# Spoon Studio User Guide

# Legal Notices

This section specifies legal notices as applicable to this product.

## Disclaimer

To the maximum extent permitted by applicable law, Code Systems Corporation provides this product AS IS AND WITH ALL FAULTS, and disclaims all warranties and conditions, either express, implied or statutory, including, but not limited to, any implied warranties or conditions of MERCHANTABILITY, of fitness for a particular purpose, of accuracy or completeness, of results, and of lack of negligence, all with regard to this product. The entire risk as to the quality of or arising out of the use of this product remains with you.

This product may contain technological defects and omissions, typographic errors, and technical inaccuracies. Code Systems Corporation may modify this product at any time.

To the maximum extent permitted by applicable law, in no event shall Code Systems Corporation be liable for any special, incidental, indirect, or consequential damages whatsoever (including, but not limited to, damages for loss of profits or confidential or other information, for business interruption, for personal injury, for loss of privacy, for failure to meet any duty including of good faith or of reasonable care, for negligence, and for any other pecuniary or other loss whatsoever) arising out of or in any way related to the use of or inability to use this product, the provision of or failure to provide Support Services, or otherwise in connection with any aspect of this product, even in the event of the fault, tort (including negligence), strict liability, breach of contract or breach of warranty of Code Systems Corporation, and even if Code Systems Corporation has been advised of the possibility of such damages.

Code Systems Corporation's cumulative liability to you or any other party for any loss of damages resulting from any claims, demands, or actions arising out of or relating to this product shall not exceed the larger of the license fee paid to CODE SYSTEMS CORPORATION for the use of this product and U.S. $5.00.

## Trademarks

Spoon, Xenocode Postbuild, and Spoon Studio are trademarks and/or registered trademarks of Code Systems Corporation.

ZENworks Application Virtualization is a trademark of Novell, Inc.

Microsoft, Windows, SQL Server, .NET, and .NET Framework are trademarks of Microsoft Corporation.

All other trademarks are the property of their respective owner.

# Overview

Thank you for using Spoon Studio!

This product will allow you to convert your Windows, .NET, Java, AIR, Flash, Shockwave, or other Windows-compatible application into a self-contained virtual application that can be streamed from the web and run instantly on an end-user device. Unlike traditional deployment methods, virtual applications do not require reboots, administrative privileges, or separate setup steps for external components and runtimes. Virtual applications are isolated from other system applications, preventing DLL conflicts and other deployment nightmares.

This guide explains how to use Spoon Studio to create your own virtual applications and begin enjoying the benefits of this next-generation deployment technology.

## What is a virtual application?

A virtual application is a virtual machine image pre-configured with all of the files, registry data, settings, components, runtimes, and other dependencies required for a specific application to execute immediately. Virtual applications allow application publishers and IT administrators to significantly reduce the costs and complexity associated with development, setup, configuration, deployment, and maintenance of software applications.

For example, a publisher of an application based on the Microsoft .NET Framework or Java runtime engine might create a virtual application combining the application with the required runtime engine. An end-user can run this application immediately, even if the user has not installed the required runtime engine, or has an incompatible runtime engine installed. This improves both the user experience and reduces test and support complexity associated with deploying the application.

Because virtual applications run in isolated execution environments, it is possible to simultaneously execute multiple applications which would otherwise interfere with one another. For example, applications which overwrite system DLLs or require different runtime engine versions can all run simultaneously on a single host device. As an additional advantage, virtual applications can provide access to internal virtualized copies of privileged system resources, allowing unprivileged users to directly execute many applications without security exceptions or irritating Vista UAC prompts.

Unlike other virtualization systems, Spoon virtual application technology:

- Does not require any "player" software or separate installation: Spoon virtual applications are executable files that run immediately on the end-user machine without changes to system infrastructure.
- Does not incur significant processing or filesystem overhead: Spoon low-overhead virtualization technology allows applications to run with essentially the same performance characteristics as native executables.
- Does not require any operating system to be installed onto the virtual application: Spoon virtual apps provide all required virtualized operating system functionality within the internal virtual environment.

# Spoon Studio features overview

Use Spoon Studio to:

- Create virtual applications that can be streamed from the web: Eliminate long downloads and installs, and run virtual apps from any desktop with broadband Internet access.
- Create an application as a single executable that runs immediately: Package all application files, registry settings, runtimes, and components into a single executable that runs immediately.
- Run Java and .NET without separate runtime installations: Java and/or .NET-based applications run immediately, with no separate installation steps or runtime versioning conflicts.
- Improve desktop security: Execute your applications without granting administrative permissions to end-users. Stabilize desktop images by deploying applications in Spoon sandboxed virtual environments.
- Eliminate third-party setup dependencies: Integrate third-party components, COM/VB controls, and content viewers such as Acrobat, Flash, and Shockwave, directly into your application.
- Eliminate Vista UAC prompts and compatibility errors: Deploy Spoon virtual apps regardless of access to privileged system resources, relieving users of annoying Vista UAC prompts.
- Leverage Terminal Services and Citrix investments: By isolating applications from global resource areas, Spoon virtual application technology allows non-compliant applications to function properly in Terminal Server and Citrix environments.
- Deploy instantly on USB drives: Improve mobile worker productivity by placing your Spoon virtual application onto a USB flash-memory drive. Run your application immediately on remote PCs, with no installation steps, administrative privileges or driver installations.
- Dramatically reduce test and support costs: Eliminate versioning conflicts, dependencies, and "DLL hell". Reduce test complexity and eliminate support requests associated with dependency installation and inter-application resource conflicts.

# Do Spoon virtual applications require any device drivers?

No. Spoon virtualization takes place entirely in user-mode, so no device drivers are installed or required.

# How is Spoon virtualization different from hardware virtualization?

Unlike hardware virtualization systems such as Microsoft Virtual PC and VMware, Spoon virtualizes only those operating system features required for application execution. This allows virtualized applications to operate extremely efficiently, with essentially the same performance characteristics as native executables.

Advantages of Spoon virtualization over hardware virtualization include:

- Optimal performance: Spoon virtual apps execute at essentially the same speed as applications running natively against the host hardware, with only a small additional memory footprint. By contrast, applications running within hardware-virtualized environments experience significant slowdowns and impose a large memory footprint because the virtual machine must include an entire virtualized host operating system.
- Dramatically reduced application size: Spoon virtual apps require a disk footprint proportional only to the size of the virtualized application, data, and included components. As a result, Spoon virtual apps are typically small enough to be conveniently and quickly downloaded by end-users. Hardware virtualization requires an entire host operating system image, including many basic subsystems that are already present on the end-user device. Each virtual machine may occupy several gigabytes of storage.
- Ability to run multiple virtual applications: Because of low-overhead characteristics, it is possible to run multiple simultaneous Spoon virtual environments per processor. Conversely, due to the high overhead of hardware virtualization, only a very small number of hardware-virtualized environments per processor can be run.
- Reduced licensing costs: Spoon does not require the purchase of separate operating system licenses to use a virtual application. Hardware virtualization systems require a host operating system in order function, possibly imposing additional licensing costs and restrictions.

Hardware virtualization may be appropriate in certain specialized scenarios, including:

- Non-Windows operating systems: Spoon virtual apps run only on the Windows operating system. Hardware virtualization can execute

any operating system compatible with the underlying virtualized hardware, such as Linux.
- Kernel mode virtualization: The Spoon Virtual OS only virtualizes user-mode operating system features, whereas hardware virtualization systems emulate the entire OS stack, including kernel mode components. Applications requiring device drivers or other non-user-mode software may require a hardware-virtualized environment to function properly.

You should carefully evaluate the advantages and disadvantages of different virtualization approaches before deciding on a technology to adopt for your deployment scenario.

## What platforms are supported?

Studio supports the following platforms for virtual application build, snapshotting, and execution:

- Windows XP Professional
- Windows Embedded XP
- Windows 2000 Professional
- Windows 2000 Server
- Windows Server 2003, all editions
- Windows Vista, all editions
- Windows Server 2008, all editions
- Windows 7

Studio supports these operating systems running within VMware and Microsoft hardware virtualization and hypervisor environments.

Studio also has limited support for the Windows Preinstallation Environment (WinPE), though certain applications depending on operating system features unavailable in WinPE may not function properly.

Studio creates 32- and 64-bit executables. Both 32-bit (under 32-bit mode) and 64-bit executables can be run on x64-based platforms.

## What applications can be virtualized using Spoon Studio?

Spoon Studio and the Spoon virtualization engine support most major Windows desktop applications. However, certain applications- by their nature- are unsuitable for virtualization using Spoon's user-mode virtualization technology. These include application features which contain or directly depend on interaction with specialized kernel-mode device drivers or other kernel-mode extensions; operating system components and extensions; anti-virus applications; and kernel event filtering, monitoring, and intrusion detection applications.

Spoon applications are compatible with most major anti-virus, runtime, and security packages currently available.

# Getting Started

This section describes the system requirements for installing and running Spoon Studio, provides an overview of the Studio user interface, and walks you through the basic steps of creating a virtual application.

## System requirements

Spoon Studio requires a Windows XP, Windows 2000 edition, or higher operating system. The Studio graphical interface assumes a screen resolution of at least 800×600, although a screen resolution of at least 1024×768 is highly recommended.

## Control Panel Overview

The Studio control panel allows you to configure your virtual application filesystem and registry, embed external runtimes and components, take snapshots of the application, and create Spoon Virtual Machine (**SVM**) files. The primary interface consists of a ribbon bar and several panes grouped by a functional area.

Located above the ribbon bar are:

- The **Start menu** button- located in the circle on the top left of the window- allows virtual application configurations to be imported, opened, applied, saved, and closed.
- The **Options** bar provides Studio interface customization options, and the ability to install license certificates
- The **help** bar provides access to the Studio documentation and knowledge base, including a searchable version of this document.

The ribbon bar accesses common Studio features:

- The **Virtual Application** tab provides access to the snapshot and build features, as well as output configuration options such as the startup file, output directory, and diagnostic-mode selection.
- The **Runtimes** tab provides a selection of auto-configurable runtime engines which can be embedded into your application with a single click. These include .NET Framework, Java, Flash, and Shockwave runtimes.
- The **Advanced** tab provides advanced Studio functions such as Profiling and Platform Merge

Functions in the main panel are accessed by clicking the appropriate buttons along the left side of the interface:

- The **Start** panel displays the latest Studio news, including updates, available licenses, and usage suggestions.
- The **Filesystem** panel displays the application virtual filesystem, and allows adding and removing virtual files and directories.
- The **Registry** panel displays the application virtual registry, and allows adding and removing virtual registry keys and data values.
- The **Settings** panel allows configuration of virtual application metadata, startup image, and process configuration options.
- The **Components** panel allows layering of external virtual application components, such as toolbars and optional features.
- The **Setup** panel allows configuration of MSI setup package and shell integration options.
- The **Expiration** panel allows configuration of application expiration options.

Note: Studio users are individually responsible for assuring compliance with licensing for any third-party redistributable components included using virtualization.

# Methods of creating virtual applications

Studio offers three ways to create and configure virtualized applications. The best method in a given scenario depends on the nature of the application to be virtualized.

- Use an application template: Studio includes templates for popular applications which can be built and customized using a guided, step-by-step process. This method is recommended for first-time users of Studio. (This method only applicable in certain editions of Studio)
- Install via application snapshot: Snapshotting captures system state before and after an application is installed and automatically configures virtual application settings based on observed system changes. This method is ideal for virtualizing off-the-shelf applications (Refer to the section "Snapshotting applications" for more information on this method).
- Manually configure an application: This method is most often used by developers virtualizing internally developed applications. Manual configuration requires a high degree of technical knowledge but allows the maximum amount of control over virtual application settings (Refer to the sub-section "Manually configuring a simple virtual application" in the "Walkthroughs" section for more information on this method).

All methods allow additional configuration and customization once the initial virtual application configuration has been constructed.

For information on current available editions of Spoon Studio, visit the *Studio* section of Spoon.net.

# Creating your first virtual application

(Only applicable in certain editions of Studio)

Studio includes automated virtual application configuration wizards for certain popular software applications. Spoon recommends that first-time users begin by building one of these auto-configurable virtual applications using the Studio Configuration Wizard.

To build an auto-configured application:

- Open the Studio Configuration Wizard. The wizard is displayed on program startup, or can be opened by pressing the **Configuration Wizard** button on the **Virtual Application** ribbon bar.
- Click on the box labeled **Build a virtual application from a template**.
- Select an application to virtualize from the **Application** dropdown. Some applications may require download of additional configuration information or source application media.
- Follow the wizard steps to construct the virtual application.

After completing the wizard, the virtual application configuration will remain loaded in the Studio interface. This allows the configuration settings generated by the wizard to be inspected and additional customization to be performed (Refer to the following sections "Configuring virtual applications" and "Customizing virtual applications" for more information on configuration and customization).

Congratulations on building your first virtual application!

# Configuring virtual applications

Virtualization allows application deployment to be dramatically simplified by allowing files, registry settings, components, and other application dependencies to be directly embedded into the application executable. Use of Spoon virtualization reduces setup complexity, prevents DLL collisions, and allows applications to simulate the use of privileged disk and registry resources without requiring administrative privileges on the host machine.

# Snapshotting Applications

Most commercial applications require complex combinations of filesystem and registry entries in order to function properly. In order to facilitate virtualization of these applications, Studio can *snapshot* application installations and automatically configure them based on modifications made to the host system during application setup.

## Snapshotting

Snapshotting uses "before" and "after" images of the host machine to determine the virtual application configuration:

- "Before" snapshot: Taken prior to installing the application. This captures the state of the host device without the target application installed.
- "After" snapshot: Taken after installing the application. This captures all changes to the host device during application installation. Studio then computes the changes, or *deltas*, between the before and after snapshots, and inserts these changes into the configuration.

To use the snapshot feature:

- Prepare the host device by either removing the target application and all dependencies, or copying Studio onto a "clean" machine.
- Click on the **Virtual Application** tab on the ribbon bar and click **Capture Before**. This captures the "before" snapshot image. Snapshotting iterates through the filesystem and registry, and therefore may take several minutes to complete.
- (Optional) Spoon recommends you save the "before" snapshot before continuing. This allows you to skip this step when snapshotting subsequent applications from the same clean machine image. To save the snapshot, click on the down arrow underneath the **Capture Before** button and select **Save Snapshot**. Note that while Studio automatically saves the most recent captured "before" snapshot, this snapshot is reset once the Capture and Diff process is complete.
- Install your application along with other files, settings, runtimes, and components you wish to include in the virtual application (Refer to the following sub-section "Adding runtimes and components" for more information on additional configuration). If the application setup requests a reboot, be sure to save the "before" snapshot and then proceed with the reboot.
- On the **Virtual Application** tab on the ribbon bar, click **Capture and Diff**. This captures the "after" snapshot, computes the deltas between the two snapshots, and populates the virtual application with the delta entries.
- (Optional) Review the filesystem and registry entries, and remove any files or settings which are not required for proper execution of your virtual application. Removing unused entries will reduce virtual application size, but be sure to avoid accidental removal of required resources, as it will cause your virtual application to no longer function properly.

## Saving snapshots

In many cases, the desired "before" snapshot remains fixed while many "after" snapshots are taken. Studio allows you to save the "before" snapshot image so that the snapshot does not need to be re-captured each time. Because snapshotting may take several minutes, this significantly reduces the time required to build virtual applications in this scenario.

To save the "before" snapshot, click on the down arrow underneath the **Capture Before** button on the **Virtual Application** ribbon bar and select **Save Snapshot** from the dropdown menu. Select an appropriate filename and location and press **Save**. Similarly, to load a saved snapshot, select the **Load Snapshot** menu item and navigate to the saved snapshot file. To clear the current "before" snapshot image, select the **Clear Snapshot** menu item.

# Adding runtimes and components

Many components and runtime systems consist of large, complex sets of filesystem entries and registry settings. To simplify configuration of the most common components, Studio contains a collection of pre-configured component settings which can be added to your virtual application with a single click.

Additional runtimes and components should be added to the virtual application during the snapshot process *before* the "After" snapshot has been taken.

To add a runtime or component:

- Click on the **Runtimes** tab on the ribbon bar.
- Click on the appropriate runtime or component to select it for inclusion. Selected components are indicated with a highlighted button. To remove a component, click on the button again. This toggles the component inclusion state.

For example, if your application is a .NET Framework 4.0 application, then selecting the **.NET Framework 4.0** component will allow your executable to run on machines without the .NET Framework installed.

Note: Depending on the size of the component, selecting a component for inclusion can significantly increase the size of the resulting executable. Therefore, you should only select components which are required for proper execution of your application.

Note: You are responsible for assuring compliance with licensing for any third-party redistributable components included in your virtualized application.

## Configuring the Java runtime

Studio provides specialized support for the Java runtime. If your application is based on Java runtime, press the **Sun Java Runtime** button on the **Runtimes** ribbon bar. This displays the Java configuration menu.

Select the appropriate version of the Java runtime from the **Java runtime version** dropdown. If you are deploying your application as a set of .class files, then select the **Class** option from the **Startup type** dropdown; if you are deploying within a .jar file, select the **Jar** option. Enter the startup class name or Jar name in the appropriate textbox, along with any additional Java runtime options.

# Loading and saving configurations

Once you have configured your virtual application, and taken your "After" snapshot, you will likely want to save the configuration for future use or modification. It is important to save the virtual application snapshot in its original state in case errors are introduced during virtual application customization and optimization. This also allows the application to be tested and modified without the need to re-snapshot after each iteration.

To save a configuration:

- Click on the **Start button** menu and select **Save Configuration As...**
- Select a filename and location and click **Save**. This saves the virtual application configuration file. By default, configuration files use the extension **.xappl**.

Note: Configuration files do not store the *contents* of virtual filesystem files. The configuration file specifies only the *source path* for each virtual filesystem entry. The source file must exist at build time or the virtual application will not build successfully.

Studio automatically stores source file locations as paths relative to the location of the saved **XAPPL** file, in the same directory as the **XAPPL** file (the folder marked **Files**).

Note: Studio users can create their own default settings for application building by saving a **XAPPL** file with a few customization settings (E.g. changing the default sandbox location) and no associated application, then loading this **XAPPL** file (and saving as a new file) before beginning the snapshot process in a new project.

# Specifying a startup file

The virtual filesystem may contain a large number of executable files (such as .exe, .cmd, and .java) and viewable file formats (such as .html and .swf). However, your virtual application is consolidated into a single executable. It is therefore necessary for the virtual application designer to indicate a *startup file* - the executable or viewable file which is opened when the user executes the virtual application.

A startup file should be specified *after* the application has been installed and configured with runtimes, and the "After" snapshot has been taken.

To select the startup file:

- Click on the **Virtual Application** tab on the ribbon bar.
- Click on the **Startup File** dropdown list. This displays a list of all files in the virtual filesystem.
- Select the file to be used as the startup file, or navigate to the desired startup file in the virtual filesystem display, right-click the file, and select **Set as Startup File**.

Files located on the host device (outside of the virtual filesystem) may also be used as startup files. To select a file on the host device as the startup file, enter the full path to the desired startup file in the **Startup File** text box. Remember to use well-known root folder variables such as **@WINDIR@** and **@PROGRAMFILES@** as the root of the full path to ensure that the startup file can be properly located on all base operating systems.

Note: While any file can be selected as the startup file, you should only select a file which is executable or viewable. Selecting a file which cannot be opened will cause an error when the virtual application is started.

# Specifying multiple startup files (Jukeboxing)

In some situations, a virtual application may want to expose multiple startup files. For example, when using a virtualized office productivity suite, one may want to launch either the word processor, spreadsheet, or presentation component of the suite, while still deploying a single executable file.

Studio enables this scenario by allowing multiple entry points into the virtual application to be triggered based on a command-line argument to the virtual application executable. For example, in the office suite scenario described above, one might use the command line argument **office word** to trigger the word processor and **office spreadsheet** to trigger the spreadsheet (Refer to the sub-section "Building OpenOffice via snapshot process" in the "Walkthroughs" section for an example of Jukeboxing during the OpenOffice build process).

To specify multiple startup files:

- Click the **Multiple** button next to the **Startup File** textbox on the **Virtual Application** ribbon bar. This displays the **Startup Files** selection dialog.
- Click on the **File** column on the first empty row in the startup file list and select the desired file from the dropdown list. Files located on the host device (outside of the virtual filesystem) may also be used as startup files. To select a file on the host device as the startup file, enter the full path to the desired startup file in the **Startup File** text box.
- Enter the desired command line arguments, if any, in the **Command Line** column.
- Enter the desired command line trigger in the **Trigger** column. For example, in the command line **office word**, the trigger would be **word**.
- Check the **Auto Start** checkbox if you want the startup file to always be automatically launched on virtual application startup.
- After adding a new startup file, hit **Enter** in order to save

Note: When specifying a startup file located outside of the virtual filesystem, remember to use well-known root folder variables such as **@WINDIR@** and **@PROGRAMFILES@** as the root of the full path to ensure that the startup file can be properly located on all base operating systems.

Note: The **Auto Start** flag can be specified for multiple startup files to automatically launch multiple applications that are typically used together in a single session (also known as "shotgunning").

# Editing the virtual filesystem

Studio allows you to embed a *virtual filesystem* into your executable. Embedded files are accessible by your Spoon-processed application as if they were present in the real filesystem. However, virtual files- unlike actual files on the host device- are not visible from and do not require changes to the host device. In particular, the use of virtual files does not require any security privileges on the host device, even if the virtual files reside in a privileged directory such as the Windows directory. Also, because virtual files are embedded in the application executable, shared DLLs embedded in the virtual filesystem will not interfere with or be overwritten by other applications on the host device.

To add virtual files:

- Click on the **Filesystem** button located on the left side of the Studio window.
- Using the view on the right, add the files and folders you wish to embed in the application executable. The **Application Directory** root folder represents the folder containing the virtual application binary on the executing device; the other root folders represent the corresponding folders on the host device.

Note: When running a virtual application on Windows 7, the **All Users Directory\Application Data** and **All Users Directory** root folders will map to the same folder at runtime. Ensure that the isolation settings for these folders are the same, preventing unpredictable behavior as one setting will override another.

## Virtualization Semantics

In the event of a collision between a file in the virtual filesystem and a file physically present on the host device, the file in the virtual filesystem takes precedence.

Folders may be virtualized in **Full**, **Merge**, **Write Copy**, or **Hide** mode:

- Full mode: Only files in the virtual filesystem will be visible to the application- even if a corresponding directory exists on the host device- and writes are redirected to the sandbox data area. Full mode is generally used when a complete level of virtual application isolation is desired.
- Merge mode: Files present in a virtual folder will be merged with files in the corresponding directory on the host machine, if such a directory exists. Writes to host files are passed through to the host device and writes to virtual files are redirected into the sandbox data area. Merge mode is generally used when some level of interaction with the host device is desired. For example, Merge mode might be used to allow the virtualized application to write to the host device's **My Documents** folder.
- Write Copy mode: Files present on the host device are visible to the virtual environment, but any modifications to folder contents are redirected to the sandbox data area. Write Copy mode is generally used when a virtual application needs to read from files already present on the host device but isolation of the host device is still desired.
- Hide mode: Files and folders with Hide isolation enabled will return a 'File Not Found' message to the application at runtime. This applies to both the virtual filesystem and the host file system. Hide mode is generally used when a file on the host machine may interfere with the application's ability to run properly. By adding the file or folder to the virtual package with Hide isolation enabled, the application will receive a 'File Not Found' message, even if the file or folder exists on the host machine.

Tip: To apply the selected isolation mode to all subfolders, right-click on the folder, choose **Isolation**, click on the checkbox for **Apply to subfolders**, and click **OK**.

## File Attributes

Files and folders may optionally be hidden from shell browse dialogs and other applications enumerating virtual directory contents. This is often used to prevent internal components and data files from being displayed to the user. To hide a file or folder, click on the checkbox in the **Hidden** column next to the desired file or folder.

Note: The hidden flag should not be confused with Hide isolation mode. Enabling the hidden flag only prevents a file or folder from being displayed in browse dialogs or from directory enumeration APIs. It does not prevent the application, and therefore potentially the end-user, from accessing the folder or file contents by direct binding. In order to prevent the file or folder from being found by the application, Hide isolation mode should be enabled.

Flagging files and folders as read-only will prevent the application from modifying the file or folder contents. To make a file or folder read-only, click on the checkbox in the **Read Only** column next to the desired file or folder.

## Sandbox Upgradeable Files

By default, Studio allows files in the virtual filesystem to be upgraded with patches (refer to the topic "Updating a Virtual Application" in the "Registering virtual applications in the Windows shell" section for more information on application updates). The **Upgradeable** flag will allow a specific file or folder to be patched in the sandbox at runtime. To automatically allow a file or group of files to be upgraded, leave the **Upgradeable** checkbox in the column next to the desired file or folder checked.

## Filesystem Compression

To reduce executable size, Studio can compress virtual filesystem contents. This typically reduces virtual application payload size by approximately 50%, but also prevents profiling and streaming of the app. By default, the **Compress Payload** option in the **Process Configuration** area of the **Settings** panel is unchecked. This option should remain unchecked during the build process in order to be able to profile and stream applications.

Note: Disabling payload compression may significantly increase the size of the virtual application binary.

# Editing the virtual registry

Studio allows you to embed a *virtual registry* into your executable. Embedded registry keys are accessible by your Spoon-processed application as if they were present in the real registry. Unlike data present in the registry on the host device, virtual registry keys and values are not visible from and do not require changes to the host device. In particular, the use of a virtual registry does not require any security privileges on the host device, even if the virtual registry entries reside in a privileged section of the registry, such as **HKEY_LOCAL_MACHINE**. Also, because virtual registry entries are embedded in the application executable, other applications are unable to disrupt application execution by inadvertent modification of registry entries required by the application.

To add virtual registry data:

- Click on the **Registry** button located on the left side of the Studio window.
- Using the view on the right, add the registry keys and values you wish to embed in the application executable. When adding blob data, enter the values in hexadecimal format.

The **Classes** root, **Current user** root, **Local machine**, and **Users** root folders correspond to the **HKEY_CLASSES_ROOT**, **HKEY_CURRENT_USER**, **HKEY_LOCAL_MACHINE**, and **HKEY_USERS** keys on the host machine.

Registry string values may include well-known root folder variables such as **@PROGRAMFILES@** and **@WINDIR@**.

## Virtualization Semantics

In the event of a collision between a key or value in the virtual filesystem and data present on the host device registry, information in the virtual registry takes precedence.

Keys may be virtualized in **Full, Merge,** or **Hide** mode:

- Full mode: Only values in the virtual registry will be visible to the application- even if a corresponding key exists on the host device- and writes are redirected to the user registry area.
- Merge mode: Values present in a virtual key will be merged with values in the corresponding key on the host machine, if such a key exists. Writes to host keys are passed through to the host registry and writes to virtual keys are redirected to the user registry area.
- Hide mode: Keys and values in the virtual registry or the corresponding host registry will not be found by the application at runtime.

Tip: To apply the selected isolation mode to all subkeys, right-click on the key, choose Isolation, click on the checkbox for Apply to subkeys, and click OK.

## Importing Registry Hive Files

Studio can import registry hive (.reg) files into the virtual registry. To import a .reg file, click the **Import** button in the **Registry** panel and select the registry hive file to be imported.

# Embedding a database engine

Studio allows SQL Server 2005 Express to be embedded directly into the executable file. This allows executables to run a SQL Server user-instance for a virtual .mdf database file.

## Selecting a database engine to embed

To select SQL Server 2005 Express:

- Click on the **Runtimes** button on the ribbon bar.
- Click on the dropdown next to the **Embed SQL Server Express** option. This displays a list of supported SQL Server Express engines.
- Select an engine which is appropriate for your database.

Selecting a database engine displays the database configuration dialog. Selecting the **Do not embed SQL Server Express** option disables this feature.

## Configuring your application to use an embedded database

The code displayed on the database configuration dialog provides an example of how to create a connection string for connecting to the user-instance database that loads an .mdf file from the virtual filesystem.

The virtualized SQL Server 2005 Express default instance name is stored in a special environment variable named **%SQLSPOON%**. A unique instance of this name is created at build time and is not user-configurable.

To embed a database file:

- Verify that the desired .mdf database file has been added to the virtual filesystem.
- Choose the .mdf database file to attach using the **AttachDBFilename** dropdown. This path is stored in the **%AttachDBFilename%** environment variable.

To manually modify this selection at a later time, click the **Environment Variables** button on the **Process Configuration** tab of the **Settings** pane.

Note: The .NET 2.0 Framework is automatically included when SQL Server Express 2005 engine is selected. SQL Server 2005 Express requires this component to be installed. If this component is removed, the application will only run on machines that have the .NET Framework 2.0 installed.

# Creating and using shared virtual components

In some scenarios, multiple virtual applications may share a common set of virtual machine configuration options. For example, multiple applications often share a common set of components or runtime engines; or system administrators may want to share a common set of configuration options (browser bookmarks, application settings, etc.) across a department or enterprise. Studio makes it easy to create, share, and consume virtual machine settings across multiple Spoon virtual applications using Spoon **SVM**-format virtual components.

To create a shared virtual component:

- Configure the virtual application settings exactly as in the case of a standard virtual application (i.e. using snapshotting, manual configuration, etc.).
- On the **Settings** pane, select **Component** from the **Project type** dropdown.
- Press the **Build** button.

In **Component** mode, the build process results in creation of an **SVM** file instead of an executable file. An **SVM** contains the virtual application settings and data payload. **SVMs** are similar to virtual executable outputs, except that **SVMs** do ***not*** contain the Spoon virtual machine runtime engine. Therefore, an **SVM** can only be used when combined as part of another virtual application.

To use an existing shared virtual component:

- Click on the **Components** button to navigate to the project components pane.
- If the component does not already appear in the components table, click **Import Components...**
- Select the **SVM** you wish to load into your project, and click **OK**. The **SVM** is then loaded into your project and the layer metadata is displayed in the **Components** list.
- Select the checkbox next to the desired component.

Project virtualization settings take precedence over virtualization settings in any loaded shared components.

To remove a shared virtual component from the project, select the component and click the **Remove Component** button.

## Sandbox merge

Sandbox merge allows sandbox content and settings generated during virtual application execution to be applied to a Studio configuration. Sandbox merge is an alternative to manual registry or filesystem configuration, and is particularly useful for applying additional customizations to existing virtual application configurations or configurations generated from a virtual application template.

To merge an existing sandbox into the active configuration:

- Click the **Sandbox Merge** button in the **Tools** section of the **Virtual Application** toolbar.
- Enter the path of the sandbox to be merged into the current configuration.
- Click **OK**.

For example, to customize the home page of the Firefox virtual application template:

- Use the **Configuration Wizard** to create a Firefox virtual application. (The wizard allows customization of the home page, but we will later use the sandbox merge feature to override the setting specified in the wizard.)
- Press **Build and Run** to launch the virtualized Firefox application.
- Using the Firefox interface, specify a new browser home page.
- Exit the Firefox virtual application.
- Press **Sandbox Merge** to display the sandbox merge dialog. The sandbox path will be pre-populated with the location of the Firefox virtual sandbox.
- Click **OK**.

The virtual application settings are updated with the configuration changes made during Firefox execution, including the updated browser home page.

# Virtual Application Customization

This section describes advanced virtual application customization options, such as executable metadata, startup images, command-line arguments, and process startup options.

## Selecting a project type

(Only applicable in certain editions of Studio)

Studio supports two project types:

- **Application**: A virtual application project produces an executable file output (**.exe** file) that can be run directly from the operating system. Application output mode is appropriate for most users and is the default selection.
- **Component**: A component project produces an **SVM** (.svm file). **SVM** is a Spoon file format encoding all virtual application configuration and content into a single binary file. **SVM**s cannot be executed directly from the operating system. **SVM**s are used to exchange virtual application and component data between multiple virtual applications. For example, component output mode is used to submit components into the Spoon Component Gallery.

Note: In order to create **SVM**s for use in streaming applications on Spoon Server, the project type must be set to component.

To set the project type, press the **Settings** button and select the appropriate option from the **Project type** dropdown.

## Customizing executable metadata

Executable metadata provides external applications such as the Windows shell with information regarding the application's identity, publisher, version, preferred display icon, and description. Metadata may be viewed and edited by clicking on the **Properties** tab of the **Settings** pane.

### Standard metadata

Standard metadata includes information such as the product title, publisher, description, icon, web site URL, and version. By default, Studio will apply metadata inherited from the virtual application startup file to the output virtual application executable. However, in some instances, it may be desirable to override the metadata.

To manually override executable metadata:

- Uncheck the **Inherit properties** option
- Enter the desired metadata in the appropriate fields in the **Properties** area.

To revert to the default inheritance behavior, recheck the **Inherit properties** option.

## Custom metadata

In addition to standard Windows shell metadata, Studio allows introduction of custom metadata into the output executable. Custom metadata may be used by specialized external executable viewer applications, inventory scanners, and other asset and licensing management systems.

To add or modify custom metadata:

- Click the **Custom Metadata...** button. This displays the **Custom Metadata** dialog.
- Enter the custom metadata property names and values into the dialog.  Only string-type custom metadata values are supported.

For information on programmatically reading custom executable metadata, please consult the Microsoft Windows Software Development Kit.

# Adding a startup image

Studio allows you to specify a startup "splash" image to be displayed during virtual application startup. Startup images improve application branding and are especially useful if your application requires several seconds to initialize.

To add a startup image:

- Click on the **Startup Settings** tab of the **Settings** pane
- Click the **Select...** button next to the **Splash Image** textbox.
- Navigate to a BMP-format image to use as the startup graphic, and click **Open**.

If you wish to remove the current startup image, click the **Reset** button.

## Transparency keying

Transparency keying allows the startup image to contain transparent regions. Transparencies can improve the visual effectiveness of your startup image.

To select the transparency key color:

- Click the **Select...** button next to the **Transparency key** label.  This displays the transparency key selection dialog.
- Select the color which represents transparent regions in the startup image and click **OK**.

## Previewing the startup image

To preview the startup image, press the **Preview** button.  Previewing is particularly useful to assure that the transparency key has been set properly.

# Compiling and adding a startup shim

Studio allows you to specify a compiled startup shim that is invoked prior to the startup file. Startup shims are used to perform customized licensing checks or other initialization tasks. The compiled startup shim must conform to the expected Studio interface in order to be validated.

The startup shim must be compiled with an **OnInitialize** method.

**C-style startup shim signature**

```
typedef BOOL (__stdcall *FnOnInititialize) (LPCWSTR pwcsInitilizationToken);
```

The return value indicates whether virtual machine execution should proceed.

Methods are to be acquired via **::LoadLibrary** followed by **::GetProcAddress** calls.

**Example**

```
LPCWSTR pwcsInitToken = "VendorSpecificToken";
HMODULE hShim = ::LoadLibrary("Shim.dll");
FnOnInititialize fnOnInit = (FnOnInititialize)::GetProcAddress(hShim, "OnInitialize");
BOOL fResult = fnOnInit(pwcsInitToken);
```

To add a compiled startup shim DLL:

- Click on the **Startup Settings** tab of the **Settings** pane
- Click the **Select...** button next to the **Startup Shim DLL** field.
- Navigate to a DLL-format file to use as the startup shim, and click **Open**.

Use the **OnInitialize parameter** field to specify parameters for your startup shim.

If you wish to remove the current startup shim, click the **Reset** button.

# Process configuration options

Studio provides several options that control the startup of the primary and child processes. Options can be accessed by clicking the **Settings** button, then clicking the **Process Configuration** tab.

## Command line arguments

By default, command line arguments specified by the user upon virtual application execution are passed to the virtual application startup executable. However, it is possible to override this behavior and specify a fixed set of command line arguments to be passed to the startup executable. For example, one can use this option to specify Java virtual machine behavior.

To specify an explicit command-line:

- Click on the **Settings** button
- Click on the **Process Configuration** tab
- Enter the command-line arguments in the **Command line** textbox.

Note that these arguments override any arguments that might be specified by the end-user.

## Working directory

The working directory setting determines the active directory at the time the virtual application process is launched.

The **Use startup file directory** option sets the working directory to the directory of the virtual application startup file. In the case of a jukeboxed application, the working directory is set to the directory of the startup file specified on the jukebox command line.

The **Use current directory** option sets the working directory to the directory from which the virtual application is launched.

The **Use specified path** option allows an explicit working directory to be specified.  The working directory specification can include environment and well-known root folder variables.

By default, the working directory is set to the directory of the startup file.

## Application type

Windows applications may execute in either the GUI- or console-mode subsystems.  If you have selected an executable startup file, Studio will automatically configure the virtual application to execute in the same subsystem as the startup file. However, if you have selected a non-executable startup file, it may be necessary for you to manually override the application type. Most applications execute in the GUI subsystem.

To override the application type, select the appropriate mode from the **Application type** dropdown in the **Process Configuration** section of the **Settings** panel. The **Inherit** mode sets the application type based on the type of the startup file, if possible.

## Target architecture

The **Target architecture** option matches the structure of the virtual environment to the desired host process architecture.

**x86:** Use this option for applications that were snapshotted on x86 systems. This option maps the **Program Files** directory to **C:\Program Files** on x86 systems or to **C:\Program Files (x86)** on x64 systems. .NET applications always run as 32-bit applications.

**x64:** Use this option for applications that were snapshotted on x64 systems. This option maps the **Program Files** directory to **C:\Program Files** on x64 systems. The **Program Files (x86)** directory is mapped to **C:\Program Files** on x86 systems and **C:\Program Files (x86)** on x64 systems. .NET applications run as 32-bit applications on x86 systems and 64-bit applications on x64 systems.

**Any CPU:** Use this option for .NET applications that are compiled to run on any CPU architecture. This option maps the **Program Files** directory to **C:\Program Files** on x86 systems and **C:\Program Files** on x64 systems. .NET applications run as 32-bit applications on x86 systems and 64-bit applications on x64 systems.

## Environment variables

Some applications may depend on the presence of certain Windows environment variables in order to function properly. Studio allows virtualization of environment variables to support such applications.

To add or modify virtual environment variables:

- Click the **Environment Variables...** button. This displays the **Environment Variables** dialog.
- Enter environment variable names and values into the environment variable list.

Most virtual environment variables overwrite any environment variables defined in the host environment. However, the special **PATH** and **PATHEXT** environment variables are always merged with the corresponding host environment variables.

Environment variables are automatically captured and merged during the snapshotting delta process. Therefore, it is generally unnecessary to manually configure environment variable settings.

## Virtual services

Windows services are specialized applications that execute in the background and are typically responsible for providing system services such as database services, network traffic handling, web request processing, and other server functionality. Many applications install and require specific services in order to function properly.

Studio fully supports virtualization of Windows services. To view or modify virtual service settings, press the **Virtual Services...** button. This displays the **Virtual Services** dialog.

The **Name** field specifies the internal name of the virtual service. For example, the Windows web server would use the name **w3svc**.

The **Friendly Name** field specifies the full display name of the service displayed to end users. For example, the Windows web server friendly name is **World Wide Web Publishing Service.**

The **Command Line** field specifies the full command line (including the service executable name and any parameters) used to launch the service.

The **Auto Start** flag indicates whether a virtual service automatically starts during virtual application startup, or whether the service must be launched manually or by the virtualized service control manager.

The **Keep Alive** flag indicates whether the virtual service process is automatically terminated when the primary application executable terminates, or whether the service (and, therefore, the host virtual application executable) continues to run until the service terminates itself.

Service installation and settings are automatically captured during the snapshotting process. Therefore, it is generally unnecessary to manually configure virtual service settings. The primary exception is the case of virtualized applications intended to run as background worker services (for example, virtualized web servers); in this case, it is often required to explicitly enable the **Keep Alive** option.

## Child processes

Some applications spawn new *child processes* during the course of their execution. Depending on the virtual application context, it may be preferable for such child processes to be created either within the virtual application, or outside of the virtual application in the host operating system.

Child processes include processes spawned to service COM local server requests.

Note: Child processes created outside of the virtual application will not have access to virtualized filesystem or registry contents. However, these processes will be able to access or modify host operating system contents, even if this would otherwise be forbidden by the virtual application configuration.

By default, child processes are created within the virtual application. To force child processes to be created outside of the virtual application, uncheck the **Spawn child process within virtualized environment** option. COM local servers can optionally be created within the virtual application context. To force COM local servers to be created outside of the virtual application, uncheck the **Spawn COM servers with virtualized environment** option.

Exceptions to the child process virtualization behavior specified by the **Spawn child process within virtualized environment** and **Spawn COM servers within virtualized environment** flags can be enumerated in the **Child Process Exception List.** Process names listed in the child process exception list behave *opposite to* the master child process virtualization setting. To edit the child process exception list, click the **Child Process Exception List** button. Process names may or may not include the process filename extension.

## Read-only virtual environments

In some scenarios, it may be desirable to prevent the user from making any modifications to the virtual environment, including the virtual filesystem and registry.

To block all modifications to the virtual environment, check the **Virtual environment is read-only** option.

## Automatic sandbox reset

The sandbox can optionally be configured to be reset automatically on application shutdown. An application publisher may want to do this to assure that any changes made to an application's settings are reverted when the application closes.

To enable the automatic sandbox reset feature, check the **Delete sandbox on application shutdown** option.

## Shutdown process tree on root process exit

The **Shutdown process tree on root process exit** option enables the shutdown of all child processes when the root process exits.

Note: By default, the startup file is the root process. However, if a virtual service is specified in the application configuration file and is set to auto-start when the application is launched, the virtual service acts as the root process in the process tree.

## Compress payload

Both the application profiling and streaming processes require that packages be built uncompressed.

To build applications without compression, leave the **Compress Payload** option unchecked.

## Startup executable optimization

The **Enable startup executable optimization** option attempts to launch the startup executable within the initial virtual machine process. This prevents the creation of a separate application process but may be incompatible with some applications.

## Spoon command-line arguments

Spoon supports command-line arguments of the form **/x[arg]** which modify virtual application behavior at run-time. In rare instances, these arguments may collide with command-line arguments designed for use by the virtualized application.

To disable processing of these arguments, uncheck the **Enable Spoon command-line arguments** option.

## Window class isolation

The **Enable window class isolation** option prevents the virtualized application from viewing window classes that have been registered by external processes. For example, this option may be used to prevent interaction between virtualized and non-virtualized versions of the same application when the application checks for existing class registrations.

## Enhanced DEP compatibility for legacy applications

The **Enhanced DEP compatibility for legacy applications** option provides compatibility for systems with Data Execution Protection (DEP) enabled. This configuration option is used primarily for virtual applications running on Windows 2003.

## Enhanced DRM compatibility

The **Enhanced DRM compatibility** option provides additional compatibility with common DRM systems, such as Armadillo.

## Trace process starts in debug output

The **Trace process starts in debug output** option sends a notification to **OutputDebugString** whenever a new process is started within the virtual environment. This notification takes the form of an XML snippet describing basic information, and it can be monitored with any debugging tool. The notification can also be monitored by a parent process within the virtual environment if a child process is being debugged.

## Force read-share files

The **Force read-share files** option forces any file opened by any process within the virtual environment to do so with the **READ_SHARE** flag set. This option is used to overcome compatibility issues caused by sharing violations.

## Always launch child processes as current user

By default, child processes launched by the virtual machine have reduced privileges. Enabling the **Always launch child processes as current user** option provides child processes with the same level of privileges as the virtual machine root process.

## Emulate elevated security privileges

The **Emulate elevated security privileges** option forces an application to run as if it has elevated security privileges, even if the application does not. Enabling this option eliminates UAC security prompts for elevation and subsequent application crashes due to lack of elevated privileges.

# Configuring the sandbox location

Depending on the configured isolation settings, certain edit and write operations may be redirected by the Spoon virtualization engine into an *application sandbox*- a filesystem folder where isolated modifications are persisted. Typically, the sandbox is located in a folder or network share where the user has full read and write permissions, allowing sandbox contents to be accessed and modified by the end user without any authentication or UAC prompts.

## Sandbox placement considerations

Please note the following recommended practices when configuring the sandbox location:

By default, the sandbox is placed in the **@APPDATALOCAL@\Spoon\Sandbox\@TITLE@\@VERSION@** folder, where the **@APPDATALOCAL@** token represents the local **Application Data** folder, and **@TITLE@**, and **@VERSION@** represent the application title and version respectively. The application title and version are configured in the **Properties** area. This location is the recommended default location for sandbox contents, as end users have full permissions to this location on standard Windows configurations. Note that distinct builds of the same virtual application use the same sandbox locations by default; you may want to modify this behavior if persisted user settings should not be preserved between virtual application updates.

When publishing a new version of a virtual application, direct the sandbox to the *same* location as the older version if you want user settings and data to be retained in the new version. Direct the sandbox to a *different* location (typically, by rolling the subdirectory version number forward) if you want user settings and data to be reset.

If deploying the virtual application on a USB device, place the sandbox in a subfolder of the **@APPDIR@** directory, which represents the location of the virtual application executable. This will have the effect of directing writes to the USB device. The recommended sandbox location for USB deployment is:
**@APPDIR@\Spoon\Sandbox\@TITLE@\@VERSION@**

If deploying the virtual application on an intranet file share, place the sandbox in a user-accessible subfolder on a shared network drive. The recommended sandbox location for intranet deployment is:
**\\ServerName\ShareName\%USERNAME%\Spoon\Sandbox\@TITLE@\@VERSION@**

Generally, you should not place the sandbox under any privileged folders, such as **@WINDIR@** or **@PROGRAMFILES@**. The virtual application may fail to execute properly if the Spoon engine is unable to write to the sandbox location at runtime.

Environment variables may be referenced within the sandbox location by enclosing the variable between percent signs, i.e. **%VARIABLE%**.

## Sandbox location variables

In addition to the standard root folder variables, the sandbox location can contain the following token variables:

@**TITLE**@:  Product title
@**PUBLISHER**@:  Product publisher
@**VERSION**@:  Full version string, in dotted quad format
@**WEBSITE**@:  Publisher web site
@**BUILDTIME**@ Virtual application build time, in a format similar to **2008.02.01T08.00.**

With the exception of the @**BUILDTIME**@ variable (set automatically), these variables are based on the values specified in the **Properties** area of the **Settings** pane.

# Building MSI Setup Packages

(Only applicable in certain editions of Studio)

Studio includes the ability to generate Microsoft Windows Installer (MSI) setup packages to facilitate deployment of virtualized applications. In addition to deploying the virtual application executable file to the host filesystem, Studio-generated MSI packages also allow creation of desktop and Start Menu shortcuts, creation of shell file extension associations to virtualized applications, and Control Panel uninstallers for application cleanup.

This section describes configuration and build processes for MSI setup packages.

## Configuring package information

This section describes global MSI package configuration options.  These options are located on the **MSI** root tree node of the **Setup** settings pane.

### Setting the MSI package location

The MSI package generated by Studio will be written to the file specified in the **Output Location** textbox. This textbox should contain the fully qualified name of the desired output file, including the output path and MSI file name.

By default, MSI packages are not automatically generated or updated when the virtual application is rebuilt.

To automatically update MSI packages after the virtual application is rebuilt, check the **Automatically generate MSI after successful application build** option.

Regenerating MSIs may significantly increase the time required to complete the build process. Therefore, Spoon recommends that this option be disabled during the virtual application development process.
It is also possible to manually force the MSI package to be regenerated.  To manually build the MSI package, click the **Build MSI** button.

Note: You must build the virtual application executable *before* the MSI package may be generated.  The **Build MSI** button will be disabled if the virtual application executable has not yet been built.

### Specifying package metadata

MSI setup packages contain a package manifest describing the product's name, version (can only be numbers and periods), and manufacturer. To configure the MSI package metadata, enter the appropriate values in the **Product Info** area.

Note: The metadata published on the MSI package is distinct from the metadata published on the virtual application executable itself. To modify executable shell metadata, specify the appropriate metadata on the **Settings** pane.

### Installation parameters

The **Installation parameters** area allows installation options such as install location and permissions parameters to be configured.

Applications may be installed either for the current user or for all users of the target device.  To install the application for all users, check the **Install applications for All Users** option.

Note: Installing applications for all users requires privileged access to the host device. Do not enable this option if the MSI package is designed for use by end users with standard user permissions.

The **Application Folder** specifies the location where the application executable will be installed. This usually has the form **[Application Data]\[Company Name\Product Name]**.

In the event that a user runs the setup package on a device which already has a version of the application installed, the MSI package may be designed to update the existing application version or side-by-side install with the existing application version.

To automatically update existing versions, select the **Automatically upgrade earlier application versions** option; to use side-by-side installation, select the **Allow side-by-side versions of the same application** option.

Note: Building with the **Allow side-by-side versions of the same application** option enabled causes a new setup package GUID to be generated. Once a build is completed with this option enabled, previous installations will no longer be upgraded in place, even if you revert to **Automatically upgrade earlier application versions** mode.

## Extended properties

MSI setup packages may also contain extended metadata, such as keywords, product author, product description, and publisher URL. To configure MSI package extended properties, click on the **Extended Properties** tree node in the **MSI** pane and enter the desired values.

# Creating desktop and Start Menu shortcuts

Desktop and Start Menu shortcuts allow the end user to launch the application directly from the Windows shell.

To add a desktop shortcut:

- Click on the **Desktop** node under the **Shortcuts** node in the **MSI** tree view. This displays the desktop shortcut list.
- Click the **Add Shortcut** button and select the desired shortcut name, target, and options. The **Target** dropdown is populated with the startup file list, allowing shortcuts to be quickly connected to jukebox entry points.

To install additional folders and subfolders on the desktop, click the **Add Folder** button and specify the folder name.

The same procedure can be used to add shortcuts and folders to the **Programs** section of the Windows Start bar, except that start bar items are configured under the **Programs Menu** of the **Shortcuts** node in the **MSI** tree view. Note that Start Menu items are installed either to the current user's Start Menu or to **All User**s' start menu depending on the **Install application for All Users** setting in the MSI installation parameters section.

# Creating file associations

File associations allow the appropriate viewer or editor application for a given file type to be automatically launched when the user double-clicks on a document in the Windows shell. For example, the **.doc** file extension might automatically launch a virtualized word processing application.

To create a file association:

- In **Setup**, Click on the **ProgIds** node and click **Add ProgId**.
- Enter a ProgId and Description in the **Create ProgId** dialog. File associations naming generally follows the convention **[Company Name].[Product Name].[Version]**.
- In the new ProgId, click **Add Extension** and enter an Extension and an optional MIME Type.
- In the new file extension, click **Add Verb** and enter a Verb, Command, and choose the Inherit behavior and Default.

Some common verbs are **open**, **edit**, **print**, and **view**. The **Verb** that is entered will be the text that is displayed when the user right-clicks on the file. When **Inherit** is checked, the behavior of the **Verb** will be controlled by the setup information in the virtual environment. When **Inherit** is unchecked, a **Target Startup File** and **Arguments** will need to be entered manually. The **Arguments** field should contain **"%1"** which is the full path to the file. When **Default** is checked, the **Verb** will be automatically executed when the file is double-clicked.

File association properties may be modified or deleted by selecting the appropriate **ProgId** in the **Setup** tree view and modifying the settings as appropriate.

# Deploying Virtual Applications

This section describes several different methods for deploying applications built in Spoon Studio.

## Deploying using Spoon Server

Spoon Server allows you to launch your applications instantly from web sites, portals, and client desktops. A typical Spoon Server setup consists of an Administration Site (where the server is managed), and a Portal Site (where virtual apps are hosted and streamed). Virtual applications are uploaded through the Spoon Server Administration Site, and are then instantly available on the Portal Site. They can then be launched with a

single click from within most popular web browsers using the Spoon Plugin- a small browser extension. In addition to enabling simple web-based distribution, Spoon's unique application delivery technology allows most applications to launch 5 to 20 times faster than with a traditional download.

For more information on deploying applications using Spoon Server, please refer to the Spoon Server User Guide. Visit the Spoon.net web site for details on Spoon Server pricing and licensing.

## About Spoon delivery technology

Spoon uses machine learning technology to automatically decompose complex applications into smaller functional and data units. Spoon does not require any streaming servers or specialized protocols, and works with no infrastructure changes.

Spoon automatically identifies a *prefetch* consisting of the components of the application which must be loaded in order for the user to begin using the application. The prefetch is generally around 10% of the total application size, though this can vary considerably depending on the behavior of the specific application. Once the prefetch is transferred, the application launches immediately.

Applications can optionally be registered to the local device upon transfer completion. Registration moves the application content to a permanent location on the local device, making it available offline, and creates all Start Menu icons, desktop shortcuts, and file associations related to the application.

To learn more about Spoon, please visit spoon.net.

## Supported Platforms

The Spoon Plugin works with most popular Internet browsers, including Internet Explorer, Firefox, Safari, Chrome, Opera, and all other browsers built using the Gecko API.

# Deploying using the Publish to USB feature

(Only applicable in certain editions of Studio)

This section describes how to deploy virtual applications to USB storage devices using the **Publish to USB** feature.

## Publishing virtual applications to USB storage devices

The **Publish to USB** feature publishes virtual applications to USB storage devices. When the USB storage device is attached to a host system, the virtual application automatically registers the **Setup** information to the host shell environment. This information is automatically unregistered when the USB device is removed from the host system.

To deploy virtual applications on USB devices:

- Open an existing virtual application configuration.
- Attach a USB storage device to the host system.
- Click **Publish to USB**, select the USB storage device, and click **Publish**.
- After the virtual application is published to the USB storage device, click **OK**.

To use virtual applications that are published to USB storage devices:

- Attach the USB storage device to the host system.
- (If prompted by AutoPlay, choose the **XUsb.exe** option. XUsb will then register the file associations and shortcuts associated with the virtual application.)

Remove the USB storage device to unregister the virtual applications from the host system.

Note: If AutoPlay is disabled on the host system, open the USB storage device's contents and manually run **XUsb.exe**.

# Registering virtual applications in the Windows shell

This section explains how to use the SpoonReg* *deployment tool via the Windows shell or command-line script to register and manage virtual applications built using Spoon Studio.

## Introduction to the SpoonReg registration tool

SpoonReg is a tool that provides a simple command-line interface for deploying virtual applications and managing the virtual desktop environment. Users and administrators can use SpoonReg to register virtual applications for a single user or, in the case of administrators, a group of users or devices. SpoonReg can be used to deploy and manage virtual applications and layers built using Spoon Studio.

After virtualizing an application with Spoon Studio, it is often desirable to make the application Start Menu icons, shortcuts, and file associations available on the users' desktop. SpoonReg allows you to register Spoon virtual applications in the shell, creating all of the shell associations that would generally be created during a standard install process. Unlike performing an installation, however, registration and un-registration can be performed almost instantaneously.

SpoonReg also provides the ability to create, reset, and remove application sandboxes- virtual environment "bubbles" where the virtualized applications reside. Sandbox management provides fine-grained control over application linking and intercommunication.

Spoon Server users and administrators can use the SpoonReg mechanism associated with Spoon Server to register applications to the desktop. For specialized deployment scenarios, contact your Spoon representative to learn how to obtain your own version of the SpoonReg.exe utility.

## Registering virtual applications using SpoonReg

SpoonReg provides a simple command-line interface for managing the virtual desktop environment. This section describes basic SpoonReg command-line syntax, including steps for registering, updating, and unregistering virtual applications.

## Command-line syntax

The following naming conventions are used in this section:

| Parameter | Description |
|---|---|
| **AppSpec** | An AppSpec is a path (relative or fully qualified) to a virtual executable or layer built with Spoon Studio |
| **SandboxSpec** | A SandboxSpec is the name or path of a virtual sandbox |

## Registering a virtual application

To register an application, use the command:
**SpoonReg.exe AppSpec**

This command creates all Start Menu items, desktop shortcuts, and file associations associated with the virtual application executable.

By default, registration will create a local cached copy of the virtual application executable and use the user's local profile as the sandbox location.

Note: The sandbox location specified during the virtual application build is ignored when registering applications using the SpoonReg tool or Spoon Virtual Desktop Service.

## Advanced registration options

Command-line parameters can be used to control the caching behavior and sandbox where the virtual application should be registered:
**SpoonReg.exe [Options] AppSpec[@SandboxSpec]**

| Parameter | Behavior |
|---|---|
| **/nocache** | The virtual application executable will not be copied to the client machine. All shortcuts and file associations will point to the full path as given by **AppSpec**. |
| **SandboxSpec** | This parameter refers to the name and path to an existing sandbox. If this parameter is specified and a sandbox with that name exists, the application will be registered into that sandbox. (See the "Sandbox management" topic in this section for additional details.) |

## Updating registration settings

Application registration settings can be changed by re-executing the registration command with the desired options:
**SpoonReg.exe [Option] AppSpec[@SandboxSpec]**

| Parameter | Behavior |
|---|---|
| **/nocache** | Disable caching of the specified application (reverses the **/cache** setting) |

| | |
|---|---|
| **/cache** | Enable caching of the specified application (reverses the **/nocache** setting) |

## Unregistering a virtual application

Unregistering a virtual application reverses the registration process, removing the virtual application, Start Menu icons, shortcuts, and file associations.

To unregister a virtual application, use the following command:
**SpoonReg.exe /unregister AppSpec[@SandboxSpec]**

It is also possible to unregister all applications with the single command:
**SpoonReg.exe /unregisterall**

# Client profiles

The SpoonReg command can be applied to the **Local**, **All Users** or **Roaming** Windows profiles. These profiles correspond to the user profiles available on the Windows operating system. It's recommended that the SpoonReg command is applied to the **All Users** profile whenever possible.

## The Local profile

Each user on a Windows device has a local user profile. Any changes to the local profile affect only that user on that device.

The Local profile is the default profile used by SpoonReg if no profile is explicitly specified on the SpoonReg command line.

## The All Users profile

Each device has a single **All Users** profile. Any changes made to the **All Users** profile affect all users on the device.

To register an application to the **All Users** profile, execute the SpoonReg command with the **/allusers** command line flag.

You must have administrative permissions on the device to register applications to the **All Users** profile.

## The Roaming profile

Each user in an Active Directory environment can have a roaming profile which is mirrored to other machines according to directory policy. Typically, the roaming profile is stored on a network server and is available from all devices on a network.

To register an application to the **Roaming** profile, execute the SpoonReg command with the **/roaming** flag.

Note: There is no roaming profile for **All Users**. Therefore, the **/roaming** flag has no impact when used in conjunction with the **/allusers** flag.

Note: Because SpoonReg is applied to specific user profiles, it cannot be used for the **LocalSystem** account (has no associated profile).

# Sandbox management

The SpoonReg tool allows creation and management of one or more virtual environment sandboxes.

A *sandbox* contains all of a virtual application's isolated data and settings as determined by the virtual application's isolation configuration settings. Applications registered to the same sandbox can view and modify each others' virtualized data and settings.

By default, all applications are registered into a single default sandbox named **Default**. In some cases, it may be desirable to group related applications into a sandbox that can be treated as a single management unit. When a sandbox is reset, all of the application content and data stored in that sandbox is purged and reverts back to the default state.

## Creating a sandbox

If no sandbox is specified during registration, the application will be registered to the default sandbox (**Default**).

To create an additional sandbox, use one of the following commands:

**SpoonReg.exe [Profile] /create [SandboxName] [SandboxPath]SpoonReg.exe [Profile] /c [SandboxName] [SandboxPath]**

If no path is provided, a default path is created under the **AppData** folder under the specified profile.

## Resetting a sandbox

Resetting a sandbox reverses all changes made to the sandbox, including any changes to data or settings made by the user. This restores all applications registered to the sandbox to their default state.

To reset a sandbox, use one of the following commands:
**SpoonReg.exe [Profile] /reset [SandboxSpec]SpoonReg.exe [Profile] /r [SandboxSpec]**

If a **SandboxSpec** is not supplied, the default sandbox is reset.

## Deleting a sandbox

Deleting a sandbox removes all applications, data, and settings from the sandbox.

To delete a sandbox, use one of the following commands:
**SpoonReg.exe [Profile] /delete [SandboxSpec]SpoonReg.exe [Profile] /d [SandboxSpec]**

If a **SandboxSpec** is not supplied, the default sandbox will be reset (the default sandbox cannot be deleted). Any applications registered to the deleted sandbox will be moved to the default sandbox.

## Moving a sandbox

You can use SpoonReg to move the sandbox location to a given path.

To move a sandbox:
**SpoonReg.exe [Profile] /move [SandboxSpec] [SandboxPath]**

# Deploying in Active Directory environments

This section describes how an organization using Microsoft's Active Directory can leverage that infrastructure with the SpoonReg tool to deploy Spoon virtual applications to their users.

## Active Directory

Active Directory allows the network administrator to manage users and groups within an organization. Many organizations use Active Directory to manage their network services. By combining Active Directory with SpoonReg, administrators can deploy virtual applications easily and reliably to one or more users in their organization.

## SpoonReg

SpoonReg manages the virtual desktop environment for a given user by registering and unregistering virtual applications. A typical SpoonReg command to register an application such as Firefox would be:
\\VirtualAppServer\Tools\SpoonReg.exe \\VitualAppServer\Apps\Firefox.exe

This represents the basic functionality of SpoonReg which would copy the virtual application executable, create Start Menu items and desktop shortcuts and setup file associations. (For additional functionality dealing with sandboxes, caching and automatic updates, refer to "Deploying using Spoon Virtual Desktop".)

## Using SpoonReg with Active Directory

In an organization, it is generally more desirable to manage a group of users rather than one at a time. By combining Active Directory with SpoonReg you can manage the virtual environment for a user, group, or Organizational Unit. Here we will walk through some scenarios on how this can be accomplished.

Note: In order to manage virtual applications using Active Directory, the users must have access to a shared network drive where the virtual application executable files exist. This can be specified by a full UNC path, or by using a mapped network drive.

# Scenario 1: Linking an organizational unit (OU) to a group policy object (GPO)

Active Directory offers many ways to manage network services for an organization.  Here we are going to look at how we can use an OU in combination with a GPO to manage the virtual application environment for a set of users.

# Organizational unit (OU)

In Active Directory, OUs are containers where you can place users, groups and other OUs. Using these containers the administrator can create a structure that models the hierarchical or logical structures within the organization. By setting up the OUs we can isolate the different groups of users that will receive their own set of virtual applications. Some examples would be Accounting, Sales, Marketing, etc.

# Group policy object (GPO)

GPOs are a way of applying a set of rules and features to a targeted set of users. Typically GPOs handle security, application installation, logon/logoff scripts, Internet Explorer settings and more. A GPO generally gets applied when a user logs on to a given domain. Based on their profile, various GPOs will be applied.

# SpoonReg and the GPO

For the purposes of virtual application deployment we can configure the GPO to run the necessary SpoonReg commands to register a given set of virtual applications. You can create and edit GPOs using the **Group Policy Object Editor**. This comes as an add-on to Microsoft's Active Directory. The simplest way to deal with the virtual application management is by creating a logon script that registers the virtual applications for a particular group in the organization. You would do this under **User Configuration > Windows Settings > Scripts** in the **Group Policy Object Editor**. You might add the following logon script for the accounting group:
\\VirtualAppServer\Tools\SpoonReg.exe \\VirtualAppServer\AllVirtualApps\Excel.exe
\\VirtualAppServer\Tools\SpoonReg.exe \\VirtualAppServer\AllVirtualApps\Firefox.exe
\\VirtualAppServer\Tools\SpoonReg.exe \\VirtualAppServer\AllVirtualApps\AcrobatReader.exe

Whereas you might add the following for the graphic design group:
\\VirtualAppServer\Tools\SpoonReg.exe \\VirtualAppServer\AllVirtualApps\AdobeIllustrator.exe
\\VirtualAppServer\Tools\SpoonReg.exe \\VirtualAppServer\AllVirtualApps\Firefox.exe
\\VirtualAppServer\Tools\SpoonReg.exe \\VirtualAppServer\AllVirtualApps\AcrobatReader.exe

# Linking the GPO to the OU

Once you have set up the OU and the GPO, you simply have to link them through the Group Policy Object Editor. After they are linked, any users that are put into that OU will have the linked GPOs applied when they logon. All of their virtual applications will appear when they logon to any machine on the domain where the GPO applies.

Note: SpoonReg has many other capabilities that can be applied in the GPO logon script, such as unregistering applications or registering applications to a specific sandbox.

# Scenario 2: Using Startup Scripts with GPO's to deploy virtual applications

This section will describe using Startup Scripts to deploy virtual applications to the **All Users** profile on a host system.

# Create a GPO with the Startup Script

For the purposes of virtual application deployment we can configure the GPO to run the necessary SpoonReg commands to register a given set of virtual applications to the **All Users** Profile. This will allow any user who logs into a host system to have access to the registered virtual applications.

Start by creating a new GPO with the **Group Policy Management Console (GPMC):**

- Open the **GPMC**
- Right-click on the OU that you want to link a GPO to
- Select **Create and Link a GPO Here** and give it a name
- Right-click the GPO and select **Edit.** This will open the **Group Policy Object Editor**
- Navigate to **Computer Configuration > Windows Settings > Scripts**
- Open the **Startup** item
- Click **Show Files**
- In the directory that is displayed by the **Show Files** button, create a **.bat** file. This file will serve as the **Startup Script** that will deploy your virtual applications to all the computers in the specified OU.

Use these samples as a guide to create the startup script:
**Sample 1**. Register virtual applications to the default sandbox in the **All Users** profile:
\\VirtualAppServer\Tools\SpoonReg.exe /allusers \\VirtualAppServer\AllVirtualApps\Excel.exe
\\VirtualAppServer\Tools\SpoonReg.exe /allusers \\VirtualAppServer\AllVirtualApps\Firefox.exe
\\VirtualAppServer\Tools\SpoonReg.exe /allusers \\VirtualAppServer\AllVirtualApps\AcrobatReader.exe

**Sample 2**. Register virtual applications to the specified sandbox in the **All Users** profile:
\\VirtualAppServer\Tools\SpoonReg.exe /allusers /c sandboxname
\\VirtualAppServer\Tools\SpoonReg.exe /allusers \\VirtualAppServer\AllVirtualApps\Excel.exe
\\VirtualAppServer\Tools\SpoonReg.exe /allusers \\VirtualAppServer\AllVirtualApps\Firefox.exe
\\VirtualAppServer\Tools\SpoonReg.exe /allusers \\VirtualAppServer\AllVirtualApps\AcrobatReader.exe

Once the Startup Script is created, add it to the GPO:

- Navigate to **Computer Configuration > Windows Settings > Scripts** in the **GPMC**
- Open the **Startup** item
- Click **Add**
- Click **Browse**
- Select the **Startup Script** that was created
- Click **Open**
- Click **OK**
- Click **OK**

# Deploying using the Spoon Virtual Desktop service

This section describes how to use Spoon Virtual Desktop to manage virtual desktop environments in non-Active Directory environments.

## Introduction to Spoon Virtual Desktop Service

The Spoon Virtual Desktop service runs on the user's machine and synchronizes their virtual desktop environment with a central SMB file server. The service is automatically installed when a virtual application or client configuration file is registered using the **/allusers** flag.

## Creating a client configuration file

Client configuration files (**.xclient** files) provide a way of managing all of a user's virtual applications in a single file. When registering a virtual application with SpoonReg, the information for that registration is stored in the default client configuration file.

The recommended approach for using the virtual desktop service is to set up one machine with all of the virtual applications that you want to distribute (see "Registering virtual applications in the Windows shell"). The client configuration file from this machine will serve as the master configuration file. Any users that you want to have the same virtual desktop configuration synchronize with that file using the virtual desktop service.

To register an application to the initial client, use the following command:
**SpoonReg.exe AppSpec**

Note: Remember that the location of the virtual applications on the network must be accessible and consistent for all of the users that are accessing those applications.

## Publishing the client configuration file

After you have created the master client configuration file, publish it to a network share that is accessible to the desired client machines by using either of the following commands:
**SpoonReg.exe /publish PublishPathSpoonReg.exe /p PublishPath**

Note: The Spoon Virtual Desktop Service runs as LocalSystem, meaning that security must be carefully considered. LocalSystem has access to resources on its domain via the machine account. The machine account name is the machine name followed by a **$** (e.g. the machine account for **server** is **server$**). The machine account is also included in the Everyone domain group. The final ramification of this is that the network location where the remote application source is located must be a network share with read access for Everyone or for the specific machine account. It cannot be a mapped network drive, since mapped drives are at the individual user level.

## Registering a client configuration file

After publishing the master configuration file, register it on the machines that will share the same virtual desktop environment with the following command:
**SpoonReg.exe /allusers ClientConfigurationPath**

Note: You must be an administrator to use the **/allusers** flag.

Registering the client configuration file will automatically install and start the Spoon Virtual Desktop Service and synchronize the virtual desktop environment.

## Updating the default polling interval

The default polling interval for the Spoon Virtual Desktop Service is 60 seconds. You can update this interval on an individual machine using SpoonReg with the following command:
**SpoonReg.exe /updateinterval [seconds]**

Using zero or nothing as the parameter for seconds will disable automatic updates.

## Disabling auto updates at the application level

To disable automatic updates for individual applications you need to update the master client configuration file. The following command updates the file, which should then be re-published to the shared location:
**SpoonReg.exe /noupdate /allusers ApplicationPath**

# Deploying virtual applications using MSI setup packages

Spoon allows virtual applications and components to be deployed using legacy MSI setup package technology.

## Create MSI setup packages directly within Studio

Spoon Studio can be used to create standalone MSI packages directly within the Studio environment. Generated MSI setup packages can include Start Menu items, desktop shortcuts, file associations, and other custom shell integration behaviors.

Deployment using generated MSI packages is appropriate in situations where existing MSI package deployment mechanisms are in place, or for deploying applications with shell integration without the SpoonReg Virtual Desktop client tool.

Virtual application and component shell integration settings are shared between MSI- and Virtual Desktop-based deployment, enabling easy migration between deployment models.

Refer to the section "Building MSI setup packages" for more information on the creation of MSI setup packages.

## Deploy virtual applications into legacy MSI setup packages

Studio also supports deployment of virtual application executables into legacy MSI setup packages.

## Import legacy MSI setup packages

Spoon Studio supports one-click import of legacy MSI setup packages into the Studio environment. Following import, the application can be customized and deployed as a Spoon-based virtual application or **SVM**.

# Deploying virtual applications using Microsoft Terminal Services RemoteApp

This section describes how to deploy Spoon virtual applications using Microsoft Windows 2008 Terminal Services RemoteApp server.

## Terminal Services RemoteApp

Terminal Services RemoteApp is a server-side program that provides end-users remote access to applications on a terminal server. Applications configured with TS RemoteApp appear as though they are running locally on the user's machine. End-users can run RemoteApps side-by-side with local programs or other RemoteApps.

To utilize virtual applications created in Studio with Microsoft TS RemoteApp:

- On the TS RemoteApp server, open the TS RemoteApp Manager and choose **Add RemoteApp Programs** by right-clicking inside the **RemoteApp Programs** list or through the **Action** drop down menu.
- Click **Next** in the **RemoteApp Wizard**.
- Click **Browse** and select the virtual application executable.
- After the virtual application is added to the list, select it and click **Properties**.
- If the virtual application has multiple Startup Files, configure the RemoteApp Program **Name** and **Alias**. If this is not done, the TS RemoteApp server will not distinguish between the separate applications in the suite.
- Still in the **Properties** window, select **Always use the following command-line argument**, enter in the **Trigger** for the Startup File that is to be executed, and click **OK**.
- In the **RemoteApp Programs** list, right-click the program that you added and choose **Create .rdp File**. There are no special requirements for the .rdp files.

If there are multiple Startup Files, repeat these steps for the other applications in the suite and deploy the shortcuts on the host systems.

# Walkthroughs

This section provides step-by-step instructions for using Spoon Studio in common scenarios.

## Manually configuring a simple virtual application

This section provides a walkthrough of manual configuration for a simple virtual application in Studio, based on the Windows Notepad application. In general, manual configuration should only be performed by experienced software developers virtualizing internally developed software applications.

- Click on the **Filesystem** button.
- Click on the **Application Directory** folder.
- Click on **Add Files....**
- Navigate to the **System32** folder under your Windows installation directory and double-click on **notepad.exe**. This adds the Windows Notepad executable to the virtual application.
- (Windows Vista only) Click on **Add Folder** and set the new folder name to **en-us**. Navigate into the **en-us** folder and click **Add Files....** Navigate to the **System32\en-us** folder under your Windows installation directory and double-click on **notepad.exe.mui**. This file is required by the version of Notepad which shipped with Windows Vista.
- Using Notepad or another text editor (on your machine, not the virtual application), create a file called **hello.txt** containing the text "Hello world" and save it to the **Desktop** folder.
- In Studio, Click on **Add Files...** again, navigate to your **Desktop** folder and select the **hello.txt** file that you just created. The display on the right should now show both **notepad.exe** and **hello.txt**.
- If it is not already visible, click on the **Virtual Application** tab on the ribbon bar to display the virtual application settings.
- In the **Startup File** dropdown, select **notepad.exe**. The startup file indicates which executable or file will be executed when the virtual application is started by the user.
- Click on the **Browse...** button next to the **Output Directory** textbox. Navigate to your Desktop folder and press **OK**.
- Press the **Build** button. Studio will now display a status dialog while it builds your virtual application.

To use your new Notepad virtual application, navigate to your desktop in a shell window and double-click on Notepad.exe. The Notepad application starts.

But how do we know we are inside a virtual application? In the shell, delete the **hello.txt** file from the desktop. Now, inside the Notepad window, click **File / Open...**, and navigate to the **Desktop** folder. Notice that the **hello.txt** file is still present! This is because the Notepad virtual application is using the virtual filesystem, which includes the **hello.txt** file that we added in the sixth step. You can open and view **hello.txt** exactly as if it were a real file in the physical filesystem.

## Building OpenOffice via snapshot process

This walkthrough describes creation of OpenOffice as a virtual application using the snapshot, jukebox, and setup features of Studio. Although the examples used in this walkthrough are for OpenOffice, the processes described can be used to virtualize and configure almost any application.

### Snapshotting

The snapshot process consists of two phases – the "before" snapshot and the "after" snapshot. The before snapshot takes an inventory of all files and settings that are installed on the computer. The after snapshot is taken after the application being virtualized is installed. The contents of the after snapshot are compared to the before snapshot to determine all changes that were made to the host system during installation. The after snapshot also copies the new or modified files to a snapshot directory specified by the user.

Since it is required to install and capture all application dependencies during this process, it is important that snapshotting be performed on a clean Windows machine. This guarantees that all dependencies will be included in the installation and captured by the snapshot process.

To snapshot the OpenOffice installation:

- Capture the before snapshot by clicking **Capture Before** on the Studio ribbon bar. The snapshot process may take a few minutes.
- Install OpenOffice and all of its necessary dependencies.
- Capture the after snapshot by clicking **Capture and Diff**. Studio will prompt for the directory where the snapshot files are to be stored. This directory is usually located on an external PC or network share.

## Saving the virtual application configuration

It is important to save the virtual application snapshot in its original state in case errors are introduced during virtual application customization and optimization. This also allows the application to be tested and modified without the need to re-snapshot after each iteration.

To save the virtual application configuration:
Open the configuration menu and click **Save Configuration** or **Save Configuration As**.

Studio configuration files are stored in the Spoon **XAPPL** file format. The **XAPPL** file does not contain the filesystem content. The filesystem content is stored in the **Files** directory which is created during the "after" snapshot.

Note: The **XAPPL** file uses relative paths to identify snapshot files. Therefore it is required that the **Files** directory and the **XAPPL** file be located in the same directory.

## Configuring the Virtual Application

Once the application snapshot has been created, the configuration can be modified or optimized depending on the desired virtual application behavior. For example, OpenOffice is a suite of applications; therefore program entry points need to be setup so each application can be run individually.

The following will need to be configured to get OpenOffice to function as a virtual application:

- Set the **Output File**
- Configure the **Startup Files** (Jukeboxing)
- Configure the **Setup** options
- ***(Optional)*** Remove unused installation files

## Setting the output file name

The **Output File** is the name of the virtual executable file or **SVM** that is created by Studio.

To set the **Output File**, click **Browse** next to the **Output File** field and assign it a file name.

## Startup files (Jukeboxing)

The **Startup File** settings identify the specific executable files that can be started by the virtual application. These can be started by executing the virtual application, from a command prompt, or from the shortcuts.

To configure the **Startup Files**:

- Open the **Startup Files** window by clicking on **Multiple** next to the **Startup File** field.
- Click the dropdown menu next to the **File** field, or navigate to the desired startup file in the **Filesystem** pane, right-click the file, and select **Add to Startup Files**.
- Enter in command line arguments as needed.
- Enter in a unique **Trigger** for that executable. **Triggers** are command line arguments that are passed onto the virtual executable that specify which **Startup File** to execute.
- Select **Auto Start** if the executable is to be started each time the virtual executable is run.

## Setup options

When virtualizing application suites such as OpenOffice, the user may want to create a setup file. Setup files generated by Studio are created using the MSI setup file format. Studio setup files only install the virtual application and create file associations and shortcuts. No other installation functions are performed.

To create a setup file for the virtual application, the **Output Location**, **Product Info**, **Installation Parameters**, **Shortcuts**, and **File Associations** need to be configured.

The **Output Location** defines where the setup file is created.

The **Product Info** is the metadata that will be associated with the setup file. The Product Info is displayed in the **Add/ Remove Programs** window. It is recommended that this information be accurate to avoid confusion.

The **Installation Parameters** control how the virtual application will be installed on the host system.

**Shortcuts** allow the end user to launch the application directly from the Windows Start Menu or Desktop.

## Output Location

To define the Output Location for the virtual application setup file:

- Click **Browse** next to the **Output Location** and assign it a file name.
- Check the **Automatically generate MSI after successful build** checkbox if the setup file should be created automatically after the build process.

## Installation Parameters

To install the virtual application for **All Users**, check the **Install application for All Users** check box. If checked, this option requires administrative privileges on the target system.

To automatically update existing versions, select the **Automatically upgrade earlier application versions** option. This option will update previous versions of this virtual executable.

To use side-by-side installation, select the **Allow side-by-side versions of the same application** option.

Note: The **Company Name\Product Name** needs to be entered in the **Application Folder** field. If neither is entered, the virtual application will be installed under folders named "**Company Name\Product Name**".

## Creating Shortcuts

To create shortcuts that open specific application within the OpenOffice suite:
Click **Add Shortcut** and assign it a **Name**, assign it a **Target**, select an **Icon**, and enter any arguments that need to be passed to the specific application.

Folders that are created by the setup package can also be setup in this pane.

## File Associations

To create File Associations:
Click on the **Add ProgId** button and enter the **ProgId** and a **Description** of the file association.

Click **Add Extension** and enter the file extension and **MIME Type** (if necessary) to the **ProgId**. If a verb is needed for the file extension, click **Add Verb** and enter the necessary information.

## Removing unnecessary files from the snapshot

During the installation process for many applications, temporary installation files are created. While these files are necessary for the installation, they are not required for the virtual application to run.

In OpenOffice, the installation files are created in the same directory that the install executable was executed from.

To remove these files from the snapshot:

- Open the **Filesystem** pane
- Navigate to the location of the installation files, right-click the folder, and select **Delete.**

Note: For OpenOffice, removing the installation files from the snapshot may reduce the output virtual application binary size by up to 151 MB.

## Build and Test

After the virtual application has been configured, customized, and optimized, it can now be built and tested.

To build the virtual application:

- Click on the **Build** or the **Build and Run** button. This will create virtual application binary file where the **Output File** is defined. The build process may take a few minutes.

To test the virtual application:

- Execute the virtual application binary file. The OpenOffice splash screen will display and the OpenOffice Quickstarter will open.
- Execute the virtual application binary file from a command prompt with the "**swriter**" Trigger. For example, run "**openoffice.exe swriter**" from a command prompt and the OpenOffice Quickstarter and Writer applications will open.

To test the Setup options:

- Execute the setup file on a system without OpenOffice installed. The virtual application, shortcuts, and file associations will be installed on the host system.
- Open the OpenOffice Writer Program from the start Menu shortcut.
- Open a file that is associated with the OpenOffice Writer program.

# Best Practices

This section describes various best practices for making use of Spoon Studio. Note that these methods of use are all optional.

## Best practices for snapshotting

The following practices are recommended for optimal use of the snapshotting feature.

### Machine configuration

- Perform snapshotting on a clean machine: Snapshotting on a clean machine assures that all dependencies will be installed by the application setup. Installing on a machine with existing components may cause dependencies to be inadvertently included in the "before" snapshot and therefore excluded from the final virtual application output.
- Use snapshotting in conjunction with whole-machine virtualization: Configuring a clean machine using a whole-machine virtualization tool such as Microsoft Virtual PC and saving a "before" snapshot based on this image allows many distinct virtual applications to be snapshotted in rapid succession by reverting the whole-machine virtual state.
- Snapshot on the earliest operating system variant you expect to target: Most applications can be successfully configured by snapshotting on the earliest (least common denominator) base operating system to be targeted. A small number of applications may require multi-platform snapshotting for successful deployment across all operating system variants.

### Snapshot process

- Save the "before" snapshot: Saving the snapshot assures that you need only take the "before" snapshot a single time. It may be necessary to restart the target application during the installation process, in which case you will need to reload the "before" snapshot prior to capturing the "after" snapshot.
- To prevent problems with open file handles, it is recommended that processes and services that were started during installation are stopped prior to capturing the "after" snapshot. In most cases it is sufficient to exit the target application, unless the application installs services, such as Microsoft SQL Server, in which case the services should also be stopped prior to taking the "after" snapshot.
- Clean up your image: While Studio automatically excludes many unnecessary files and registry keys, snapshotting often picks up many unnecessary items. If you have adequate technical understanding to do so, you may significantly reduce virtual application size by manually removing unnecessary items from the snapshot delta.

## Capturing updates to an application via snapshot process

(Only applicable in certain editions of Studio)

Virtual application updates can be captured within Studio via the snapshot process.

To capture an update via snapshot process:

- Install the original native version of the application you wish to update on a clean machine.
- Click **Capture Before** to snapshot the original version of the app.
- Install the necessary updates to the native application.
- Click **Capture and Diff** to create the "after" snapshot of the app. This will capture the deltas between the original version and the updated version.
- Make sure the Project Type is set to **Component**, then click **Build** to create the **SVMs**.

This process will only capture the changes between the original executable and the installed updates. The resultant **SVM** can then be applied to the original virtual executable (for more information on updating virtual applications using **SVMs**, refer to the topic "Specifying additional SVMs for a virtual application" in the "Advanced topics" section).

# Setting up and managing a build lab

For organizations that will be building virtualized applications on a regular basis, Spoon recommends setting up a build lab to ensure quality application builds and decrease the build time per application. We recommend that certain steps in the virtual application build process be performed on multiple operating systems in order to ensure maximum compatibility and performance. Because many of these steps are time-consuming, the process can move more efficiently if the steps are performed on multiple machines in parallel. This process is referred to as "pipeline building" and is particularly useful when virtualizing a high volume of apps.

## Setting up a build lab

To ensure that virtual application builds will function for all users, Spoon recommends a build lab is set up with the following six machine configurations:

- Windows XP, Service Pack 3, x86 architecture
- Windows XP, Service Pack 2, x64 architecture
- Windows Vista, Service Pack 2, x86 architecture
- Windows Vista, Service Pack 2, x64 architecture
- Windows 7, x86 architecture
- Windows 7, x64 architecture

Each machine will need to be frequently re-imaged back to an initial configuration, or "clean" state. "Clean" machines are required to effectively complete the build and testing processes. To facilitate this process, you will need to save an image of the machines initial configuration on a local partition, then use third party software to create an external drive that can be used to boot the machine from that image.

Spoon recommends following these steps to setup the initial configuration for each machine:

- Install the operating system; delete all existing partitions, create a new partition to save the "clean" image.
- Install all current patches for the OS (don't upgrade Internet Explorer).
- Create accounts for an administrator, a standard user with UAC enabled, and a standard user with UAC disabled.
- Disable automatic updates.

Steps 5 through 9 are optional, but still recommended:

- For Windows 7 only: run Internet Explorer 8 once to prevent repeating the first-run wizard each time you re-image the machine.
- Enable RDP access.
- In Power Options, configure the machine to never go to sleep.
- Disable the Windows Firewall.
- Right-click My Computer, select Properties, select Advanced, edit the Performance Settings to "Adjust for best performance"
- Image the system, saving the image files to the appropriate partition.

Once the operating system image is saved to a partition, Spoon recommends using third party software such as Paragon Partition Manager to create a USB drive that can be used to restore the machine from the image.

**Note:** It is not strictly necessary to set up a build lab using physical machines; a build lab can also be set up using a collection of virtual machine images. If this is your desired approach, Spoon recommends creating virtual machine images for each of the six machine configurations described in this section. The virtual machines should be configured following the steps set forth in this section, excluding the creation of a new partition. It's not necessary to create a partition to save the "clean" image, because it's possible to simply save the current state of a virtual machine.

## Build process

The "build process" is the process of taking a Windows desktop application and converting it into a standalone executable (EXE) or Spoon Virtual Machine (SVM).

- Use the snapshot technique to capture the application on a "clean" Windows XP 32-bit machine. The snapshot process is covered in detail in the "Snapshotting Applications" topic of the "Getting Started" section.
- Use Studio to generate a standalone executable (this will launch the virtual application).
- Test this executable per the recommendations in the "Build testing process" section.
- If the virtual application fails to run on Windows Vista or Windows 7, it may be necessary to perform the snapshot on multiple platforms, and then perform a platform merge. Refer to the "Platform Merge" topic in the "Advanced Topics" section for details on how to perform a platform merge.
- After the application has been verified to work on all six platforms, use Studio to generate a component (an SVM file) if needed.

If the virtual application fails to build or function correctly, please refer to the "Troubleshooting" section in this document or contact Spoon support.

## Build testing process

The build test process exists to ensure that the build process successfully captured all aspects of the target application necessary for the virtual application to run correctly.

Spoon recommends adhering to the following strategies while testing virtual applications:

- Conduct testing on each of the six machine platforms identified in the "Setting up the build lab" section.
- Under ideal circumstances, each machine in your build lab should be "cleaned" prior to testing the virtual application.
- At a minimum, the virtual application should not be tested on a machine where the application is also natively installed.

## Profiling the application

If the virtual application is intended to be delivered via the Web as a streaming application, you may choose to profile the virtual application in order to dramatically reduce the buffering time needed to launch the application. Spoon recommends profiling any application greater than 10 MB in size.

During the profiling process, you will use the application as a typical user would, while Studio observes this behavior and stores it in a transcript. These transcripts are then subsequently used to construct a streaming model of the application. There is no limit to the number of transcripts that can be used to construct a streaming model, Spoon recommends gathering at least one transcript from each of the six machine platforms identified in the "Setting up the build lab" section. The profiling process is covered in detail in the "Creating application streaming models" topic in the "Advanced Topics" section.

## Managing builds

Spoon recommends allocating a folder on a shared network drive to store and organize the SVMs, EXEs and models generated as a result of the virtualization process.

During the build process, Spoon recommends storing the snapshot files, the EXE and SVM in the following folder structure: \\<Share>\Builds\<Application>\<Version>.

During the profiling process, Spoon recommends storing the transcripts and model files in the following folder structure:Transcripts - \\<Share>\Builds\<Application>\<Version>\Transcripts\Models - \\<Share>\Builds\<Application>\<Version>\XStream\<Model Number>\

If it's necessary to deliver virtual applications and models to Spoon, please contact Spoon support (support@spoon.net) for additional details.

# Advanced Topics

This section deals with advanced topics you may encounter while using Spoon Studio.

## Customizing the Studio interface

This section describes Studio interface customization options.  All options described in this section can be found under the **Options** menu item.

## Proxy settings...

Studio uses the Internet to check for product updates and download update packages. If your computer is located behind a firewall, it may be necessary for Internet access to take place through a proxy server. By default, Studio will use the default Internet settings configured on the host machine. In some circumstances, however, it may be necessary for you to manually configure the proxy server settings.

To manually configure proxy settings, select the **Proxy settings...** option from the **Options** menu.  Provide the proxy server address, the server port and the type of authentication, if any, the proxy server uses. Select the 'Bypass proxy server for local addresses' option to bypass the proxy server when accessing resources located on the local network. Please contact your network administrator if you need assistance configuring the proxy settings.

## Automatically detect associated runtimes and components

By default, Studio will scan the virtual filesystem at build time and verify that the current configuration includes all available runtimes and components associated with file types contained in the virtual filesystem. This behavior is recommended to assure maximum virtual application reliability.

If you wish to disable this scan, uncheck the **Automatically detect associated runtimes and components** option in the **Options** menu.

## Play sound on build completion

By default, Studio plays a short sound to notify the user of virtual application build completion.

If you wish to disable this sound, uncheck the **Play sound on build completion** option in the **Options** menu.

# Quick snapshot mode

By default, Studio uses a "quick" snapshotting algorithm that attempts to minimize the amount of time spent scanning the host system device state during snapshotting. In very rare cases, use of this mode may result in an improperly configured virtual application. Use of quick snapshot mode may also slightly increase the size of the virtual application configuration contents. It is strongly recommended that snapshotting be performed using the quick snapshot mode, as this is compatible with the vast majority of applications. Disabling quick snapshot mode significantly increases the amount of time required to complete the virtual application configuration process.

To disable quick snapshot mode, uncheck the **Quick snapshot mode** item from the **Options** menu.

Note: "Before" and "after" snapshots must be taken using the same snapshotting algorithm. Loading a saved snapshot image causes Studio to automatically configure the snapshotting mode to be consistent with the algorithm used during the saved snapshot capture.

# Snapshotting Internet Explorer

Internet Explorer snapshot compatibility mode should be used when snapshotting Internet Explorer application setups. This mode ensures a more complete capture of the Internet Explorer portions of the registry and filesystem, whereas a normal snapshot process ignores those parts of the registry and filesystem.

To use Internet Explorer snapshot compatibility mode, click the **Options** menu above the ribbon bar in Spoon Studio, and select **Internet Explorer snapshot compatibility mode** from the dropdown.

# Well-known root folder variables

The Spoon engine dynamically remaps well-known root folders such as **My Documents** and **Program Files** to the appropriate location based on the host operating system at runtime. This assures, for example, that the virtualized **My Documents** folder will be mapped to **\User\Bob\Documents** when running on Windows Vista or **\Documents and Settings\Bob\My Documents** when running on Windows 2000.

Most of the time, configurations are constructed using snapshotting or in the graphical user interface. However, if manually modifying the configuration, the following well-known root folder variables may be used to configure virtual filesystem locations. Root folder variables are case sensitive.

The following is a complete list of root folder variables recognized by Studio and the corresponding folder name displayed in the filesystem graphical user interface, followed by a brief description of the root folder.
**@APPDIR@** (Application Directory): Folder where the executing virtual application executable resides.
**@WINDIR@** (Windows): The operating system install location root.
**@SYSDRIVE@** (System Drive): The root folder of the drive containing the operating system installation.
**@PROGRAMFILES@** (Program Files): The Program Files folder.
**@PROGRAMFILESCOMMON@** (Program Files\Common): The Program Files\Common Files folder
**@SYSTEM@** (System Drive\Windows\System32): The Windows System32 folder.
**@APPDATALOCAL@** (Current User Directory\Local Application Data): The folder that serves as a common repository for application-specific data that is used by the current, non-roaming user.
**@APPDATA@** (Current User Directory\Application Data): The folder that serves as a common repository for application-specific data for the current roaming user.
**@STARTUP@** (Current User Directory\Start Menu\Programs\Startup): The folder containing the current user's startup items.
**@PROGRAMS@** (Current User Directory\Start Menu\Programs): The folder that contains the user's program groups.
**@STARTMENU@** (Current User Directory\Start Menu): The folder containing the user's Start Menu contents.
**@DESKTOP@** (Current User Directory\Desktop): The current user's Desktop folder.
**@TEMPLATES@** (Current User Directory\Templates): The folder that serves as a common repository for the current user's document templates.
**@FAVORITES@** (Current User Directory\Favorites): The current user's Favorites folder.
**@DOCUMENTS@** (Current User Directory\My Documents): The current user's My Documents folder.
**@MUSIC@** (Current User Directory\My Music): The current user's My Music folder.
**@PICTURES@** (Current User Directory\My Pictures): The current user's My Pictures folder.
**@PROFILE@** (Current User Directory): The folder that stores the current user's profile data.
**@APPDATACOMMON@** (All Users Directory\Application Data): The folder that serves as a common repository for application-specific data that is used by all users.
**@STARTUPCOMMON@** (All Users Directory\Start Menu\Programs\Startup): The folder containing startup items for All Users.

**@PROGRAMSCOMMON@** (All Users Directory\Start Menu\Programs): The folder for components that are shared across applications.
**@STARTMENUCOMMON@** (All Users Directory\Start Menu): The folder containing the Start Menu contents for All Users.
**@DESKTOPCOMMON@** (All Users Directory\Desktop): The shared Desktop folder.
**@TEMPLATESCOMMON@** (All Users Directory\Templates): The folder that serves as a common repository for shared document templates.
**@FAVORITESCOMMON@** (All Users Directory\Favorites): The shared Favorites folder.
**@DOCUMENTSCOMMON@** (All Users Directory\Documents): The shared Documents folder.
**@MUSICCOMMON@** (All Users Directory\Music): The shared Music folder.
**@PICTURESCOMMON@** (All Users Directory\Pictures): The shared Pictures folder.
**@PROFILECOMMON@** (All Users Directory): The folder that stores the shared profile data.

# Building from the command line

Studio can optionally be executed from the command line. This is particularly useful for building virtual applications as part of an automated build process.

The command line version of Studio is called **XStudio.exe** and can be found in the Studio installation directory. To build a virtual application from the command line, execute **XStudio Configuration.xappl** from the command prompt, where **Configuration.xappl** is the name of the Studio project created using the graphical interface.

Options specified in the **XAPPL** file may be overridden with the command-line flags given in the following table.

| Option | Description |
|---|---|
| **/before** [/beforepath Snapshot Path] | Performs a before snapshot and saves the snapshot to the optionally specified snapshot folder. If no snapshot folder is specified, the default snapshot folder is used. |
| **/after** [/beforepath Snapshot Path] **[/o Output Path**] | Performs an after snapshot using the optionally specified before snapshot path. If no before snapshot path is specified, the default snapshot folder is used. The output **XAPPL** and snapshot files are saved to the optionally specified output path. If no output path is specified, the output **XAPPL** and snapshot files are saved to the current user's desktop. |
| **/component** or **/layer** | Forces the output type to component. This mode causes an **SVM** to be generated. |
| **/d** | Forces a diagnostic-mode output to be generated. |
| **/l License file** | Applies the specified license file. |
| **/uncompressed** | Results in an uncompressed payload (payloads are compressed by default). This option overrides the selection in the **Configuration.xappl** file. |
| **/deletesandbox** | Forces the virtual application to delete the sandbox on application shutdown. |
| **/isolatecom** | Imports **COM** registration information from the host registry into the **Configuration.xappl** file based on any DLL or OSX files in the file system. Note that this process is supplemental to the snapshot process and would only be used in cases where the original snapshot is incomplete.<br>E.g. **XStudio.exe Configuration.xappl /isolatecom** |

Note: Snapshots generated from the command-line using the **/after** flag will not have an output path specified. When using programmatic snapshotting, it is strongly recommended that additional scripting be performed to apply necessary additional configuration to the generated **XAPPL** file.

# Importing configurations from external tools

Studio allows configurations from certain external application virtualization tools to be automatically converted into Studio configurations. Supported external configurations currently include MSI setup packages, ThinApp configurations, and Novell AXT snapshots.

To import a configuration from an external tool:

- Click the **Start menu** button control menu (or press **Alt-F**)
- Select **Import Configuration**. This displays the configuration import wizard.
- Click **Browse** to select the configuration to be imported.
- Click **Next**.
- Follow the step-by-step instructions in the wizard to complete the import process.

Note:* *Some applications which depend on specialized custom actions during the MSI installation process may require additional configuration following MSI import to be fully functional. Such applications may need to be imported using the snapshot capture method.

# Running native applications in virtual environments

Studio allows natively installed applications to launch in virtual sandboxed environments. This is helpful when a natively installed application can utilize resources contained in a virtual package. For example, a user virtualizing a plugin for Microsoft Outlook would want to enable a local version of Outlook to run in the same virtual sandbox as the plugin. This is accomplished in Studio by setting the natively installed application as the startup file (or one of the startup files).

To enable a natively installed application to launch in a virtual environment:

- In the **Virtual Application** tab, click on the **Multiple** button next to the **Startup File** field
- In the **File** column, enter the local path of the natively installed application.
- (Optional) Check the **Auto Start** option to have your natively installed application automatically run when the virtual application is launched
- Click **OK**

This will enable your virtual application and natively installed application to interact with each other in the same virtual environment.

A sample startup file path for Microsoft Word would look like this:
**@PROGRAMFILES@\Microsoft Office\Office12\WINWORD.exe**

If **Auto Start** has been enabled, Microsoft Word will launch with the virtual application, in the same virtual environment.

# Modifying virtualization behavior at run-time

Most virtualization behavior is specified during virtual application design using the Studio interface. However, in some cases, it is useful to override virtualization behaviors at application run-time.

Spoon allows the following virtualization settings to be specified on the virtual application command line. Settings specified on the command line supersede design-time virtual application settings.

| Flag | Behavior |
|---|---|
| **/XEnv=Variable Name=Value** | Specifies additional environment variables. Multiple /XEnv arguments can be used to add additional environment variables. |
| **/XLayerPath=Layer Path** | Adds the given **SVM** into the virtual environment. Multiple /XLayerPath arguments can be used to add additional virtual layers. |
| **/XSandboxPath=Sandbox Path** | Specifies the path to be used for the application sandbox. |
| **/XShellEx=Command** | Specifies a shell execute command to be launched from within the virtual application environment. This option overrides any startup files specified in the virtual application configuration. Only one /XShellEx argument can be specified. |
| **/XShellExVerb=Command Verb** | Specifies the verb to be used in conjunction with the XShellEx command. The default verb is OPEN. |
| **/XLogPath=Log Path** | Specifies the destination path for generated log files (only applies to executables built in diagnostic-mode). This path can include a custom file name pattern, e.g. **c:\logs\mylog**.log* |
| **/XSpawnVmExceptions=Process Exceptions** | Accepts a semi-colon delimited list of processes to be added to the child process exception list (refer to the Child processes section for more information). E.g. **/XSpawnVmExceptions=notepad.exe** |
| **/XEnable** and **/XDisable** | Enables or disables specific process configuration options. These options include: **SpawnVm**, **ReadOnly**, **ExeOptimization**, **SpawnComServers**, **IsolateWindowClasses**, **IndicateVirtualization**, **DeleteSandbox**, **NotifyProcStarts**, **ReadShare**, **DRMCompat**, **ChildProcAsUser, ShutdownProcTree, DEPCompat**, and **IndicateElevated**; all of which correspond to specific options in the Process Configuration tab, e.g. **/XEnable=SpawnVm;DEPCompat**. In addition, **SuppressLogging** can be specified to enable or disable diagnostic mode. |

# Specifying additional SVMs for a virtual application

Studio users may want to specify additional **SVM**s for applications, in the case of updates or patches. Studio allows two mechanisms for doing this.

The first mechanism is via the command line using the **/XLayerPath=** syntax. This syntax takes a path with optional wildcards to additional **SVM**s to load.

An example of a specified **SVM** path using a wildcard:

**virtual-app.exe /XLayerPath=@APPDIR@\patches*.svm**

An example of specifying **SVM**s from multiple locations:
**virtual-app.exe /XLayerPath=@APPDIR@\patches*.svm /XLayerPath=@APPDIR@\officepatches*.svm**

An example of specifying **SVM**s on a network share:
**virtual-app.exe /XLayerPath=\\network\share\patches*.svm**

An example using Microsoft Office:
**MSOffice.exe /XLayerPath=c:\Patches\MSOffice_*.svm.**

This would do a wildcard match finding any files such as **MSOffice_001.svm** in the **c:\Patches** directory.

Note: The **SVM**s are applied in reverse-alphabetical priority. This means that items in **MSOffice_002.svm** have higher priority than items in **MSOffice_001.svm**.

The second mechanism is a **XAPPL** file specified way to load additional **SVM**s. It is via the **<XLayers>** portion of the **XAPPL** file and has the following elements:

| Attribute | Description |
|---|---|
| **XLayerSearchPattern** | Attribute to provide the default search pattern, similar to what would be passed to /XLayerPath |
| **<RequiredXLayername="@APPDIR@\APP.svm">** | Sub-elements specifying which **SVM** must be loaded, or else an error is reported |

**Sample XAPPL configuration snippet**

```
<XLayers XLayerSearchPattern="@APPDIR@\StudioDependencies.svm">
<RequiredXLayer name="StudioDependencies.svm" /> </XLayers>
```

Both methods allow the use of the @**VARIABLE**@ format.

Multiple **SVMs** may be specified after the **XLayerSearchPattern** attribute in a semi-colon delimited list. **SVMs** specified first in the list will take precedence over **SVMs** specified later in the list. If multiple **SVMs** are specified in one search pattern through the use of the '*' wildcard, the **SVMs** are applied in reverse-alphabetical priority. For example, items in **MSOffice_002.svm** would have higher priority than items in **MSOffice_001.svm** .

Note: Newer versions of Studio make use of **SVMs** as opposed to **XLayer** files. Older **XLayer** files must be rebuilt as **SVMs** as there is currently no supported conversion utility. **SVMs** function in the same way as **XLayer** files in that they will be auto-integrated with virtual executables by being placed in the same directory as the executable.

# Platform merge

The Merge Platforms feature allows virtual application configurations snapshotted on multiple operating system variants (Windows XP, Vista, etc.) to be combined into a single configuration. At runtime, the virtualization engine dynamically applies the configuration options appropriate for the operating system variant used for execution.

Tip: The most common platform merge scenario is a merge of snapshots taken on Windows XP and Windows Vista. This is because some newer applications use operating system features specific to Windows Vista.

To merge configurations from multiple platforms:

- From the **Advanced** tab, click the **Merge Platforms** button.
- Click **Browse** and open the appropriate configuration for each applicable operating system variant
- For operating systems without a configuration, choose which configuration it should use by using the Inherit option.
- When all configurations have been selected or set to **Inherit**, click **Browse** in the **Merge Settings** area, choose where to save the merged configuration, and click **Merge**.

To display or edit a specific operating system from a merged configuration:

- Open the merged configuration.
- From the **Advanced** tab, click on the **Display** drop down menu.
- Select the operating system that you want to display or edit.

The **Filesystem** and **Registry** panels will only display settings specific to the selected operating system. Note that you cannot edit configurations which are inherited from other platforms; to edit inherited configurations, you must select and edit the master configuration.

To change the inheritance of an operating system in a merged configuration:

- Open a merged configuration.
- From the **Advanced** tab, click the **Display** drop down menu.
- Select the operating system that you want to modify.
- Select the platform from which to inherit using the **Inherits** drop down menu.

# Creating application streaming models

The **Streaming** section of the **Advanced** tab allows you to profile and build streaming models for a virtual application.

The **Profile** feature generates transcripts, or *profiles*, which are then used to create a streaming model for the virtual application. Clicking the **Profile** button launches the application and creates a single transcript file based on observed user behavior during that run. It is recommended that multiple transcripts are created before creating a streaming model. Using multiple transcripts allows the streaming system to take into consideration different use cases for the application. It is also recommended that at least one transcript be created for each operating system.

Note: Only uncompressed virtual applications can be profiled and streamed. Compression is automatically disabled during the model build process.

To profile virtual applications:

- Build the virtual application.
- Click the **Profile** button on the **Advanced** tab.
- Select the output location for transcripts and click **OK**.
- After the virtual application launches, use the application for approximately one minute, as if you were a typical end-user.
- Close the application. Once the application terminates, the transcript will be created in the selected output location.
- Create additional transcripts as needed.

Once the necessary profiles have been created, the streaming model is ready to be built. The model build process uses the transcripts and **Connection Speed** parameter to compute a model of execution. After the model build process is complete, the streaming files are written to the selected output folder. The **Connection Speed** setting is used to optimize delivery of application content to the end-user.

To create a streaming model:

- Select the desired **Connection Speed**. The 1.5Mbps connection speed setting is recommended for most scenarios.
- Click the **Build Model** button.
- Select the folder where the transcripts are located and click **OK**.
- Select the folder where the streaming model will be created and click **OK**.

# Application Expiration

This section describes the **Expiration** feature. With the **Expiration** feature, virtual applications can be set to expire after a certain number of days or after a certain date.

To set a virtual application to expire after a specific number of days:

- Click on the **Expiration** button.
- Check the **Disallow execution after number of days** checkbox.
- Select the number of days after the application is first executed on a system it will take to expire.
- Choose the **Time Source** the virtual application will use to validate the date.

To set a virtual application to expire after a certain date:

- Click on the **Expiration** button.
- Check the **Disallow** execution after date checkbox.
- Select the date the virtual application will expire.
- Choose the **Time Source** the virtual application will use to validate the date.

For all expiration modes, the **System clock** setting will use the host system's clock to validate the date. The **Web server clock** setting will validate the date against an HTTPS-based web server. Check the **Disallow execution if web server is unreachable** checkbox to prevent the application from being executed offline.

Tip: The **Web server clock** setting is more secure than the **System clock** setting since it prevents the expiration mechanism from being circumvented by modifying the system clock. However, this setting will prevent applications from executing on devices which cannot connect to the time server source.

Optionally, an **Expiration Warning** can be set to warn the user when the virtual application is about to expire. The message will be displayed each time the virtual application is executed when it is within the specified threshold.

# Applying the virtual application configuration to the host device

Studio allows the virtual application configuration to be applied to the host system. Applying the virtual application configuration to the host system is helpful when creating **SVM** updates for virtual applications.

To apply the virtual application configuration to the host system:

- Click the **Apply Configuration** button in the **Start menu** of the Studio application.
- Enter the path of the sandbox to be merged into the current configuration.
- Click **Yes** to acknowledge that the **Apply Configuration** process cannot be undone.

Note: The **Apply Configuration** feature is not intended for use as an installation process for virtual applications.

For example, to create an **SVM** update to the Firefox virtual application template:

- Use the **Configuration Wizard** to create a Firefox virtual application.
- With the Firefox configuration loaded, run the **Apply Configuration** process as above.
- Open a new virtual application configuration.
- Capture a before snapshot.
- Open Firefox, select **Help>Check for Updates**, and apply any updates.
- Capture an after snapshot.
- Build the captured updates as an **SVM**.
- Execute the built **SVM** on top of the original virtual Firefox browser and notice that the updates have been applied.

# Enabling shared object isolation

Studio provides the ability to isolate shared objects in memory. Certain applications will refer to objects in memory by a specific name, which can cause runtime errors if a named object in a virtual application's memory collides with other objects of the same name in the memory of a natively installed version of the application on the same device. Shared object isolation creates unique names for memory objects at runtime, in order to allow a virtual application and a natively installed version of the same application to run side by side without conflict.

Currently, shared object isolation can only be enabled by manually editing the **XAPPL** file for a virtual application.

In the following scenario, OBJECT 1 and OBJECT 2 are named objects used by a virtual application that conflict with identically-named objects used by a natively installed application. Common named objects include mutexes and named pipes.

To enable shared object isolation for OBJECT 1 and OBJECT 2 in a virtual application:

- Open the **XAPPL** file of the virtual application you are working with in a text editor
- Replace the **<NamedObjectIsolation ../>** element with the example below:

```
<NamedObjectIsolation enabled="False">
  <Exception regex="[OBJECT 1]" />
  <Exception regex="[OBJECT 2]" />
</NamedObjectIsolation>
```

- Reload the **XAPPL** file in Studio and build the application

The resulting virtual application will have shared object isolation enabled. Note that multiple objects in memory can be isolated simultaneously.

# XAppl file format

## Overview

A **XAPPL** file is an XML representation of all the virtual application configuration settings.

All paths in the **XAPPL** file are relative to where the **XAPPL** file resides. For example, the **source** attribute of a File element will begin with **".\Files\"**. The **".\"** directory is the path where the **XAPPL** file should reside in order for Studio to locate the physical source files during the build process.

The **XAPPL** file must adhere to all XML syntax rules. If there are syntax errors in the **XAPPL** file, Studio will not load the file.

In this section, attribute values are shown in parenthesis after their description and default values are shown in **bold**.

# XAPPL Configuration Elements and Attributes

**OutputLocation**

- The **outputlocation** attribute is the path to the folder where the virtual application executable will be created. This can be a local path, a UNC path, or a mapped drive.

**OutputFile**

- The **outputfile** attribute is the file name of the virtual application executable.

**Project Type**

- The **project type** attribute denotes whether this configuration is for a virtual application (Application) or an **SVM** (Component).

**Licensing**

- The **licensing** attribute contains information about the license that was used to build the virtual application.

**Output**

- The **diagnosticMode** attribute denotes when the application output should log diagnostic information (**True**) or not (**False**). If true, the virtual application will create diagnostic logs in the directory where it was executed from.
- The **sourcePackage** attribute is not used.

**MSI**
All sub-elements contain settings pertaining to the configuration of the MSI setup file.

- The **outputMsiPath** attribute indicates the location where the setup MSI will be built.
- The **title** attribute indicates the value of the MSI title property.
- The **subject** attribute indicates the value of the MSI subject property.
- The **keywords** attribute indicates the value of the MSI keywords property.
- The **productName** attribute indicates the value of the MSI product name property.
- The **productVersion** attribute indicates the value of the MSI product version property.
- The **manufacturer** attribute indicates the value of the MSI manufacturer property.
- The **productLanguage** attribute indicates the value of the MSI product language property.
- The **author** attribute indicates the value of the MSI author property.
- The **description** attribute indicates the value of the MSI description property.
- The **manufacturerUrl** attribute indicates the value of the MSI manufacturer URL property.
- The **autoBuild** attribute denotes whether the MSI should build when the virtual application build completes successfully (**True**) or not ( **False**).
- The **isolatePerUser** attribute denotes whether the MSI setup should be installed on a per-user basis (**True**) or installed for all users ( **False**). When installing per-user, the install root path is Application Data. When installing for all users, the install root path is **Program Files**.
- The **applicationFolder** attribute indicates the subfolders into which the virtual application should be installed (**Company Name\Product Name**).
- The **upgradePreviousVersion** attribute denotes whether the setup should maintain the same Upgrade code when it builds (**True**) or change the Upgrade code for each build (**False**). This allows the setup to upgrade previous versions when it is installed, or to exist side by side.
- The **productCode** attribute indicates the value of MSI product code property.
- The **upgradeCode** attribute indicates the value of MSI upgrade code property.
- The **componentId** attribute indicates the value of the MSI component id property.

**Packages**
All sub-elements contain settings pertaining to the configuration of the packages included in the virtual application.

**Clr**
The .NET *Clr* runtime element and all sub-elements contain settings pertaining to the configuration of the virtual .NET Framework runtime.

**Direct X**
The DirectX element and all sub-elements contain settings pertaining to the configuration of the virtual DirectX runtime.

**Java**
All sub-elements contain settings pertaining to the configuration of the virtual java runtime.

**RunTime**

- The **name** attribute indicates the name of the java runtime (Java).
- The **platform** attribute indicates the platform that the java runtime is designed for (x86).
- The **version** attribute indicates the version of the java runtime. The available versions are Java 5 (1.5.0.140) and Java 6 (1.6.0.30).

**Settings**

- The **startupType** attribute denotes whether to use the jar file (JAR) or class path (Class) command line parameters for java.exe to launch the application.
- The **startup** attribute indicates the jar file path or class name depending on the StartupType.
- The **classpath** attribute indicates the path to the class files of the Java runtime.
- The **options** attribute denotes any additional command line parameter.

**Package**

- The **name** attribute indicates the name of the component or runtime.
- The **platform** attribute indicates the platforms that the component or runtime is supported on. The following are the only available values:
  - **Any platform (Any)**
  - **x86 platform (x86)**
- The **version** attribute indicates the version of the component or runtime.

**Virtualization Settings**
All sub-elements contain settings pertaining to the configuration of the virtual operating system.

- The **suppressBranding** attribute controls the branding pop-up that is displayed (**False**), or not displayed (**True**) in the lower right-hand corner during application startup.
- The **isolateWindowClasses** attribute is used to isolate windows classes, as registered via the **Windows ::RegisterClass** or **::RegisterClassEx** APIs. For example, this allows a virtualized Firefox instance to run while a non-virtualized instance is running.
- The **readOnlyVirtualization** attribute denotes whether the virtual application has the ability to modify virtual files and registry settings ( **False**) or not (**True**). Setting this attribute to **True** will prevent modification to the virtual filesystem and virtual registry.
- The **disableXenocodeCommandLine** attribute controls the ability to execute (**False**) any file from within the virtual filesystem.
- The **subsystem** attribute indicates the application output type. It can be inherited from the startup file (**Inherit**) or set explicitly to be a Windows application (**GUI**) or console application (**Console**). If **Inherit** is set, but the startup file is either not in the virtual filesystem or not an executable, then the output will be a Windows application.
- The **ie6Emulation** attribute denotes a special mode required for the Internet Explorer 6 template (**True**). For all other apps, this should be disabled (**False**).
- The **sandboxPath** attribute indicates the base path of the application sandbox **(@APPDATALOCAL@\Spoon\Sandbox\@TITLE@\@VERSION@)**.
- The **workingDirectory** attribute defines what directory the application will run in.
- The **compressPayload** attribute controls whether the output executable will be compressed (**True**) or not (**False**).
- The **trimUACManifest** attribute removes items from the manifest that may require elevation (**True**).
- The **enableDRMCompatibility** attribute ensures compatibility (**True**) with applications protected by software formerly known as "Armadillo" and other DRM software.
- The **deleteSandbox** attribute will cause the sandbox to be reset automatically when the virtual application is shutdown (**True**).
- The **shutdownProcessTree** attribute will cause the all child processes spawned within the virtual environment to be shutdown when the root process exits. By default, the root process is specified by setting the startup file.
- The **exeOptimization** attribute will attempt to launch the startup executable with the initial virtual machine process, preventing the creation of a separate application process (**True**).
- The **enhancedDEPCompatibility** attribute provides compatibility for systems with Data Execution Protection enabled (**True**). This setting is used primarily for virtual applications running on Windows 2003.
- The **notifyProcessStarts** attribute causes a notification to be sent as a debugging output string whenever a new process is started within the virtual environment (**True**).
- The **forceReadShareFiles** attribute forces any file opened by any process within the virtual environment to do so with the READ_SHARE flag set (**True**).
- The **launchChildProcsAsUser** attribute causes all child processes to be provided with the same level of privileges as the virtual machine root process (**True**).
- The **forceIndicateRunningElevated** attribute forces the application to run as if it has elevated security privileges (**True**).
- The **suppressPopups** attribute will prevent an error dialog popup if the virtual application encounters a fatal startup error, and will cause the application to exit silently (**True**).

**ChildProcessVirtualization**

- The **spawnExternalComServers** attribute controls whether the virtual application launches ComServers in the virtual environment ( **False**) or the external environment (**True**).
- The **spawnVm** attribute denotes whether the spawned external applications are spawned inside the virtual environment (**True**) or outside the virtual environment (**False**).

**ChildProcessException**

- The **name** attribute indicates the name of the executable file (extension included) to except from the effects of the **spawnVm** attribute.

**CustomMetadata**
All sub-elements contain settings pertaining to the configuration of the individual custom metadata items.

**CustomMetadataItem**

- The **property** attribute indicates the name of the custom metadata item.
- The **value** attribute indicates the value of the custom metadata item.

**StandardMetadata**
All sub-elements contain settings pertaining to the configuration of the individual standard metadata items.

**StandardMetadataItem**

- The **property** attribute indicates the name of the standard metadata item. The following are the available standard metadata:
  - Product Title (**Title**)
  - Publisher (**Publisher**)
  - Description (**Description**)
  - Website (**Website**)
  - Product Version (**Version**)

**SplashImage**

- The **path** attribute indicates the source path to the splash image displayed at application startup.
- The **transparency** attribute indicates the color in the splash image that should be made transparent when the image is displayed (E.g. **Magenta**).

**StartupFiles**
All sub-elements contain configuration pertaining to the individual startup files.

**StartupFile**

- The **node** attribute indicates the path of the startup file.
- The **tag** attribute indicates the command line trigger used to specify this entry as the startup to use.
- The **commandLine** attribute indicates the command line arguments to pass to the startup file.
- The **default** attribute denotes whether this entry is executed automatically when no tag is specified (**True**) or not (**False**).

**StartupShim**
The startup shim is a virtualized binary that is invoked prior to the startup file. Startup shims are used to perform customized licensing checks or other initialization tasks.

- The **useShim** attribute indicates whether to use the startup shim.
- The **shimDllPath** attribute indicates the path to the virtual shim DLL implementation.
- The **paramOnInitialize** attribute indicates a string to be passed to the shim **OnInitialize** function.
- The startup shim signature is **typedef BOOL (__stdcall *FnOnInititialize) LPCWSTR pwcsInitilizationToken)**. The return value indicates whether virtual machine execution should proceed.

**Layers**
All sub-elements are individual virtual layers.

**Layer**
The **Layer** element and all sub-elements contain settings pertaining to the configuration of this layer of the virtual operating system.

- The **name** attribute indicates the name of the layer. The default layer (**Default**) is the only layer for whom the name matters. All other layer names are purely informational.

**Condition**

- The **variable** attribute indicates the host system setting that will be evaluated. The operating system version (**OS**) is the only available option.
- The **operator** attribute indicates the Boolean operation that will be used to evaluate the host system. The available Boolean operations are:
  - greater than or equal to (**GREATEREQUAL**)
  - greater than (**GREATER**)
  - equal to (**EQUAL**)
  - not equal to (**NOTEQUAL**)
  - less than (**LESS**)
  - less than or equal to (**LESSEQUAL**)
- The **value** attribute indicates the value against which the host system will be evaluated, using the Boolean operation. The available values in ascending order are:
  - Windows 2000 (**Win2k**)
  - Windows XP (**WinXP**)
  - Windows 2003 (**Win2k3**)
  - Windows Vista (**Vista**)

**Filesystem**
All sub-elements contain settings pertaining to the configuration of the virtual filesystem.

**Directory**
All sub-elements contain settings pertaining to the configuration of this directory of the virtual filesystem.

- The **rootType** attribute indicates the root system folder that this virtual folder is mapped to on the host filesystem. Directory elements with the **rootType** attribute are always directly beneath the **Filesystem** element. The following are the available **rootType** values:
  - Application Directory (**Application**)
  - Windows\System32 (**System**)

        - Windows (**OS**)
        - System Drive Root Directory (**SysDrive**)
        - Program Files\Common (**AllProgramsCommon**)
        - Program Files (**AllPrograms**)
        - Current User - Start Menu (**StartMenu**)
        - Current User - Start Menu\Programs (**Programs**)
        - Current User - Start Menu\Programs\Startup (**Startup**)
        - Current User - Application Data (**AppData**)
        - Current User - LocalSetting\Application Data (**AppDataLoca**l)
        - Current User - Desktop (**Desktop**)
        - Current User - Templates (**Templates**)
        - Current User - Favorites (**Favorites**)
        - Current User - Music (**Music**)
        - Current User - Pictures (**Pictures**)
        - Current User - My Documents (**Documents**)
        - %PROFILE% (**Profile**)
        - All Users - Start Menu (**StartMenuCommon**)
        - All Users - Start Menu\Programs (**ProgramsCommon**)
        - All Users - Start Menu\Programs\Startup (**StartupCommon**)
        - All Users - Application Data (**AppDataCommon**)
        - All Users - Desktop (**DesktopCommon**)
        - All Users - Templates (**TemplatesCommon**)
        - All Users - Favorites (**FavoritesCommon**)
        - All Users - Music (**MusicCommon**)
        - All Users - Pictures (**PicturesCommon**)
        - All Users - My Documents (**DocumentsCommon**)
        - %ALLUSERSPROFILE% (**ProfileCommon**)
    - The **isolation** attribute indicates the isolation setting of the virtual folder. The available values are:
        - Full isolation (**Full**)
        - WriteCopy isolation (**WriteCopy**)
        - Merge isolation (**Merge**)
    - The **name** attribute indicates the name of the virtual directory.
    - The **hide** attribute denotes whether the directory is marked as hidden (**True**) or visible (**False**).

**File**

- The **name** attribute indicates the name of the file.
- The **hide** attribute denotes whether the file is marked as hidden (**True**) or visible (**False**).
- The **source** attribute indicates the source path to the file.

**Registry**
All sub-elements contain settings pertaining to the configuration of the virtual registry.

**Key**
All sub-elements contain settings pertaining to the configuration of this key of the virtual filesystem.

- The **rootType** attribute indicates the root system folder that this virtual folder is mapped to on the host filesystem. Key elements with the **rootType** attribute are always directly beneath the **Registry** element. The following are the available **rootType** values:
    - HKEY_CLASSES (**ClassesRoot**)
    - HKEY_CURRENT_USER (**CurrentUser**)
    - HKEY_LOCAL_MACHINE (**CurrentUser**)
    - HKEY_USERS (**Users**)
- The **name** attribute indicates the name of the key.
- The **namePathInformationTuples** indicates that there is a path in the name or value of the registry item. There are 3 comma delimited integers for each path found in the name/value.
    1. Flags that indicate the state of the path (valid combinations: 0x0, 0x1, 0x2, 0x4, 0x5, 0x6)
        - 0x1 – All Uppercase
        - 0x2 – All Lowercase
        - 0x4 – Uses Short Path Names
    2. Start index of the path
    3. Length of the path
- The **isolation** attribute indicates the isolation setting of the virtual folder. The available values are:
    - Full isolation (**Full**)
    - Merge isolation (**Merge**)

**Value**

- The **name** attribute indicates the name of the value.
- The **type** attribute indicates the type of the value. The available values are:
    - REG_SZ and REG_EXPAND_SZ (**String**)
    - REG_DWORD (**DWORD**)
    - REG_QWORD (**QWORD**)
    - REB_BINARY (**Binary**)

    - REG_MULTI_STRING (**StringArray**)
- The **namePathInformationTuples** indicates that there is a path in the name or value of the registry item. There are 3 comma delimited integers for each path found in the name/value.
    1. Flags that indicate the state of the path (valid combinations: 0x0, 0x1, 0x2, 0x4, 0x5, 0x6)
        - 0x1 – All Uppercase
        - 0x2 – All Lowercase
        - 0x4 – Uses Short Path Names
    2. Start index of the path
    3. Length of the path
- The **value** attribute indicates the value of the value. This is true for all types, except **StringArray**, which contains the String sub-element.

**Environment Variables**

- The **name** attribute indicates the name of the environment variable.
- The **value** attribute indicates the value of the environment variable.

**Services**

- The **name** attribute indicates the name of the windows service.
- The **autoStart** attribute denotes whether the windows service starts when the virtual application starts (**True**) or not (**False**).
- The **commandLine** attribute indicates the startup command line of the windows service.
- The **friendlyName** attribute indicates the friendly name of the windows service.
- The **description** attribute indicates the description of the windows service.
- The **objectName** attribute indicates the account under which the windows service ran when not virtualized.
- The **keepAlive** attribute denotes whether the windows service should continue running after the startup application has closed (**True**) or not (**False**).
- The **start** attribute indicates the value of the **Start DWORD** value in the Windows Services registry key.
- The **type** attribute indicates the value of the **Type DWORD** value in the Windows Services registry key.
- The **errorControl** attribute indicates the value of the **ErrorControl DWORD** value in the Services registry key.

**Shortcuts**
All sub-elements contain settings pertaining to the configuration of the MSI shortcuts.

**Folder**
All sub-elements contain settings pertaining to the configuration of the MSI shortcuts in this folder.

- The **name** attribute indicates the name of the folder. The two top level folders represent the Desktop (**Desktop**) and the Programs menu on the Start menu (**Programs Menu**).

**Shortcut**

- The **name** attribute indicates the name of the shortcut.
- The **targetPath** attribute indicates the path of the StartupFile that is the target of the shortcut.
- The **targetParameter** attribute indicates the Trigger or Tag of the StartupFile that is the target of the shortcut.
- The **arguments** attribute indicates the arguments passed to the target of the shortcut at runtime.
- The **showCmd** attribute denotes whether the application should start in a maximized(**3**), minimized(**7**) or regular(**1**) window state.
- The **description** attribute indicates the description of the shortcut.

**IconResource**

- The **IconResource** sub-element contains an identifier of the icon that is used for the Shortcut.

**ProgIds**
All sub-elements contain settings pertaining to the configuration of the ProgId.

- The **name** attribute indicates the name of the ProgId.
- The **description** attribute indicates the description of the ProgId.

**IconResource**

- The **IconResource** sub-element contains an identifier of the icon that is used for the file association.

**Extension**
All sub-elements contain settings pertaining to the configuration of the file extensions for the ProgId.

- The **extension** attribute indicates the file extension that is associated with the ProgId.
- The **mimeType** attribute indicates the MIME type of all files with the extension.

**Verb**
All sub-elements contain settings pertaining to the configuration of the Verb for the file extension.

- The **title** attribute indicates the title of the verb.
- The **verb** attribute indicates the verb value.
- The **arguments** attribute indicates the arguments passed to the target of the verb at runtime.

- The **default** attribute denotes whether this verb is the default verb (**True**) or not (**False**).

# Troubleshooting

This section describes the most common configuration errors that occur when using Studio.

If you encounter a problem with a virtual application, please carefully read this section or query the online knowledge base before using other support options. It is very likely that the issue you have encountered is addressed in one of these places.

## Problems accessing Internet-based resources

Several Studio features require access to Internet-based resources in order to function properly. These features may be unavailable if Studio is unable to connect to the Internet.

In many corporate environments, access to the Internet is filtered through a firewall or proxy server. In these cases, Studio will attempt to automatically configure itself for Internet access based on the system Internet settings. In some cases, however, it may be necessary to manually configure the proxy server settings.

To configure proxy server settings:

- Click the **Proxy Settings...** option on the **Options** menu. This displays the **Proxy Settings** dialog.
- Enter appropriate proxy server settings in the dialog. It may be necessary for you to consult your system administrator to obtain your organization's proxy server settings.

## Generating diagnostic-mode virtual applications

Occasionally, errors during virtual application configuration result in an executable which fails to run properly; that is, to exactly emulate the behavior of the original source application. Usually this is a result of an error in the virtualization configuration, such as a missing file or registry entry.

To assist in diagnosis of these problems, Studio offers the option of creating *diagnostic-mode executables*. Diagnostic-mode executables generate logging data during execution that can assist in diagnosis of problems related to virtualization.

To generate a diagnostic-mode executable, check the **Generate diagnostic-mode executable** option on the **Output** section of the **Virtual Application** ribbon bar. Then click **Build** to generate the instrumented executable.

Execution of the instrumented executable will generate a **xclog_<id>.txt** file in the application startup directory that contains detailed diagnostic data gathered during execution. Inspection of this file, particularly of entries labeled **WARNING** or **ERROR**, often allows diagnosis of virtualization errors. If you require assistance from Spoon technical support to resolve your problem, we strongly encourage you to submit this information along with your support request to facilitate resolution of your issue.

Note: Because diagnostic-mode executables run significantly slower than standard executables and generate very large log files, diagnostic-mode executables should not be distributed to your end-users.

# Thank you for using Spoon Studio!

We hope you enjoy using Spoon. Please let us know any way we can improve your Spoon experience.

*- The Spoon Team*