges for an application.

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

UACRequestedPrivilegesUIAccess Parameter

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.

Example: Ensuring an Application Cannot Access Protected Elements

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

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

Configuring Objects and DLL Files 10

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

This chapter includes the following topics:

ExternalCOMObjects Parameter

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 for 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, all changes that the COM objects make are stored by ThinApp in the sandbox.

The capture process does not generate this parameter. You can add this parameter to the `Package.ini` file under the `[BuildOptions]` section.

Example: Running COM Objects Outside the Virtual Environment

When you troubleshoot a problem with VMware support and determine that an application implements COM objects that are incompatible with ThinApp, you can modify the ExternalCOMObjects parameter to run the COM objects outside the virtual environment.

You can list the CLSID keys.

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

ExternalDLLs Parameter

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 in the virtual file system. In this case, Windows cannot load the DLL file.

Example: Forcing Windows to Load DLL Files From the Virtual File System

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 Parameter

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 an application must load external system DLL files that depend on DLL files located in 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

Example: Forcing ThinApp to Load DLLs in the Virtual Environment

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 Parameter

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.

This 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 a semicolon (`;`).

The following parameters are described in this section:

Example: Isolating a Memory Object

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 Parameter

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 a semicolon (;).

Example: Isolating a Synchronization Object

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 Parameter

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. However, ThinApp SDK users can add this parameter to the file.

Example: Making Calls to DLL Files

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

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

NotificationDLLSignature Parameter

The NotificationDLLSignature parameter works with the NotificationDLLs parameter and verifies that a specified DLL file has a signature.

If the DLL lacks a signature, ThinApp does not load the file.

Example: Ensuring That a DLL File is Authenticode-signed

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

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

Example: Ensuring That a DLL File is Entity-signed

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

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

ObjectTypes Parameter

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 that were generated by the capture process.

Example: Starting a Virtual Application When a COM Object is Created

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 Parameter

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 COM objects that the virtual application registers in the sandbox.

Example: Making COM Objects Registered Outside the Sandbox Visible

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 Parameter

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 addresses 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 for 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. See "ExternalCOMObjects Parameter," on page 31.

Example: Running Out-of-process COM Objects in the Physical Environment

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
```

11 Configuring File Storage

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

This chapter includes the following topics:

CachePath Parameter

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 for the `CachePath` parameter based on the `SandboxPath` parameter. The value is set 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 where the user travels with the sandbox. ThinApp creates a `Stubs` directory within the cache.

Example: Setting an Absolute Path

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

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

Example: Setting a Relative Path

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
```

Example: Setting the Cache and Sandbox at the Same Directory Level

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

```
CachePath=<sandbox_path>
```

UpgradePath Parameter

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

The default `UpgradePath` location is the same directory as that in which the application executable file is located on the local machine.

The Application Sync utility accumulates log and cache files and constructs the updated executable file in the `UpgradePath` location.

Side-by-side integer updating looks for updated versions of the application in the `UpgradePath` location.

Example: Specifying an Alternative Location for Storing Side-by-side Updates

When the default location, such as a USB device, has limited space, or to isolate upgrades from the application executable file, you can modify the `UpgradePath` parameter to specify an alternative location for storage of side-by-side updates.

The parameter can include environment variables in the path, but does not support folder macros.

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

VirtualDrives Parameter

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.

Modifying Virtual Drive Isolation Modes

You can modify isolation modes for virtual drives, to override the default isolation mode of the project.

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.

Example: Assigning a Serial Number and FIXED Type to a Virtual Drive

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
```

Example: Assigning a Drive Letter to a Virtual Drive

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

- 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.

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

Configuring Processes and Services 12

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.

This chapter includes the following topics:

AllowExternalKernelModeServices Parameter

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.

Example: Allowing an Application to Create or Open a Native Windows Kernel Driver Service

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 Parameter

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 a 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.

Example: Supporting Write Operations From Virtual Processes to Native Processes

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
```

AutoShutdownServices Parameter

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

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

Example: Keeping a Virtual Service Running After the Application Exits

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 Parameter

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 automatic startup type option. The virtual services start when the user runs the first parent process.

Example: Preventing a Virtual Service From Starting

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 Parameter

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 Parameter," on page 43.

Example: Create Child Processes in the Physical Environment

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 Parameter

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 in the virtual environment.

Example: Specifying Exceptions to Running Child Processes in the Virtual Environment

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
```

13 Configuring Sizes

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

This chapter includes the following topics:

BlockSize Parameter

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.

Example: Increasing the Size of a Block

You can increase the 64KB default size of 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 Parameter

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 to the ZIP algorithm.

Table 13-1. Sample Compression Ratios and Startup Times for a Microsoft Office 2003 Package Running From a Local Hard Drive

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 second
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 in which 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 Parameter," on page 46.

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.

Example: Optimizing for Moderate Disk Space Requirements

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
```

Example: Optimizing for Maximum Disk Space Requirements

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

```
[Compression]
CompressionType=Fast

[BuildOptions]
OptimizeFor=Disk
```

MSICompressionType Parameter

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.

Example: Preventing MSI File Compression

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
```

Configuring Logging 14

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

This chapter includes the following topics:

DisableTracing Parameter

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 correctly. Producing unnecessary `.trace` files wastes disk space and CPU time.

Example: Preventing Generation of Trace Files in Log Monitor

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

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

LogPath Parameter

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 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%`.

Example: Specifying a Log Storage Directory

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

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

Configuring Versions 15

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

This chapter includes the following topics:

CapturedUsingVersion Parameter

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.

Example: Displaying the ThinApp Version Number With Which an Application Was Captured

The `CapturedUsingVersion` parameter might display ThinApp version 4.0.0-2200.

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

StripVersionInfo Parameter

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. The 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.

Example: Generating an Application Without Version Information

In rare cases, you might 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 Parameter

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.

Example: Setting a New Product Name

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 16

You can use ThinApp parameters to verify locale information.

This chapter includes the following topics:

AnsiCodePage Parameter

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.

Example: Verifying AnsiCodePage parameter

When the operating systems of the deployed and captured computers have different languages, you can check the `AnsiCodePage` parameter to know the operating system language of the computer on which you capture the application.

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

LocaleIdentifier Parameter

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.

Example: Setting a LocaleIdentifier Parameter

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 Parameter

The `LocaleName` parameter displays the name of the locale when you capture an application on Microsoft Vista.

Displaying the LocaleName of a Captured Application

The `LocaleName` parameter can display a Japanese locale name.

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

Configuring Individual Applications 17

You can modify ThinApp parameters to configure specific applications.

Parameters specific to entry points are specified 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.

This chapter includes the following topics:

CommandLine Parameter

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 pathname conventions.

Example: Modifying a CommandLine Parameter

You can modify the `AnsiCodePage` 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 Parameter

The `Disabled` parameter determines whether the application build target is only a placeholder, in which case it 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 **Start** 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.

Example: Modifying Disabled parameter to generate entry points

If you do not select the `cmd.exe, regedit.exe,` or `iexplore.exe` troubleshooting entry points during the capture process, and you subsequently 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 Parameter

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.

NOTE Do not modify this parameter. The `Package.ini` file displays this parameter in case you want to find 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.

Example: Setting a ReadOnlyData Parameter values

ThinApp sets the required value of the `ReadOnlyData` that specifies `Package.ro.tvr` as the name of the virtual registry file.

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

Shortcut Parameter

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 Parameter," on page 56

NOTE Do not modify the value of the `Shortcut` parameter. ThinApp detects the primary data container during the capture process.

Example: Pointing to an Executable File of an Application

ThinApp can point the shortcut executable file `AcroRd32.exe` to the `Adobe Reader 8.exe` parameter, which is the primary data container.

```
[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 Parameter

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.

Example: Modifying a Shortcuts Parameter

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 Parameter

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 find 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` 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.

NOTE 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.

Example: Pointing to an Entry Point using the Source Parameter

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 Parameter

The `WorkingDirectory` parameter determines the first location in which an application looks for files or 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 pathname conventions.

The `WorkingDirectory` parameter sets the initial value of the working directory but the directory is dynamic as you navigate to other locations.

Example: Modifying the WorkingDirectory Parameter value

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. To map the working directory to the `My Documents` directory on the physical system, use the Merged isolation mode setting. To map the working directory to the sandbox on the local machine, use the WriteCopy or Full isolation mode setting.

Configuring Dependent Applications Using the Application Utility 18

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 keep the benefits of application virtualization.

ThinApp can link up to 250 packages at a time. Each package can be any size. 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 change the preferences, ThinApp stores the changes in the sandbox for Acrobat Reader. When you start Firefox, Firefox cannot detect those changes because Firefox has its own sandbox. Opening a `.pdf` file with Firefox does not reflect the preference changes that exist in the standalone Acrobat Reader application.

This chapter includes the following topics:

Application Link Pathname Formats

The Application Link utility supports the following pathname 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 system or user environment variables that dynamically expand to a specific location for each user or computer. 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 include spaces.
- Path names can specify multiple links or dependencies with a semicolon that separates individual filenames. An example is `RequiredAppLinks=Dependency1.exe; Dependency2.exe;.`
- Path names can contain asterisk and query wildcard characters (* and ?) in filenames and directory paths. For example, `RequiredAppLinks=WildPath*\WildFilename*.dat.`

  If a path containing a wildcard character matches more than one directory in the file system, each matching directory name will be returned, to enable additional path or filename matching.

  Wildcards that are used in combination with environment variables can provide powerful customized recursive searching for dependent applications. For example, `OptionalAppLinks=%HOMEPATH%\OfficePlugins\*\*`.

RequiredAppLinks Parameter

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 pathname formats, see "Application Link Pathname Formats," on page 59.

Example: Link the Application to .NET

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 Parameter

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`.

`RequireAppLinks` and `OptionalAppLinks`parameters use the same syntax. For information about the `RequireAppLinks`parameter and examples, see "RequiredAppLinks Parameter," on page 60.

When using the Application Link utility via the `OptionalAppLinks` parameter, the base package and linked dependencies must be built with the same version of ThinApp. To upgrade older packages to include the latest version of the ThinApp runtime, use the `relink.exe` command-line utility included with ThinApp.

Configuring Application Updates with the Application Sync Utility 19

The Application Sync utility keeps deployed virtual applications up to date. When an application starts, the Application Sync utility 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.The next time that the application starts after the download is complete, ThinApp activates the new version.

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
```

This chapter includes the following topics:

AppSyncClearSandboxOnUpdate Parameter

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.

Example: Modifying AppSyncClearSandboxOnUpdate Parameter

You can modify the AppSyncClearSandboxOnUpdate parameter to clear the sandbox after application updates.

```
AppSyncClearSandboxOnUpdate=1
```

AppSyncExpireMessage Parameter

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.

Example: Viewing AppSyncExpireMessage Parameter

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 Parameter

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.

Example: Setting Default value for AppSyncExpirePeriod Parameter

You can prevent the package from expiring with the default never value.

```
AppSyncExpirePeriod=never
```

AppSyncURL Parameter

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.

NOTE You must uncomment the AppSyncURL parameter to activate all Application Sync parameters.

Example: Assigning values to AppSyncURL Parameter

You can assign an HTTP or HTTPS value to the AppSyncURL parameter according to the following format.

```
AppSyncURL=https://<site.com>/<path>/<primary_data_container_name>
```

You can specify local and network drive paths. A primary data container can be either a `.exe` or `.dat` file.

```
file:///C:/<path>/<primary_data_container_name>
```

You can use a UNC path and access locations of network resources.

```
file://<server>/<share>/<path>/<primary_data_container_name>
```

AppSyncUpdateFrequency Parameter

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.

Example: Modifying AppSyncUpdateFrequency Parameter

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 Parameter

The `AppSyncUpdatedMessage` parameter sets the message that appears when an updated package first starts.

Viewing Updated AppSyncUpdatedMessage Parameter

You can use the `AppSyncUpdatedMessage` parameter to confirm that the application is updated.

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

AppSyncWarningFrequency Parameter

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.

Example: Modifying AppSyncWarningFrequency Parameter

You can modify the `AppSyncWarningFrequency` parameter to configure the warning to appear each time the application starts.

```
AppSyncWarningFrequency=0
```

AppSyncWarningMessage Parameter

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.

Example: Viewing Default Message of AppSyncWarningMessage Parameter

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 Parameter

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.

Example: Specifying Default Period for AppSyncWarningPeriod Parameter

The default period of the `AppSyncWarningPeriod` parameter is five days.

```
AppSyncWarningPeriod=5d
```

Configuring MSI Files 20

You can modify ThinApp parameters to configure MSI files for deployment through desktop management systems.

Information about compression of MSI files appears with other parameters that control file sizes. See "MSICompressionType Parameter," on page 46.

This chapter includes the following topics:

MSIArpProductIcon Parameter

The `MSIArpProductIcon` parameter specifies the icon that is used to represent the application in the Windows Add or Remove Programs dialog. The icon can reside in ICO, DLL, or executable files.

NOTE Do not modify this parameter. If an MSI package does not have an application icon, the application appears with a generic icon.

Example: Specifying an Icon using MSIArpProductIcon Parameter

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 applied only when multiple icons are available in a DLL file or executable file.

MSIDefaultInstallAllUsers Parameter

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.

The parameter is applied only when the `MSIFilename` parameter requests the generation of a Windows Installer database.

Example: Setting value for MSIDefaultInstallAllUsers Parameter

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 individually.

```
[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 Parameter

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.

Example: Replacing the MSIFilename Parameter

You can generate an MSI file during the build process and replace the filename with your own filename.

```
[BuildOptions]
MSIFilename=<my_msi>.msi
```

The inventory name is the default name in the `MSIFilename` parameter.

```
[BuildOptions]
;MSIFilename=<inventory_name>.msi
```

MSIInstallDirectory Parameter

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 is applied only when the MSIFilename parameter requests the generation of a Windows Installer database.

Example: Specifying a Directory location for MSIInstallDirectory Parameter

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 Parameter

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 is applied only when the MSIFilename parameter requests the generation of a Windows Installer database.

Example: Modifying MSIManufacturer Parameter

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 Parameter

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 is applied only when the MSIFilename parameter requests the generation of a Windows Installer database.

NOTE Do not modify the MSIProductCode parameter.

Example: Creating an MSI file using MSIProductCode Parameter

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 Parameter

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.

Example: Modifying the MSIProductVersion Parameter

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 Parameter

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.

Example: Modifying the MSIRequireElevatedPrivileges Parameter

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 Parameter

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 MSIwrapper. For information about the inventory name, see "InventoryName Parameter," on page 73.

The parameter is applied only when the `MSIFilename` parameter requests the generation of a Windows Installer database.

NOTE Do not modify the `UpgradeCode` value unless the new value is a valid GUID.

Example: Creating an MSIFile using MSIUpgradeCode Parameter

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}
```

MSIStreaming Parameter

The `MSIStreaming` parameter determines the use of `.cab` files that can affect application performance.

The default setting is `MSIStreaming=0`. With this value, the packaged `.exe` files and `.dat` files are compressed in the `.cab` file. Disabling the parameter in `Package.ini` has the same effect as setting the value to 0. When the MSI runs, the packaged files are extracted to the `Program Files` directory of the Windows operating system, and registered either to the user or the system, depending on other `Package.ini` settings.

Setting the parameter value `MSIStreaming=1`, results in the packaged `.exe` files and `.dat` files being excluded from the `.cab` file, and subsequently from the `.msi` file. This value enables a choice to be made during deployment between registering the application from a network share or installing in the`Program Files` directory of the Windows operating system.

If you set this value, ensure that both the MSI and the `.exe` and `.dat` files of the ThinApp package are available on the network share.

Example: Modifying the MSIStreaming Parameter

You can modify the `MSIStreaming` parameter to ignore 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]
MSIStreaming=1
```

MSIIs64Bit Parameter

You can use the `MSIIs64Bit`parameter to install 64-bit applications at `ProgramFiles` directory.

On 64 bit OS ThinApp installs MSI package application at `ProgramFiles(x86)` for 32 bit MSI package. For 64 bit MSI packages user should set `MSIIs64Bit` parameter in package.ini file to install application at `ProgramFiles`. This parameter will be present by default in `Package.ini` if application is captured using 5.0 build.

```
[BuildOptions]
MSIIs64Bit=1
```

Configuring Sandbox Storage and Inventory Names

21

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.

This chapter includes the following topics:

InventoryName Parameter

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 window in the 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 window 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 window. The applications overwrite each other in the Add or Remove Programs window and prevent you from uninstalling all of the registered packages. 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 window displays the first application installed within the package.

Example: Modifying the InventoryName Parameter

You can modify the InventoryName parameter to Microsoft Office 2003.

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

RemoveSandboxOnExit Parameter

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 for theRemoveSandboxOnExit 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, which 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.

Example: Modifying the RemoveSandboxOnExit Parameter

You can modify the RemoveSandboxOnExit to delete the sandbox 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 Parameter

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 keep previous personal settings or require new settings. Changing the sandbox name with new deployments affects the need to create a sandbox with different settings or keep the same sandbox.

Example: Modifying the SandboxName Parameter

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 Parameter

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.

Example: Modifying the Settings of SandboxNetworkDrives Parameter

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 Parameter

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.

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.

Example: Modifying the SandboxPath Parameter

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 keep 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 Parameter

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.

Example: Modifying the SandboxRemovableDisk Parameter

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
```

SandboxWindowClassName Parameter

When you set the `SandboxWindowClassName = 1` you can isolate the application defined window classnames created and used within the thinapp package.

Other Configuration Parameters 22

You can use some ThinApp parameters to configure printers, disable the cut and paste option, specify settings for a computer. and group SIDs and modify the user name that appears in the status bar.

This chapter includes the following topics:

DisableCutPaste Parameter

You use the `DisableCutPaste` parameter to disable the cut and paste option in the ThinApp application.

LoadDotNetFromSystem Parameter

The `LoadDotNetFromSystem` parameter specifies that the ThinApp application must use the .NET installation that is present in the user's system, not the .NET installation that is in the ThinApp application.

PermittedComputers Parameter

The `PermittedComputers` parameter allows you to increase security by specifying a limited set of Active Directory Groups of computers that are permitted to use the virtual application .

Services Parameter

The `Services` parameter specifies how to register the ThinApp package as a system service.

StatusbarDisplayName Parameter

The `StatusbarDisplayName` parameter displays the application's title in the status bar of the ThinApp application start dialog box. You can modify this value to display different application name in the status bar.

DisableTransactionRegistry Parameter

You use the `DisableTransactionRegistry` parameter to prevent data corruption when an application fails during a write operation, a power failure, or an incomplete disk flush on a removable disk present in the ThinApp package.

DisableTransactionRegistry Parameter might help with performance in some rare circumstances if your Sandbox is located on a network share.

PreventDLLInjection

You can use the `PreventDlInjection` parameter to prevent ThinApp runtime from loading a DLL, when another external application calls the `SetWindowHook` to set a global hook.

This should be used only when the hook is in conflict with a virtualized application. For example, if virtualized application is unable to start when the other application `NxPowerLite` calls `SetWindowsHook`to set a global hook `oehook.dll`, modifying the package.ini with this entry allows the virtualized application to start.

ProcessExternalNameBehavior Parameter

You can use the `ProcessExternalNameBehavior` parameter to distinguish a child process from a parent process.

By default, both child process and parent process names in Task Manager will be entry point names. But if `ProcessExternalNameBehavior` is set to default the child process will get the name of native application while parent process will still have the entry point name. This parameter allows users to distinguish between parent and child processes in some scenarios.

PreventDllInjectionExceptions Parameter

You can use the `PreventDlInjectionExceptions` parameter only when the `PreventDlInjection` parameter is set to 1. When you set the `PreventDlInjectionExceptions`parameter it allows you to load the specified dll files, even if the `PreventDlInjection`is set to true.

Example

```
PreventDllInjectionExceptions=<inject.dll>;<injectme2.dll>
```

NOTE Please use this parameter only when `PreventDllInjection=1.`

LargeAddressAware Parameter

You can use the `LargeAddressAware`parameter to capture applications which install on both 32-bit and 64-bit operating systems.

ThinApp can capture applications which install both 32 bit and 64 bit application and components together. It is possible that 32 version of application launches 64 bit version. In rare cases with ThinApp, the 64 bit application might fail to start from 32 bit application. To solve this problem set a package.ini parameter `LargeAddressAware` to 1 in the `package.ini` file and rebuild the package.

```
[BuildOptions]
LargeAddressAware=1
```

PermittedComputers Parameter

When captured application is started, the `PermittedComputers` parameter verifies whether the computer is a member of a specified Active Directory group. If the computer is not a member of Active Directory group, Thinapp does not start the application.

This parameter can be used as:

```
PermittedComputers=xpsystemgroup;win7systems
```

The Package.ini entry for application PDC overrides the global settings and all entry points will inherit these settings from PDC. For example:

```
[BuildOptions]
PermittedComputers=OfficeComputers

[Microsoft Office 2010.dat]
PermittedComputers=xpsystemgroup;

[Microsoft Word 2010.exe]
...
...
[Microsoft excel 2010.exe]
...
...
```

In the above example, `PermittedComputers` settings for [Microsoft Office 2010.dat] will override the global option provided in `[BuildOptions]` as this file is the PDC. Word and Excel applications will inherit settings from [Microsoft Office 2010.dat].

PermittedComputersAccessDeniedMsg Parameter

You can use the `PermittedComputersAccessDeniedMsg` parameter to display an error message to the user, the error message is displayed only if the system is not a member of `PermittedComputers`. However, this error message disappears automatically after few seconds.

PermittedComputersOfflineAccess Parameter

When you enable the `PermittedComputersOfflineAccess` parameter it prevents those computers that are part of permitted computer group in domain controller from accessing the virtual application in offline mode. When the parameter is not enabled permitted computers continue to use the virtual application.

IgnoreDDEMessages Parameter

The `IgnoreDDEMessages` parameter is automatically set for virtualized Internet Explorer 6 packages. When this parameter is set the application ignores any DDE messages that the application receives. The `IgnoreDDEMessages` can be used to ensure that the Document Open operations always start in the native application and not in the virtual application.

Locating the ThinApp Sandbox 23

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.

Search Order for the Sandbox

During startup of the captured application, ThinApp searches for an existing sandbox in specific locations and in a specific order.

ThinApp uses the first sandbox it detects. If ThinApp cannot find an existing sandbox, ThinApp creates one according to certain environment variable and parameter settings. Review the search order and sandbox creation logic before changing the placement of the sandbox.

Only one computer can use a shared sandbox at a time. 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.

Search Order Variables

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 .

- *sandbox_path* is Z:\sandboxes

  The `SandboxPath` parameter in the `Package.ini` file determines the path. See .

- *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.

Environment Variables

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

%<sandbox_name>_SANDBOX_DIR%	This environment variable changes the sandbox location for specific applications on the computer.

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

%THINSTALL_SANDBOX_DIR%

This environment variable changes the location of all sandboxes on a computer.

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

When ThinApp Does Not Detect an Environment Variable

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` parameter in `Package.ini` 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.

Controlling the Sandbox Location 24

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.

You can store the sandbox on a network or on a portable drive.

- Store the Sandbox on the Network on page 83

  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 so that users can log in to any machine and keep their application settings.

- Store the Sandbox on a Portable Device on page 84

  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.

- Store the Sandbox in a Thinstall Directory on a USB Drive at the Same Level as the Executable File on page 84

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

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 so that users can log in to any machine and keep their application settings.

Procedure

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
```

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.

Procedure

1 Open the `Package.ini` file.

2 Under the `SandboxName` parameter, set the `SandboxPath` parameter to the directory on a USB drive where the executable file resides.

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

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

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

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

Prerequisites

If the *%THINSTALL_SANDBOX_DIR%* or *%<sandbox_name>_SANDBOX_DIR%* environment variables are set, unset the variables.

Procedure

1 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.

2 (Optional) 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 25

ThinApp stores the sandbox using a file structure almost identical to the build project structure. ThinApp uses macro names for shell folder locations, such as `%AppData%`, instead of hard coded paths.

This structure enables the sandbox to migrate to different computers dynamically when the application runs from new locations.

The sandbox contains the following registry files:

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

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

You can make changes to files in the sandbox under specific conditions, as described in these topics.

- Making Changes to the Sandbox on page 85

  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.

- Listing Virtual Registry Contents with vregtool on page 86

  Because the sandbox contains the modifications to the registry, you might need the vregtool utility to view modified virtual registry changes.

Making Changes to the Sandbox

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.

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.

If a 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 26

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.

Using the Command Line to Create Snapshots

The `snapshot.exe` utility creates a snapshot of a computer file system and registry and creates a ThinApp project from two previously captured snapshots.

You do not have to start the snapshot.exe utility directly because the Setup Capture wizard starts it.

Only advanced users and system integrators who are building ThinApp capability into other platforms might make direct use of this utility.

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

- File information for all local drives

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

- `HKEY_LOCAL_MACHINE` and `HKEY_USERS` registry trees

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

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

Methods of Using the Snapshot.exe Utility

After you have created a snapshot of a machine state, you can use the utility to create a template Package.ini file and use that file to create a ThinApp project. You can also use the utility to display the contents of a snapshot file.

These steps take you from capturing a snapshot through creating a ThinApp project.

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
]
```

Example: Create Machine Snapshot

Use the following example to create a machine snapshot.

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

Options

The options specify the directories or subkeys in the snapshot.

Table 26-1. Snapshot Directories and Subkeys

Option	Description
`–Config ConfigFile.ini`	Specifies directories or registry subkeys to exclude during snapshot creation. If you do not specify a configuration file, ThinApp uses the `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
```

Example: Using Snapshot Files to Create a Package.ini Template

ThinApp requires all of the parameters.

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

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]
```

Example:

`–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.

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

Displaying the Contents of a Snapshot File

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

Usage

```
snapshot.exe SnapshotFileName.snapshot –Print
```

Example: Printing Content of a Snapshot File

ThinApp requires all of the parameters.

```
Snapshot Start.snapshot ––Print
```

Sample snapshot.exe Commands

These sample commands can be used with the `snapshot.exe` utility.

The parameters are not case-sensitive.

The commands are wrapped in the Command column because of space constraints.

Table 26-2. 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 for packaging a large number of applications or automate ThinApp project creation. The typical location of the `snapshot.exe` utility is `C:\Program Files\VMware\VMware ThinApp\snapshot.exe`.

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

Procedure

1 At the command line, run`snapshot.exe` without adding any parameters.

The help menu for the `snapshot.exe` utility appears.

2 Use the commands in the menu to create the ThinApp project.

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

4 (Optional) To generate multiple projects with different configurations, reuse the original file and repeat the procedure.

Index

Q

R

S