

APP-V 5 SP2 APPLICATION PUBLISHING AND CLIENT INTERACTION



©2014 Microsoft Corporation. All rights reserved. This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples are for illustration only and are fictitious. No real association is intended or inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

Contents

Overview	4
Intended Audience	4
Prerequisite Knowledge	4
App-V Files	5
App-V Package Files	5
APPV File	6
App-V Client Data Storage Locations	7
Package Store	8
Shared Content Store	8
Package Catalogs	8
Machine Catalog	8
User Catalog	9
Shortcut Backups	9
Copy on Write Files	10
Package Registry	11
Package Registry Staging vs. Connection Group Registry Staging	11
Virtual Registry	12
Registry Locations	12
App-V Package Store Behavior	15
Add Packages	15
Mounting Packages	15
Streaming Packages	15
Background Streaming	16
Optimized Streaming	16
Stream Faults	16
Package Upgrades	16
Package Removal	16
Roaming Registry and Data	17
Registry Based Data	17
App-V and Folder Redirection	17
App-V Client Application Lifecycle Management	20
Publishing Refresh	20

Adding an App-V Package	21
Publishing an App-V Package	24
Application Launch.....	26
Upgrading an App-V Package.....	28
Upgrading an in use App-V Package	29
Removing an App-V Packages	29
Repairing an App-V Package	30
Integration of App-V Packages.....	31
Rules of Integration.....	31
Extension Points.....	32
Shortcut.....	32
File Type Associations	33
Shell Extensions.....	34
COM	35
Software Clients and Application Capabilities	35
URL Protocol handler	36
AppPath.....	36
Virtual Application	37
Extension Point Rules.....	37
Dynamic Configuration Processing	38
Example for Dynamic Configuration Files	38
Side by Side Assemblies	41
Automatic publishing of SxS assemblies	41
Client Logging.....	42
Conclusion.....	43

Overview

The App-V 5 SP2 Application Publishing and Client Interaction whitepaper enables App-V administrators and sequencers to gain a better understanding of how the App-V Client processes packages and presents them to users. This document provides details around typical client operations with important locations for data storage, how the publishing refresh process works, and the available integration points with the local operating system.

The App-V client performs many tasks to present virtual applications that work more like traditionally installed applications. Understanding the process of publishing and running applications is an important part of implementing a successful App-V solution. Often, these tasks do not receive the same level of attention as the delivery methods, but may be more important. To effectively manage you must understand the functions performed by the App-V client, and how these processes affect the local operating system.

This document provides information about the following topics:

- App-V Files and Data Storage Locations
- Package Registry
- App-V Package Store Behavior
- Roaming Registry and Data
- App-V Client Application Lifecycle Management
- Integration of App-V Packages
- Dynamic Configuration
- Side by Side Assemblies
- Client Logging

Intended Audience

This document is intended for App-V administrators and provides a better understanding of client operations to assist with managing your App-V environment.

Prerequisite Knowledge

This document assumes an understanding of App-V infrastructures and concepts. In order to gain a greater command of App-V knowledge and better understanding of this document, please refer to the App-V Document Resources Download Page located at: <http://www.microsoft.com/en-us/download/details.aspx?id=27760> and the App-V 5 Administrators Guide at: <http://technet.microsoft.com/en-us/library/jj713487.aspx>.

App-V Files

The App-V Sequencer monitors software installation and creates a package. The package is comprised of several files that have specific functionality. The primary package file is the APPV file which contains the captured assets and state information. The additional files provide custom integration information for publishing applications, sequencing detailed reporting, and optionally sequencing templates and package accelerators.

The AppV file contains the captured files and state from the sequencing process in a single file. This file includes architecture of the package file, publishing information, and registry in a tokenized form that can be reapplied to a machine and to a specific user upon delivery.

For information on creation of App-V Packages review the Microsoft Application Virtualization 5.0 Sequencing Guide available at: www.microsoft.com/en-us/download/details.aspx?id=27760









App-V Package Files

The Sequencer creates App-V Packages and produces a virtualized application comprised of several files. The following is a description of each of the files created.

File	Description
.APPV	The Virtual Application Package file containing all assets and state organized and divided into logical blocks.
.MSI	Executable deployment wrapper allowing the manual deployment of .APPV files or deployment via existing third-party deployment platforms.
_DeploymentConfig.XML	Used for customizing the default publishing parameters for all applications in a package.
_UserConfig.XML	Used for customizing the publishing parameters directed to specific users for all applications in a package.
Report.xml	Summary of sequencing messages including omitted drivers, files, and registry locations.
.CAB	<i>Optional:</i> Package Accelerator file used to automatically rebuild a previously sequenced virtual application package.
.APPVT	<i>Optional:</i> Sequencer Template file used to retain commonly re-used sequencer settings.

APPV File

The AppV file format is a container built from the specifications of the AppX format, based on the Open Packaging Conventions (OPC) standard, and used to store a combination of XML and non-XML files together in a single entity. The AppV file contents can be viewed by renaming the file to a ZIP extension and exploring its contents. The following should be present in the AppV file:

Name	Type
 Root	File folder
 [Content_Types].xml	XML File
 AppxBlockMap.xml	XML File
 AppxManifest.xml	XML File
 FilesystemMetadata.xml	XML File
 PackageHistory.xml	XML File
 Registry.dat	DAT File
 StreamMap.xml	XML File

Note: It is recommend to make a copy of the package prior to renaming and exploring. Editing the App-V package directly with this method is not supported.

This folder and included files are utilized during virtual application addition and publishing, and will be covered in more detail. The following is a brief description of each file.

- **Root:** Directory containing the file system for the virtualized application that was captured during sequencing.
- **[Content_Types].xml:** Identifies the core content types in the AppV file (e.g. DLL, EXE, BIN)
- **AppxBlockMap.xml:** Contains the layout of the AppV file utilizing File, Block, and BlockMap elements that enable location and validation of files in the App-V package.
- **AppxManifest.xml:** Metadata for the package that contains required information for adding, publishing, launching the package. Includes the names and GUIDs associated with the package, as well as extension points (file type associations and shortcuts).
- **FilesystemMetadata.xml:** Contains a list of the files captured during sequencing including attributes (e.g. Directories, Files, Opaque Directories, Empty Directories, Long and short names).
- **PackageHistory.xml:** Information about the sequencing machine (OS version, IE version, .Net framework version) and process (Upgrade, Package version).
- **Registry.dat:** Registry keys and values captured during the sequencing process for the package.
- **StreamMap.xml:** Contains the list of files for the Primary and publishing feature block. The publishing feature block contains the ICO files and required portions of files (EXE and DLL) for publishing the package. The Primary Feature Block, when present, includes files optimized for streaming during the sequencing process.

App-V Client Data Storage Locations

The App-V client performs a number of tasks to ensure the virtual applications run properly and have the appearance of traditionally installed applications. The process of opening and running virtual applications requires mapping from the virtual file system and registry to ensure the application has the required components of a traditional application expected by users. This section focuses on where App-V stores the assets required to run virtual applications, and provides a description of these assets that will be used throughout the document. The following table provides a detailed list of locations where App-V stores data. The following location names are defined in this section and will be used throughout the document.

Table 1: App-V File Locations

Name	Location	Description
Package Store	%ProgramData%\App-V	Default location for read only package files
Machine Catalog	%ProgramData%\Microsoft\AppV\Client\Catalog	Contains per machine configuration documents
User Catalog	%AppData%\Microsoft\AppV\Client\Catalog	Contains per user configuration documents
Shortcut Backups	%AppData%\Microsoft\AppV\Client\Integration\ShortCutBackups	Stores previous integration points that enable restore on package unpublish
Copy on Write (COW) Roaming	%AppData%\Microsoft\AppV\Client\VFS	Writeable roaming location for package modification
Copy on Write (COW) Local	%LocalAppData%\Microsoft\AppV\Client\VFS	Writeable non-roaming location for package modification
Machine Registry	HKLM\Software\Microsoft\AppV	Contains package state information including VReg for machine or globally published packages (Machine hive)
User Registry	HKCU\Software\Microsoft\AppV	Contains user package state information including VReg
User Registry Classes	HKCU\Software\Classes\AppV	Contains additional user package state information

Additional details for the table are provided in the section below and throughout the document.

Package Store

The App-V Client manages the applications assets mounted in the package store. This location is, by default, stored at %ProgramData%\App-V, but is configurable during setup or post setup with the **Set-AppVClientConfiguration** PowerShell® command which modifies the local registry (**PackageInstallationRoot** value under the **HKLM\Software\Microsoft\AppV\Client\Streaming** key). The package store must be located at a local path on the client operating system. The individual packages are stored in the package store in subdirectories named for the Package GUID and Version GUID. An example of a path to a specific application is:

C:\ProgramData\App-V\PackGUID\VersionGUID

Change the default location of the Package Store during setup following the guidance at:

<http://technet.microsoft.com/en-us/library/jj713460.aspx>.

Shared Content Store

When the App-V Client is configured in Shared Content Store mode no data is written to disk when a Stream Fault occurs. Therefore local disk space taken by the packages is minimal (publishing data). This is highly desirable in VDI environments where local storage can be limited and streaming the applications from a highly performing network location (such as a SAN) is preferable. For more information on shared content store mode, review:

<http://blogs.technet.com/b/appv/archive/2013/07/22/shared-content-store-in-microsoft-app-v-5-0-behind-the-scenes.aspx>

Note: The Machine and User Catalogs must be located on a local drive even when using Shared Content Store configurations for the App-V Client.

Package Catalogs

The App-V Client manages the following two file-based locations:

- Catalogs (user and machine).
- Registry locations, depending on how the package is targeted for publishing. There is a Catalog (data store) for the computer and one for each individual user. The Machine Catalog stores global information applicable to all users or any user, and the User Catalog stores information applicable to a specific user. The Catalog is a collection of Dynamic Configurations and manifest files; there is discrete data both file and registry per package version.

Machine Catalog

The machine catalog is stored in %ProgramData% by default, but is not the same location as the Package Store. The Package Store is the golden or pristine copy of the package files. The machine catalog is located by default at %programdata%\Microsoft\AppV\Client\Catalog\ and will include the following files:

- Manifest.xml
- DeploymentConfiguration.xml
- UserManifest.xml (Globally Published Package)
- UserDeploymentConfiguration.xml (Globally Published Package)

If a package is part of a connection group the machine catalog is created at the following location in addition to the specific package location above at
%programdata%\Microsoft\AppV\Client\Catalog\PackageGroups\ConGroupGUID\ConGroupVerGUID
and includes the following files:

- PackageGroupDescriptor.xml
- UserPackageGroupDescriptor.xml (Globally published Connection Group)

The Machine Catalog stores package documents that are available to users on the machine, when packages are added and published. However, if a package is “global” at publishing time, the integrations are available to all users. If a package is non-global, the integrations are only published for specific users, but there are still global resources that are modified and visible to anyone on the machine (e.g. the package directory is in a shared disk location).

If a package is available to a user on the computer (global or non-global), the manifest is stored in the Machine Catalog. When a package is published globally, there is a Dynamic Configuration file, stored in the Machine Catalog; therefore, the determination of whether a package is global or not is defined as whether there is a policy file (UserDeploymentConfiguration file) in the Machine Catalog.

User Catalog

The User Catalog is created during the publishing process and contains information utilized for publishing the package, and also at launch to ensure that a package is provisioned to a specific user. The user catalog is created in a roaming location and includes user specific publishing information at appdata\roaming\Microsoft\AppV\Client\Catalog\Packages\PkgGUID\VerGUID, in the following files:

- UserManifest.xml
- DynamicConfiguration.xml or UserDeploymentConfiguration.xml

If a package is part of a connection group, the user catalog is created at the following location, in addition to the specific package location above at
appdata\roaming\Microsoft\AppV\Client\Catalog\PackageGroups\PkgGroupGUID\PkgGroupVerGUID
and includes the following file:

- UserPackageGroupDescriptor.xml

When a package is published for a user, the policy file is stored in the User Catalog. At the same time, a copy of the manifest is also stored in the User Catalog. When a package entitlement is removed for a user, the relevant package files are removed from the User Catalog. Looking at the user catalog, an administrator can view the presence of a Dynamic Configuration file, which indicates the package is entitled for that user.

For roaming users, the User Catalog needs to be in a roaming or shared location to preserve the legacy App-V behavior of targeting users by default (a key customer scenario). Entitlement and policy are tied to a user, not a machine, so once provisioned they should roam with the user.

Shortcut Backups

During the publishing process the App-V Client backs up any shortcuts and integration points to %AppData%\Microsoft\AppV\Client\Integration\ShortCutBackups. This enables the restoration of these integration points when the package is unpublished, to the previous versions.

Copy on Write Files

The Package Store contains a pristine copy of the package files that have been streamed from the publishing server. During normal operation of an App-V application the user or service may require modification of files. These modifications are not made in the package store to preserve the capability of repairing the application, which removes these changes. These locations, called Copy on Write (COW), support both roaming and non-roaming locations. The location the modifications are stored depends on where the application has been programmed to write changes in a native experience.

COW Roaming

The COW Roaming location as described above stores changes to files and directories that are destined to the typical %AppData% location or \Users\{username}\AppData\Roaming location. These directories and files are then roamed based on the operating system settings.

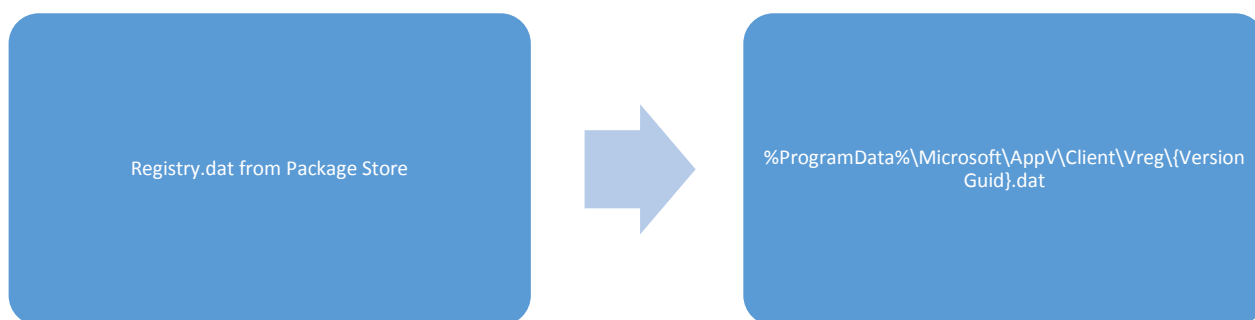
COW Local

The COW Local location is similar to the roaming location, but the directories and files are not roamed to other computers, even if roaming support has been configured. The COW Local location described above stores changes applicable to typical windows and not the %AppData% location. The directories listed will vary but there will be two locations for any typical Windows locations (e.g. Common AppData and Common AppDataS). The **S** signifies the restricted location when the virtual service requests the change as a different elevated user from the logged on users. The non-**S** location stores user based changes.

Package Registry

Before an application can access the package registry data, the App-V Client must make the package registry data available to the applications. The App-V Client uses the real registry as a backing store for all registry data.

When a new package is added to the App-V Client, a copy of the REGISTRY.DAT file from the package is created at **%ProgramData%\Microsoft\AppV\Client\VREG\{Version GUID}.dat**. The name of the file is the version GUID with the .DAT extension. The reason this copy is made is to ensure that the actual hive file in the package is never in use, which would prevent the removal of the package at a later time.



When the first application from the package is launched on the client, the client stages or copies the contents out of the hive file, recreating the package registry data in an alternate location **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AppV\Client\Packages\PackageGuid\Versions\VersionGuid\REGISTRY**. The staged registry data has two distinct types of machine data and user data. Machine data is shared across all users on the machine. User data is staged for each user to a user-specific location **HKCU\Software\Microsoft\AppV\Client\Packages\PackageGuid\Registry\User**. The machine data is ultimately removed at package removal time, and the user data is removed on a user unpublish operation.

Package Registry Staging vs. Connection Group Registry Staging

When connection groups are present, the previous process of staging the registry holds true, but instead of having one hive file to process, there are more than one. The files are processed in the order in which they appear in the connection group XML, with the first writer winning any conflicts.

The staged registry persists the same way as in the single package case. Staged user registry data remains for the connection group until it is disabled; staged machine registry data is removed on connection group removal.

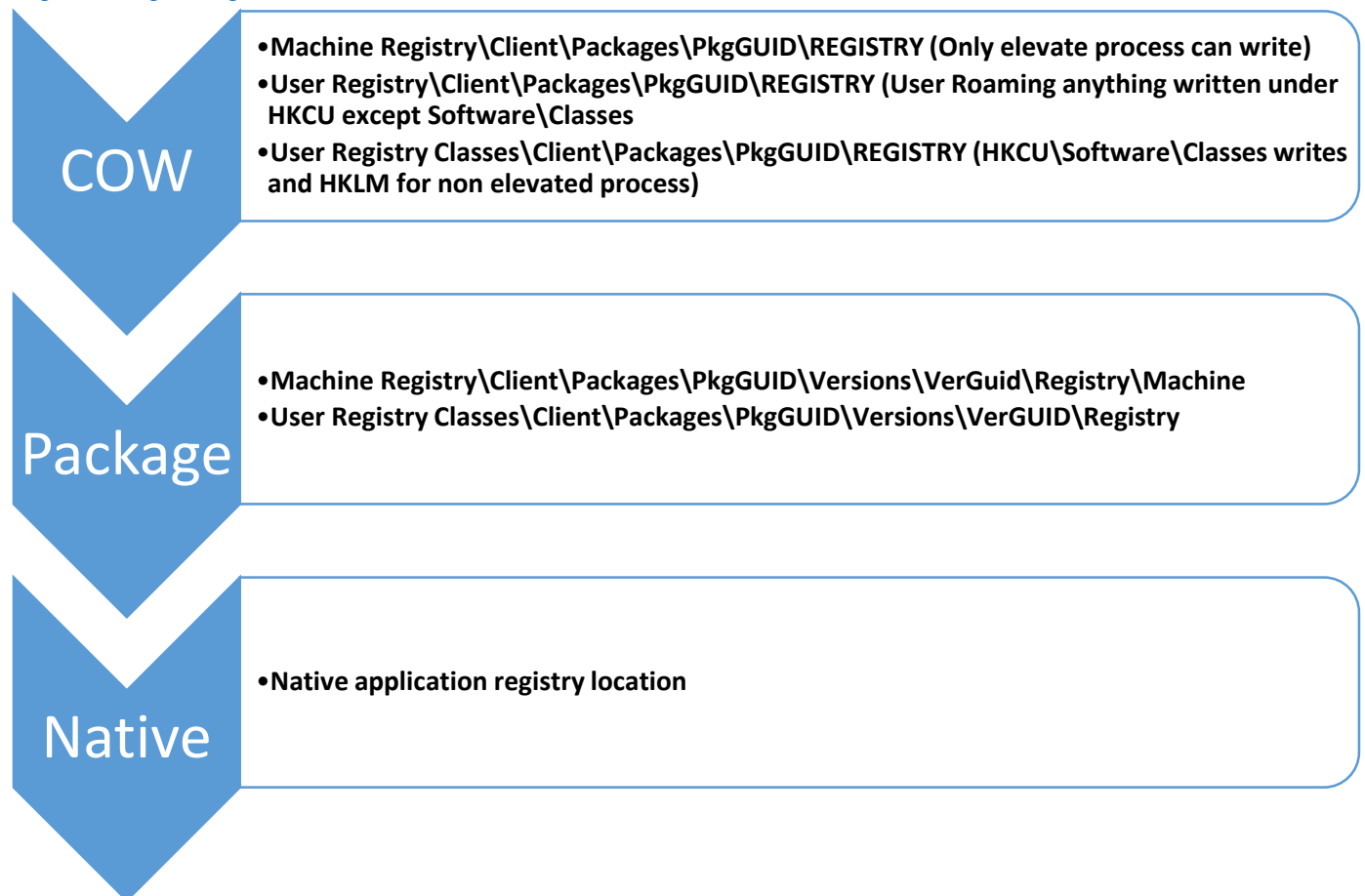
Virtual Registry

The purpose of the virtual registry (VREG) is to provide a single merged view of the package registry and the native registry to applications. It also provides *copy-on-write* (COW) functionality – that is any changes made to the registry from the context of a virtual process are made to a separate COW location. This means that the VREG must combine up to three separate registry locations into a single view based on the populated locations in the registry *COW -> package -> native*. When a request is made for a registry data it will locate in order until it finds the data it was requesting. Meaning if there is a value stored in a COW location it will not proceed to other locations, however, if there is no data in the COW location it will proceed to the Package and then Native location until it finds the appropriate data.

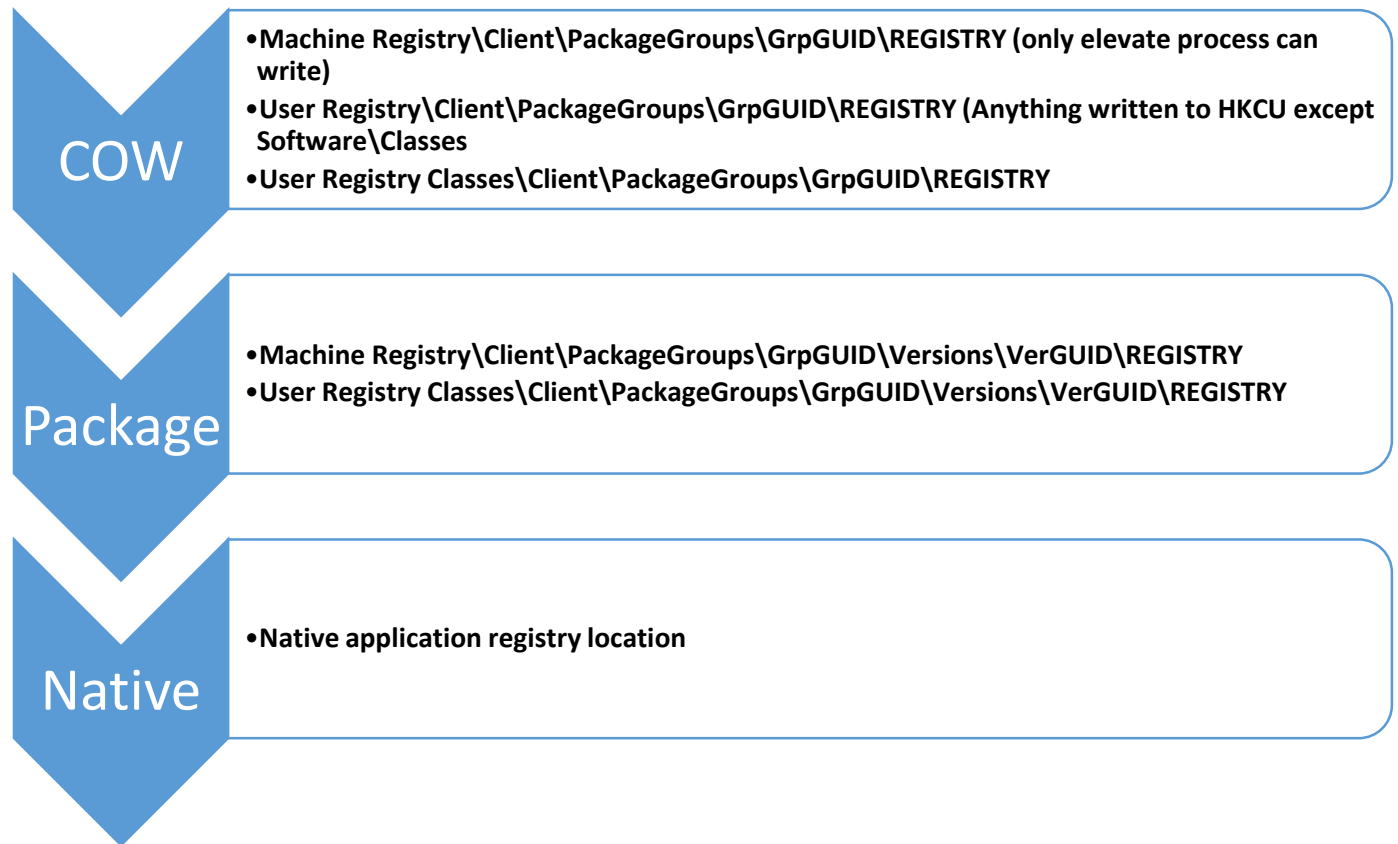
Registry Locations

There are two package registry locations and two connection group locations where the App-V Client stores registry information, depending on whether the Package is published individually or as part of a connection group. There are three COW locations for packages and three for connection groups, which are created and managed by the VREG. Settings for packages and connection groups are not shared:

Single Package VReg:



Connection Group VReg:



There are two COW locations for HKLM; elevated and non-elevated processes. Elevated processes always write HKLM changes to the secure COW under HKLM. Non-elevated processes always write HKLM changes to the non-secure COW under HKCU\Software\Classes. When an application reads changes from HKLM, elevated processes will read changes from the secure COW under HKLM. Non-elevated reads from *both*, favoring the changes made in the unsecure COW first.

Pass-through Keys

Pass-through keys enable an administrator to configure certain keys so they can only be read from the native registry, bypassing the Package and COW locations. Pass-through locations are global to the machine (not package specific) and can be configured by adding the path to the key, which should be treated as pass-through to the **REG_MULTI_SZ** value called **PassThroughPaths** of the key **HKLM\Software\Microsoft\AppV\Subsystem\VirtualRegistry**. Any key that appears under this multi-string value (and their children) will be treated as pass-through.

The following locations are configured as pass-through locations by default:

- HKEY_CURRENT_USER\SOFTWARE\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppModel
- HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppModel
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\eventlog\Application
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\WMI\Autologger
- HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
- HKEY_LOCAL_MACHINE\SOFTWARE\Policies
- HKEY_CURRENT_USER\SOFTWARE\Policies

The purpose of Pass-through keys is to ensure that a virtual application does not write registry data in the VReg that is required for non-virtual applications for successful operation or integration. The Policies key ensures that Group Policy based settings set by the administrator are utilized and not per package settings. The AppModel key is required for integration with Windows Modern UI based applications. It is recommend that administers do not modify any of the default pass-through keys, but in some instances, based on application behavior may require adding additional pass-through keys.

App-V Package Store Behavior

App-V 5 manages the Package Store, which is the location where the expanded asset files from the APPV file are stored. By default, this location is stored at %ProgramData%\App-V, and is limited in terms of storage capabilities only by free disk space. The package store is organized by the GUIDs for the package and version as mentioned in the previous section.

Add Packages

App-V Packages are staged upon addition to the computer with the App-V Client. The App-V Client provides on-demand staging. During publishing or a manual Add-AppVClientPackage, the data structure is built in the package store (c:\programdata\App-V\{PkgGUID}\{VerGUID}). The package files identified in the publishing block defined in the StreamMap.xml are added to the system and the top level folders and child files staged to ensure proper application assets exist at launch.

Mounting Packages

Packages can be explicitly loaded using the PowerShell **Mount-AppVClientPackage** or by using the **App-V Client UI** to download a package. This operation completely loads the entire package into the package store.

Streaming Packages

The App-V Client can be configured to change the default behavior of streaming. All streaming policies are stored under the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\APPV\CLIENT\STREAMING. Policies are set using the PowerShell cmdlet **Set-AppvClientConfiguration**. The following policies apply to Streaming:

AllowHighCostLaunch – On Windows 8 it allows streaming over 3G and cellular networks

AutoLoad – Specifies the Background Load setting:

0 – Disabled

1 – Previously Used Packages only

2 – All Packages

PackageInstallationRoot – The root folder for the package store in the local machine

PackageSourceRoot – The root override where packages should be streamed from

SharedContentStoreMode – Enables the use of Shared Content Store for VDI scenarios

These settings affect the behavior of streaming App-V package assets to the client. By default, App-V only downloads the assets required after downloading the initial publishing and primary feature blocks. There are three specific behaviors around streaming packages that must be explained:

- Background Streaming
- Optimized Streaming
- Stream Faults

Background Streaming

The PowerShell cmdlet **Get-AppvClientConfiguration** can be used to determine the current mode for background streaming with the **AutoLoad** setting and modified with the cmdlet **Set-AppvClientConfiguration** or from the registry (HKLM\SOFTWARE\Microsoft\AppV\ClientStreaming key). Background streaming is a default setting where the Autoload setting is set to download previously used packages. The behavior based on default setting (value=1) downloads App-V data blocks in the background after the application has been launched. This setting can be disabled all together (value=0) or enabled for all packages (value=2), whether they have been launched.

Optimized Streaming

App-V packages can be configured with a primary feature block during sequencing. This setting allows the sequencing engineer to monitor launch files for a specific application, or applications, and mark the blocks of data in the App-V package for streaming at first launch of any application in the package.

Stream Faults

After the initial stream of any publishing data and the primary feature block, requests for additional files perform stream faults. These blocks of data are downloaded to the package store on an as-needed basis. This allows a user to download only a small part of the package, typically enough to launch the package and run normal tasks. All other blocks are downloaded when a user initiates an operation that requires data not currently in the package store.

For more information on App-V Package streaming visit:

<http://blogs.technet.com/b/virtualvibes/archive/2013/02/27/feature-block-0-the-publishing-feature-block.aspx>

Sequencing for streaming optimization is available at:

<http://blogs.technet.com/b/virtualworld/archive/2013/02/27/app-v-5-0-sequencer-package-streaming-optimisation.aspx>

Package Upgrades

App-V Packages require updating throughout the lifecycle of the application. App-V Package upgrades are similar to the package publish operation, as each version will be created in its own PackageRoot location: **%ProgramData%\App-V\{PkgGUID}\{newVerGUID}**. The upgrade operation is optimized by creating hard links to identical- and streamed-files from other versions of the same package.

Package Removal

The behavior of the App-V Client when packages are removed depends on the method used for removal. Using an App-V full infrastructure to unpublish the application, the user catalog files (machine catalog for globally published applications) are removed, but retains the package store location and COW locations. When the PowerShell cmdlet **Remove-AppVClientPackage** is used to remove an App-V Package, the package store location is cleaned. Remember that unpublishing an App-V Package from the Management Server does not perform a Remove operation. Neither operation will remove the Package Store package files.

Roaming Registry and Data

App-V 5 is able to provide a near-native experience when roaming, depending on how the application being used is written. By default, App-V will roam **AppData** stored in the roaming location, based on the roaming configuration of the operating system. Other locations for storage of file-based data do not roam from machine to machine, since they are in locations that are not roamed.

Registry Based Data

App-V registry roaming falls into two scenarios; applications executed as standard users, and applications executed with elevation.

In the first scenario, when a standard user launches an App-V application, both HKLM and HKCU for App-V applications are stored in the HKCU hive on the machine. This presents as two distinct paths:

- HKLM:
HKCU\SOFTWARE\Classes\AppV\Client\Packages\{PkgGUID}\REGISTRY\MACHINE\SOFTWARE
- HKCU:
HKCU\SOFTWARE\Microsoft\AppV\Client\Packages\{PkgGUID}\REGISTRY\USER\{UserSID}\SOFTWARE

The locations are enabled for roaming based on the operating system settings.

For the second scenario, where an application is launched with elevation, the HKLM data is stored in the HKLM hive on the local machine and HKCU data is stored in the User Registry location. This eliminates these settings from being roamed with normal operating system roaming configurations, and the resultant registry keys and values are stored in the following location:

- HKLM\SOFTWARE\Microsoft\AppV\Client\Packages\{PkgGUID}\{UserSID}\REGISTRY\MACHINE\SOFTWARE
- HKCU\SOFTWARE\Microsoft\AppV\Client\Packages\{PkgGUID}\Registry\User\{UserSID}\SOFTWARE

App-V and Folder Redirection

App-V 5.0 SP2 supports **appdata** folder redirection. When the virtual environment is started, the roaming **appdata** state from the user's roaming **appdata** directory is copied to the local cache. Conversely, when the virtual environment is shutdown, the local cache associated with a specific user's roaming **appdata** is transferred to the actual location of that user's roaming **appdata** directory.

A typical package has several locations mapped in the users backing store for settings in both AppData\Local and AppData\Roaming. These locations are the Copy on Write locations that are used to store changes made to the package VFS directories and protecting the default package VFS and stored per user in the user's profile. The table below shows locations, both local and roaming, when Folder Redirection has not been implemented.

VFS directory in package	Mapped location of backing store
ProgramFilesX86	C:\users\jsmith\AppData\Local\Microsoft\AppV\Client\VFS\<GUID>\ProgramFilesX86
SystemX86	C:\users\jsmith\AppData\Local\Microsoft\AppV\Client\VFS\<GUID>\SystemX86
Windows	C:\users\jsmith\AppData\Local\Microsoft\AppV\Client\VFS\<GUID>\Windows
APPV_ROOT	C:\users\jsmith\AppData\Local\Microsoft\AppV\Client\VFS\<GUID>\APPV_ROOT
AppData	C:\users\jsmith\AppData\Roaming\Microsoft\AppV\Client\VFS\<GUID>\AppData

For situations when Folder Redirection has been implemented for %AppData%, the location is redirected, typically to a network location.

VFS directory in package	Mapped location of backing store
ProgramFilesX86	C:\users\jsmith\AppData\Local\Microsoft\AppV\Client\VFS\<GUID>\ProgramFilesX86
SystemX86	C:\users\jsmith\AppData\Local\Microsoft\AppV\Client\VFS\<GUID>\SystemX86
Windows	C:\users\jsmith\AppData\Local\Microsoft\AppV\Client\VFS\<GUID>\Windows
APPV_ROOT	C:\users\jsmith\AppData\Local\Microsoft\AppV\Client\VFS\<GUID>\APPV_ROOT
AppData	\\Fileserver\users\jsmith\roaming\Microsoft\AppV\Client\VFS\<GUID>\AppData

The current App-V Client VFS driver cannot write to network locations, so the App-V Client detects the presence of Folder Redirection and copies the data on the local drive during publishing and on Virtual Environment startup. After the user closes the App-V application and the App-V Client closes the virtual environment the local storage of the VFS AppData is copied back to the network, enabling roaming to additional machines where the process will be repeated. The detailed steps of the processes are:

1. During publishing or virtual environment startup the App-V Client detects the location of the AppData directory.
2. If the roaming AppData path is local or there is not any mapped AppData\Roaming location nothing happens.
3. If the roaming AppData path is not local, the VFS AppData directory will be mapped to the local AppData directory.

This process solves the problem of a non-local %AppData% that is not supported by the App-V Client VFS driver. However the data stored in this new location is not roamed with Folder Redirection. All changes during the execution of the application happen to the local AppData location and must be copied to the redirected location. The detailed steps of this process are:

1. App-V application is shutdown, shutting down the virtual environment.
2. The local cache of the roaming AppData location is compressed and stored in a ZIP file.
3. A timestamp at the end of the ZIP packaging process is used to name the file.
4. The timestamp is recorded in the registry
HKEY_CURRENT_USER\Software\Microsoft\AppV\Client\Packages\<GUID>\AppDataTime as the last known AppData timestamp.
5. The Folder Redirection process is called to evaluate and initiate the ZIP file upload to the roaming AppData directory.

The timestamp is used to determine a last writer wins scenario if there is a conflict and is utilized to optimize the download of the data when the App-V application is published or the virtual environment is started. Folder Redirection will make the data available from any other clients covered by the supporting policy and will initiate the process of storing the AppData\Roaming data to the local AppData location on the client. The detailed processes are:

1. User starts the virtual environment by executing an application.
2. Check the roaming AppData directory for the most recent time stamped ZIP file, if present.
3. Check the registry for the last known uploaded timestamp, if present.
4. When the local last known upload timestamp is greater or equal to the timestamp from the ZIP file, no data is downloaded.
5. If the local last known upload timestamp is less than roaming AppData location the ZIP file is extracted to the local temp directory in the user's profile.
6. After successful extraction of the ZIP file, the local cache of the roaming AppData directory is renamed and the new data is moved into place.
7. The renamed directory is deleted and the application opens with the most recently saved roaming AppData data.

This completes the successful roaming of application settings that are present in AppData\Roaming locations. The only other condition that must be addressed is a package repair operation. The details of the process are:

1. During repair detect if the path to the user's roaming AppData directory is not local.
2. Map the non-local roaming AppData path targets are recreated in both the expected roaming and local AppData locations.
3. Delete the timestamp stored in the registry, if present.

This process will recreate both the local and network locations for AppData and remove the registry record of the timestamp.

App-V Client Application Lifecycle Management

In an App-V Full Infrastructure, after applications are sequenced they are managed and published to users or computers via the App-V Management and Publishing servers. This section details the operations that occur during the common App-V application lifecycle operations (Add, publishing, launch, upgrade, and removal) and the file and registry locations that are changed and modified from the App-V Client perspective. The App-V Client operations are performed as a series of PowerShell commands initiated on the computer running the App-V Client.

This document focuses on App-V Full Infrastructure solutions. For specific information on App-V Integration with Configuration Manager 2012 visit: <http://www.microsoft.com/en-us/download/details.aspx?id=38177>

The App-V application lifecycle tasks are triggered at user login (default), machine startup, or as background timed operations. The settings for the App-V Client operations, including Publishing Servers, refresh intervals, package script enablement, and others, are configured during setup of the client or post-setup with PowerShell commands. See the **How to Deploy the Client** section on TechNet at: <http://technet.microsoft.com/en-us/library/jj713460.aspx> or utilize the PowerShell:

Get additional help for App-V Powershell commands at: <http://blogs.technet.com/b/appv/archive/2012/12/03/app-v-5-0-client-powershell-deep-dive.aspx> or begin by typing the following command from PowerShell

```
get-command *appv*
```

Publishing Refresh

The publishing refresh process is comprised of several smaller operations that are performed on the App-V Client. Since App-V is an application virtualization technology and not a task scheduling technology, the Windows Task Scheduler is utilized to enable the process at user logon, machine startup, and at scheduled intervals. The configuration of the client during setup listed above is the preferred method when distributing the client to a large group of computers with the correct settings. These client settings can be configured with the following PowerShell cmdlets:

- **Add-AppVPublishingServer:** Configures the client with an App-V Publishing Server that provides App-V packages.
- **Set-AppVPublishingServer:** Modifies the current settings for the App-V Publishing Server.
- **Set-AppVClientConfiguration:** Modifies the current settings for the App-V Client.
- **Sync-AppVPublishingServer:** Initiates an App-V Publishing Refresh process manually. This is also utilized in the scheduled tasks created during configuration of the publishing server.

The focus of the following sections is to detail the operations that occur during different phases of an App-V Publishing Refresh. The topics include:

- Adding an App-V Package
- Publishing an App-V Package

Adding an App-V Package

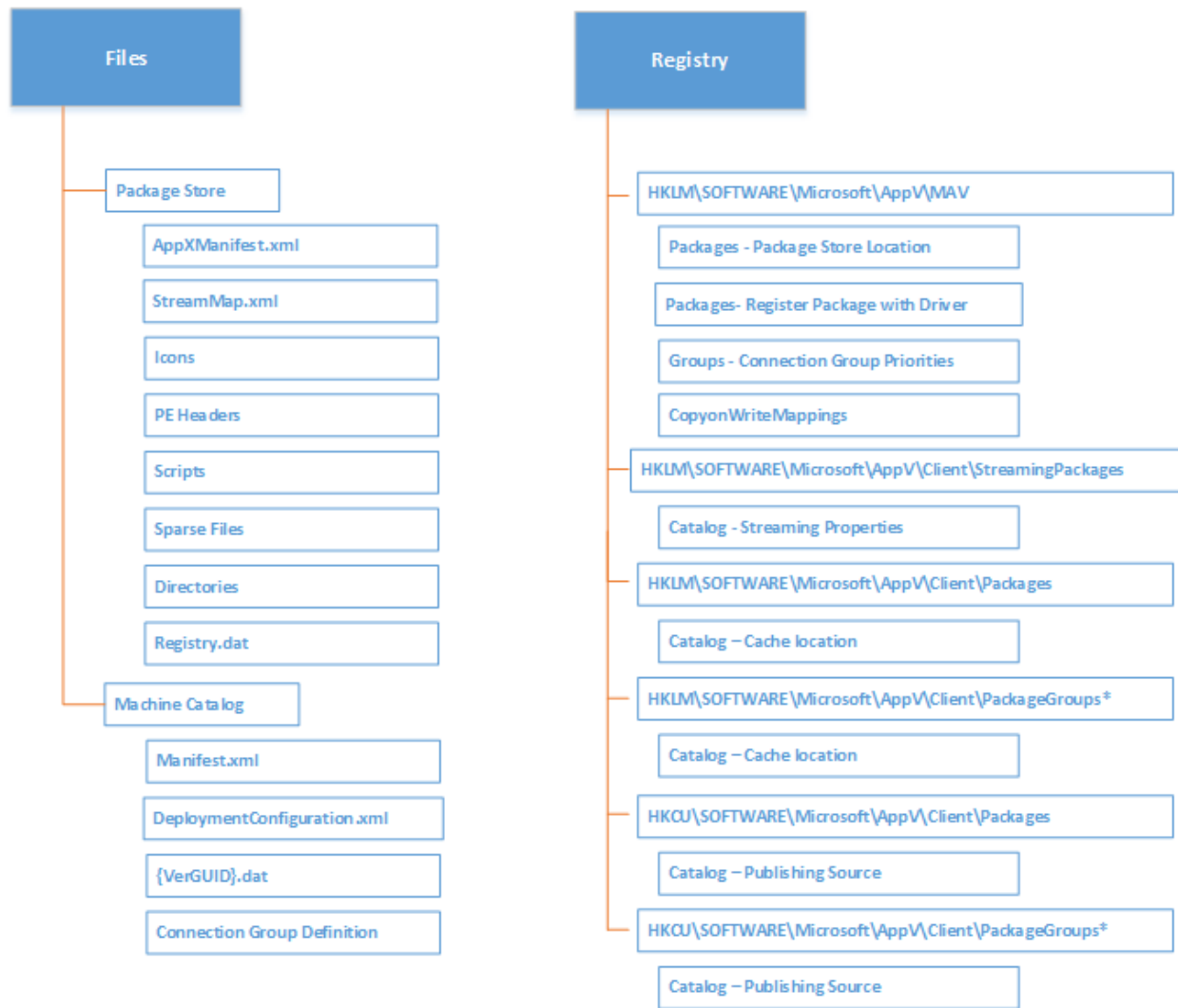
Adding an App-V package to the client is the first step of the publishing refresh process. The end result is the same as the **Add-AppVClientPackage** cmdlet in PowerShell, except during the publishing refresh add process, the configured publishing server is contacted and passes a high-level list of applications back to the client to pull more detailed information and not a single package add operation. The process continues by configuring the client for package or connection group additions or updates, then accesses the AppV file. Next, the contents of the AppV file are expanded and placed on the local operating system in the appropriate locations. The following is a detailed workflow of the process, assuming the package is configured for Fault Streaming.

Steps

1. Manual initiation via PowerShell or Task Sequence initiation of the Publishing Refresh process.
 - a. The App-V Client makes an HTTP connection and requests a list of applications based on the target. The Publishing refresh process supports targeting machines or users.
 - b. The App-V Publishing Server uses the identity of the initiating target, user or machine, and queries the database for a list of entitled applications. The list of applications is provided as an XML response, which the client uses to send additional requests to the server for more information on a per package basis.
2. The Publishing Agent on the App-V Client performs all actions below serialized.
 - a. Evaluate any connection groups that are unpublished or disabled, since package version updates that are part of the connection group cannot be processed.
3. Configure the packages by identifying an Add or Update operations.
 - a. The App-V Client utilizes the AppX API from Windows and accesses the AppV file from the publishing server.
 - b. The package file is opened and the AppXManifest.xml and StreamMap.xml are downloaded to the Package Store.
 - c. Completely stream publishing block data defined in the StreamMap.xml.
 - i. Store the publishing block data in the Package Store\PkgGUID\VerGUID\Root
 1. Icons: Targets of extension points.
 2. Portable Executable Headers (PE Headers): Targets of extension points that contain the base information about the image need on disk, directly accessed or via file types.
 3. Scripts: Download scripts directory for use throughout the publishing process.
 - d. Populate the Package store
 - i. Create sparse files on disk that represent the extracted package for any directories listed.
 - ii. Stage top level files and directories under root.
 - iii. All other files are created when the directory is listed as sparse on disk and streamed on demand.
 - e. Create the machine catalog entries:
 - i. Create the Manifest.xml and DeploymentConfiguration.xml from the package files (if no DeploymentConfiguration.xml file in the package a placeholder is created)

- f. Create location of the package store in the registry
HKLM\Software\Microsoft\AppV\Client\Packages\PkgGUID\Versions\VerGUID\Catalog
 - g. Create the Registry.dat file from the package store to
%ProgramData%\Microsoft\AppV\Client\VReg\{VersionGUID}.dat
 - h. Register the package with the App-V Kernel Mode Driver
HKLM\Microsoft\Software\AppV\MAV
 - i. Invoke scripting from the AppxManifest.xml or DeploymentConfig.xml file for **Package Add** timing.
4. Configure Connection Groups by adding and enabling or disabling.
 5. Remove objects that are not published to the target (user or machine).
Note: This will not perform a package deletion but rather remove integration points for the specific target (user or machine) and remove user catalog files (machine catalog files for globally published).
 6. Invoke background load mounting based on client configuration.
 7. Packages that already have publishing information for the machine or user are immediately restored.
Note: This condition occurs as a product of removal without unpublishing with background addition of the package.

This completes an App-V package add of the publishing refresh process. The next step is publishing the package to the specific target (machine or user).



* Only present when package is in a Connection Groups

Figure 1: Package Add File and Registry Data

Publishing an App-V Package

During the Publishing Refresh operation, the specific publishing operation (Publish-AppVClientPackage) adds entries to the user catalog, maps entitlement to the user, identifies the local store, and finishes by completing any integration steps. The following are the detailed steps.

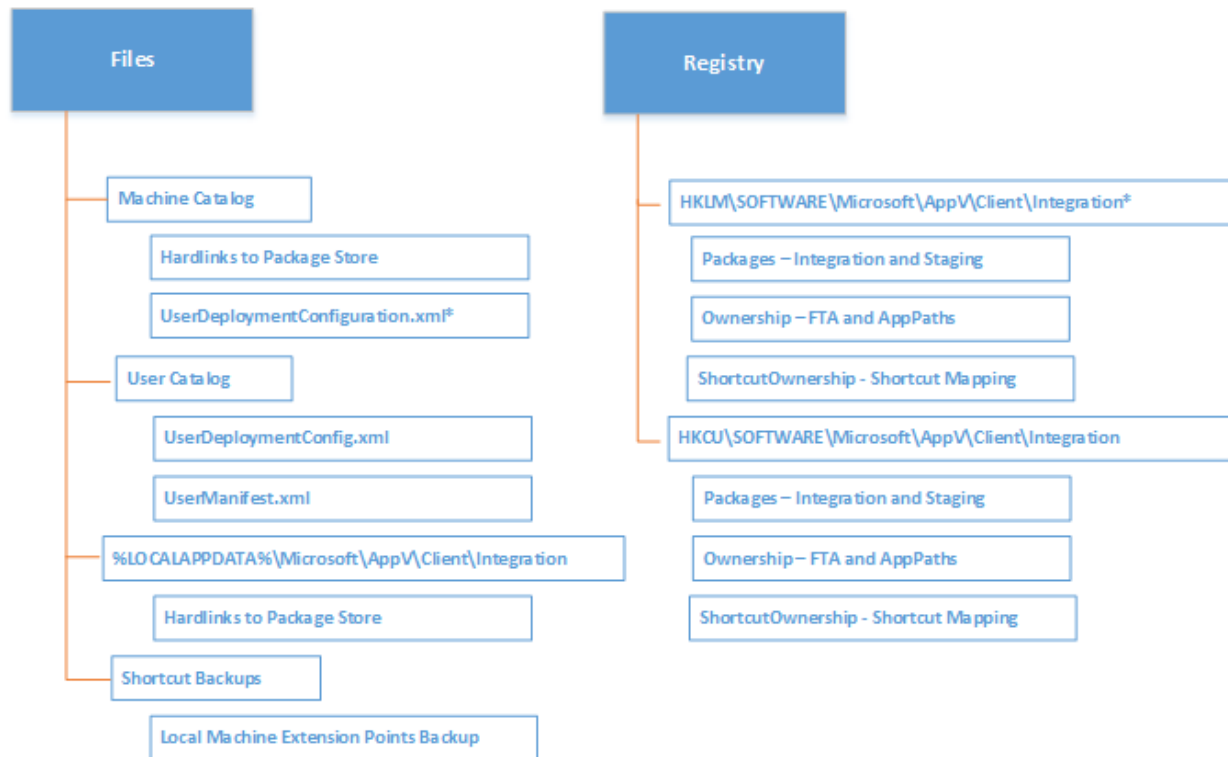
Steps

1. Package entries are added to the user catalog
 - a. User targeted packages: the UserDeploymentConfiguration.xml and UserManifest.xml are placed on the machine in the User Catalog
 - b. Machine targeted (global) packages: the UserDeploymentConfiguration.xml is placed in the Machine Catalog
2. Register the package with the kernel mode driver for the user at HKLM\Software\Microsoft\AppV\MAV
3. Perform integration tasks.
 - a. Create extension points.
 - b. Store backup information in the user's registry and roaming profile (Shortcut Backups).

Note: This enables restore extension points if the package is unpublished.

- c. Run scripts targeted for publishing timing.

Publishing an App-V Package that is part of a Connection Group is very similar to the above process. For connection groups, the path that stores the specific catalog information includes PackageGroups as a child of the Catalog Directory. Review the machine and users catalog information above for details.



* Only present if package published Globally

Figure 2: Package Publishing File and Registry Data

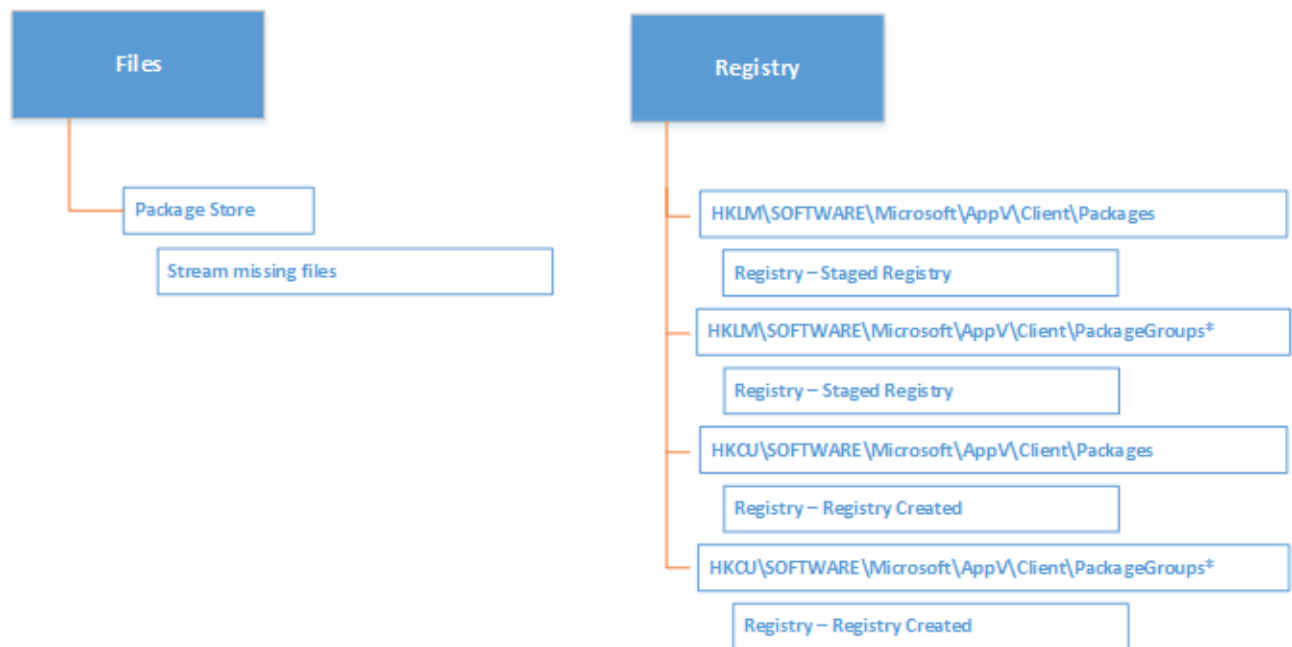
Application Launch

After the Publishing Refresh process, the user launches and subsequently re-launches an App-V application. The process is very simple and optimized to launch quickly with a minimum of network traffic. The App-V Client checks the path to the user catalog for files created during publishing. After rights to launch the package are established, the App-V Client creates a virtual environment, begins streaming any necessary data, and applies the appropriate manifest and deployment configuration files during virtual environment creation. With the virtual environment created and configured for the specific package and application, the application starts.

Steps

1. User launches the application by clicking on a shortcut or file type invocation.
2. The App-V Client verifies existence in the User Catalog for the following files
 - a. UserDeploymentConfiguration.xml
 - b. UserManifest.xml
3. If the files are present, the application is entitled for that specific user and the application will start the process for launch. There is no network traffic at this point.
4. Next, the App-V Client checks that the path for the package registered for the App-V Client service is found in the registry.
5. Upon finding the path to the package store, the virtual environment is created.
 - a. If this is the first launch, the Primary Feature Block downloads, if present.
6. After downloading, the App-V Client service consumes the manifest and deployment configuration files to configure the virtual environment and all App-V subsystems are loaded.
7. The Application launches.
8. For any missing files in the package store (sparse files), App-V will stream fault the files on an as needed basis.

Subsequent launches of the package are the same as this initial launch process, only stream faulting new data.



* Only present when package is in a Connection Groups

Figure 3: Application Launch File and Registry Data

Upgrading an App-V Package

The App-V 5 package upgrade process differs from the older versions of App-V. App-V supports multiple versions of the same package on a machine entitled to different users. Package versions can be added at any time as the package store and catalogs are updated with the new resources. The only process specific to the addition of new version resources is storage optimization. During an upgrade, only the new files are added to the new version store location and hard links are created for unchanged files. This reduces the overall storage by only presenting the file on one disk location and then projecting it into all folders with a file location entry on the disk. The specific details of upgrading an App-V Package are as follows:

Steps

1. The App-V Client performs a Publishing Refresh and discovers a newer version of an App-V Package.
2. Package entries are added to the appropriate catalog for the new version
 - a. User targeted packages: the UserDeploymentConfiguration.xml and UserManifest.xml are placed on the machine in the user catalog at
appdata\roaming\Microsoft\AppV\Client\Catalog\Packages\PkgGUID\VerGUID
 - b. Machine targeted (global) packages: the UserDeploymentConfiguration.xml is placed in the machine catalog at
%programdata%\Microsoft\AppV\Client\Catalog\Packages\PkgGUID\VerGUID
3. Register the package with the kernel mode driver for the user at
HKLM\Software\Microsoft\AppV\MAV
4. Perform integration tasks.
 - a. Integrate extensions points (EP) from the Manifest and Dynamic Configuration files.
 - i. File based EP data is stored in the AppData folder utilizing Junction Points from the package store.
 - ii. Version 1 EPs already exist when a new version becomes available.
 - iii. The extension points are switched to the Version 2 location in machine or user catalogs for any newer or updated extension points.
 - b. Run scripts targeted for publishing timing.
 - c. Install Side by Side assemblies as required.

Upgrading an in use App-V Package

App-V 5 SP2 supports upgrading packages that are in use by a user during the upgrade process. The App-V Client supports publishing multiple versions of the same package, however if a user is currently using the package that is being upgraded, it cannot be upgraded while an application or any component of the package is in use. There are two operations that must be completed before the newer version of the package can be used by a user:

1. The virtual application must be added to the machine: This operation can occur at any time as it is machine specific and only requires performing tasks detailed in the Package Add section above.
2. The virtual application must be published: The publishing process detailed in the Package Publishing section above, requires updating extension points on the system which cannot occur while a user has launched the application.

The first scenario is an App-V package that is not in use, which requires that all components of the package including any virtual application, COM server, or Shell Extension are not in use that are part of the package. The administrator publishes a newer version of the package and the upgrade works the next time a component or application inside the package is launched, the new version is streamed and run. Nothing has changed in this scenario in App-V 5 SP2 from previous releases of App-V 5.

However, when the App-V package is in use when the newer version has been published by an administrator, the process and when the App-V package is actually upgrade is different. In this upgrade scenario, since the App-V package is in use, the upgrade operation is set to pending by the App-V Client, meaning that it is queued and carried out later when the package is not in use. For typical scenarios where an App-V package application is in use the user simply shuts down the virtual application, or for scenarios where the App-V package has published Shell Extensions (Office 2013) which are permanently loaded by Windows Explorer, the user cannot be logged in. In the later scenario the users must log off and the log in to initiate the App-V Package upgrade.

Global vs User Publishing

App-V Packages can be published in one of two ways; User which entitles an App-V package to a specific user or group of users and Global which entitles the App-V package to the entire machine for all users of the machine. Once a package upgrade has been pended and the App-V package is not in use, consider the two types of publishing:

1. **Globally published:** the application is published to a machine; all users on that machine can use it. The upgrade will happen when the App-V Client Service starts, which effectively means a machine restart.
2. **User published:** the application is published to a user. If there are multiple users on the machine, the application can be published to a subset of the users. The upgrade will happen when the user logs in or when it is published again (periodically, ConfigMgr Policy refresh and evaluation, or an App-V periodic publishing/refresh, or explicitly via PowerShell commands).

Removing an App-V Packages

Removing App-V applications in a Full Infrastructure is an unpublish operation, and does not perform a package removal. The process is the same as the publish process above, but instead of adding the removal process reverses the changes that have been made for App-V Packages.

Repairing an App-V Package

The repair operation is very simple but may affect many locations on the machine. The previously mentioned Copy on Write (COW) locations are removed, and extension points are de-integrated and then re-integrated. Please review the COW data placement locations by reviewing where they are registered in the registry. This operation is done automatically and there is no administrative control other than initiating a Repair operation from the App-V Client Console or via PowerShell (Repair-AppVClientPackage).

Integration of App-V Packages

The App-V Client and package architecture provides specific integration with the local operating system during the addition and publishing of packages. Three files define the integration or extension points for an App-V Package:

- AppXManifest.xml: Stored inside of the package with fallback copies stored in the package store and the user profile. Contains the options created during the sequencing process.
- DeploymentConfig.xml: Provides configuration information of computer and user based integration extension points.
- UserConfig.xml: A subset of the Deploymentconfig.xml that only provides user- based configurations and only targets user-based extension points.

Rules of Integration

When App-V applications are published to a computer with the App-V Client, some specific actions take place as described in the list below:

- Global Publishing: Shortcuts are stored in the All Users profile location and other extension points are stored in the registry in the HKLM hive.
- User Publishing: Shortcuts are stored in the current user account profile and other extension points are stored in the registry in the HKCU hive.
- Backup and Restore: Existing native application data and registry (such as FTA registrations) are backed up during publishing.
 - App-V packages are given ownership based on the last integrated package where the ownership is passed to the newest published App-V application.
 - Ownership transfers from one App-V package to another when the owning App-V package is unpublished. This will not initiate a restore of the data or registry.
 - Restore the backed up data when the last package is unpublished or removed on a per extension point basis.

Extension Points

The App-V publishing files (manifest and dynamic configuration) provide several extension points that enable the application to integrate with the local operating system. These extension points perform typical application installation tasks, such as placing shortcuts, creating file type associations, and registering components. As these are virtualized applications that are not installed in the same manner a traditional application, there are some differences. The following is a list of extension points covered in this section:

- Shortcuts
- File Type Associations
- Shell Extensions
- COM
- Software Clients
- Application capabilities
- URL Protocol Handler
- AppPath
- Virtual Application

Shortcut

The short cut is one of the basic elements of integration with the OS and is the interface for direct user launch of an App-V application. During the publishing and unpublishing of App-V applications.

From the package manifest and dynamic configuration XML files, the path to a specific application executable can be found in a section similar to the following:

```
<Extension Category="AppV.Shortcut">
  <Shortcut>
    <File>[{Common Desktop}]\Adobe Reader 9.Ink</File>
    <Target>[{AppVPackageRoot}]\Reader\AcroRd32.exe</Target>
    <Icon>[{Windows}]\Installer\{AC76BA86-7AD7-1033-7B44-
A94000000001}\SC_Reader.ico</Icon>
    <Arguments />
    <WorkingDirectory />
    <ShowCommand>1</ShowCommand>
    <ApplicationId>[{AppVPackageRoot}]\Reader\AcroRd32.exe</ApplicationId>
  </Shortcut>
</Extension>
```

As mentioned previously, the App-V shortcuts are placed by default in the user's profile based on the refresh operation. Global refresh places shortcuts in the All Users profile and user refresh stores them in the specific user's profile. The actual executable is stored in the Package Store. The location of the ICO file is a tokenized location in the App-V package.

File Type Associations

The App-V Client manages the local operating system File Type Associations during publishing, which enables users to use file type invocations or to open a file with a specifically registered extension (.docx) to start an App-V application. File type associations are present in the manifest and dynamic configuration files as represented in the example below:

```
<Extension Category="AppV.FileTypeAssociation">
  <FileTypeAssociation>
    <FileExtension MimeAssociation="true">
      <Name>.xdp</Name>
      <ProgId>AcroExch.XDPDoc</ProgId>
      <ContentType>application/vnd.adobe.xdp+xml</ContentType>
    </FileExtension>
    <ProgId>
      <Name>AcroExch.XDPDoc</Name>
      <Description>Adobe Acrobat XML Data Package File</Description>
      <EditFlags>65536</EditFlags>
      <DefaultIcon>[{Windows}]\Installer\{AC76BA86-7AD7-1033-7B44-
A94000000001}\XDFile_8.ico</DefaultIcon>
      <ShellCommands>
        <DefaultCommand>Read</DefaultCommand>
        <ShellCommand>
          <ApplicationId>[{AppVPackageRoot}]\Reader\AcroRd32.exe</ApplicationId>
          <Name>Open</Name>
          <CommandLine>"[{AppVPackageRoot}]\Reader\AcroRd32.exe" "%1"</CommandLine>
        </ShellCommand>
        <ShellCommand>
          <ApplicationId>[{AppVPackageRoot}]\Reader\AcroRd32.exe</ApplicationId>
          <Name>Printto</Name>
          <CommandLine>"[{AppVPackageRoot}]\Reader\AcroRd32.exe" /t "%1" "%2" "%3"
"%4"</CommandLine>
        </ShellCommand>
        <ShellCommand>
          <ApplicationId>[{AppVPackageRoot}]\Reader\AcroRd32.exe</ApplicationId>
          <Name>Read</Name>
          <FriendlyName>Open with Adobe Reader 9</FriendlyName>
          <CommandLine>"[{AppVPackageRoot}]\Reader\AcroRd32.exe" "%1"</CommandLine>
        </ShellCommand>
      </ShellCommands>
    </ProgId>
  </FileTypeAssociation>
</Extension>
```

Note: In this example above, red is the extension, green is the ProgId value (which points to the adjoining ProgId), and blue is the command line, which points to the application executable.

Shell Extensions

Shell extensions will be picked up and embedded in the package automatically while sequencing. This is similar to how extension points are handled with App-V 5.0, and requires no additional work using the Sequencer. When the package is published to the client, the user will be able to utilize the shell extension functionality as they would if the application was locally installed. The application requires no additional set up or configuration on the client to enable the shell extension functionality. The following list displays the supported Shell Extensions.

Supported Shell Extensions:

Handler	Description
Context menu handler	Adds menu items to the context menu. It is called before the context menu is displayed.
Drag-and-drop handler	Controls the action upon right-click drag-and-drop and modifies the context menu that appears.
Drop target handler	Controls the action after a data object is dragged-and-dropped over a drop target such as a file.
Data object handler	Controls the action after a file is copied to the clipboard or dragged-and-dropped over a drop target. It can provide additional clipboard formats to the drop target.
Property sheet handler	Replaces or adds pages to the property sheet dialog box of an object.
Infotip handler	Allows retrieving flags and infotip information for an item and displaying it inside a popup tooltip upon mouse-hover.
Column handler	Allows creating and displaying custom columns in Windows Explorer <i>Details view</i> . It can be used to extend sorting and grouping.

COM

The App-V Client supports publishing applications with support for COM integration and virtualization. COM integration allows the App-V Client to register COM objects on the local operating system and virtualization of the objects. For the purposes of this document, the integration of COM objects requires additional detail.

App-V supports registering COM objects from the package to the local operating system with two process types: Out-of-process and in-process. Registering COM objects is accomplished with one or a combination of multiple modes of operation for a specific App-V package that includes off, Isolated, and Integrated. The integrated mode is configured for either the out-of-process or in-process type. Configuration of COM modes and types is accomplished with dynamic configuration files (deploymentconfig.xml or userconfig.xml).

Details on App-V integration are available at:

<http://blogs.technet.com/b/appv/archive/2013/01/03/microsoft-application-virtualization-5-0-integration.aspx>

Software Clients and Application Capabilities

App-V supports specific software clients and application capabilities extension points that enable virtualized applications to be registered with the software client of the operating system. This enables users to select default programs for operations like email, instant messaging, and media player. This operation is performed in the control panel with the **Set Program Access and Computer Defaults**, and configured during sequencing in the manifest or dynamic configuration files. Application capabilities are only supported when the App-V applications are published globally.

Example of software client registration of an App-V based mail client.

```
<SoftwareClients Enabled="true">
  <ClientConfiguration EmailEnabled="true" />
  <Extensions>
    <Extension Category="AppV.SoftwareClient">
      <SoftwareClients>
        <EMail MakeDefault="true">
          <Name>Mozilla Thunderbird</Name>
          <Description>Mozilla Thunderbird</Description>
          <DefaultIcon>[{ProgramFilesX86}]\Mozilla Thunderbird\thunderbird.exe,0</DefaultIcon>
          <InstallationInformation>
            <RegistrationCommands>
              <Reinstall>"[{ProgramFilesX86}]\Mozilla Thunderbird\uninstall\helper.exe"
/SetAsDefaultAppGlobal</Reinstall>
              <HideIcons>"[{ProgramFilesX86}]\Mozilla Thunderbird\uninstall\helper.exe"
/HideShortcuts</HideIcons>
              <ShowIcons>"[{ProgramFilesX86}]\Mozilla Thunderbird\uninstall\helper.exe"
/ShowShortcuts</ShowIcons>
            </RegistrationCommands>
            <IconsVisible>1</IconsVisible>
            <OEMSettings />
          </InstallationInformation>
        </EMail>
      </SoftwareClients>
    </Extension>
  </Extensions>
</SoftwareClients>
```

```

    <ShellCommands>
      <ApplicationId>[{ProgramFilesX86}]\Mozilla Thunderbird\thunderbird.exe</ApplicationId>
      <Open>"[{ProgramFilesX86}]\Mozilla Thunderbird\thunderbird.exe" -mail</Open>
    </ShellCommands>
    <MAPILibrary>[{ProgramFilesX86}]\Mozilla
Thunderbird\mozMapi32_InUse.dll</MAPILibrary>
    <MailToProtocol>
      <Description>Thunderbird URL</Description>
      <EditFlags>2</EditFlags>
      <DefaultIcon>[{ProgramFilesX86}]\Mozilla Thunderbird\thunderbird.exe,0</DefaultIcon>
      <ShellCommands>
        <ApplicationId>[{ProgramFilesX86}]\Mozilla Thunderbird\thunderbird.exe</ApplicationId>
        <Open>"[{ProgramFilesX86}]\Mozilla Thunderbird\thunderbird.exe" -osint -compose
"%1"</Open>
      </ShellCommands>
    </MailToProtocol>
  </EMail>
</SoftwareClients>
</Extension>
</Extensions>
</SoftwareClients>

```

NOTE: In this example, red is the overall Software Clients setting to integrate Email clients, green is the flag to set a particular Email client as the default Email client, and blue is the MAPI dll registration.

URL Protocol handler

Applications do not always specifically called virtualized applications utilizing file type invocation. For, example, in an application that supports embedding a mailto: link inside a document or web page, the user clicks on a mailto: link and expects to get their registered mail client. App-V supports URL Protocol handlers that can be registered on a per-package basis with the local operating system. During sequencing, the URL protocol handlers are automatically added to the package.

For situations where there is more than one application that could register the specific URL Protocol handler, the dynamic configuration files can be utilized to modify the behavior and suppress or disable this feature for an application that should not be the primary application launched.

AppPath

The AppPath extension point supports calling App-V applications directly from the operating system. This is typically accomplished from the Run or Start Screen, depending on the operating system, which enables administrators to provide access to App-V applications from operating system commands or scripts without calling the specific path to the executable. It therefore avoids modifying the system path environment variable on all systems, as it is accomplished during publishing.

The AppPath extension point is configured either in the manifest or in the dynamic configuration files and is stored in the registry on the local machine during publishing for the user. For additional information on AppPath review: <http://blogs.technet.com/b/virtualvibes/archive/2012/11/30/app-paths-a-virtual-application-extension-in-app-v-5-0.aspx>

Virtual Application

This subsystem provides a list of applications captured during sequencing which is usually consumed by other App-V components. Integration of extension points belonging to a particular application can be disabled using dynamic configuration files. For example, if a package contains two applications, it is possible to disable all extension points belonging to one application, in order to allow only integration of extension points of other application.

Extension Point Rules

The extension points described above are integrated into the operating system based on how the packages has been published. Global publishing places extension points in public machine locations, where user publishing places extension points in user locations. For example a shortcut that is created on the desktop and published globally will result in the file data for the shortcut (%Public%\Desktop) and the registry data (HKLM\Software\Classes). The same shortcut would have file data (%UserProfile%\Desktop) and registry data (HKCU\Software\Classes).

Extension points are not all published the same way, where some extension points will require global publishing and others require sequencing on the specific operating system and architecture where they are delivered. Below is a table that describes these two key rules.

Table 2: Extension Point Requirements

Virtual Extension	Requires target OS Sequencing	Requires Global Publishing
Shortcut		
File Type Association		
URL Protocols	X	
AppPaths	X	
COM Mode		
Software Client	X	
Application Capabilities	X	X
Context Menu Handler	X	X
Drag-and-drop Handler	X	
Data Object Handler	X	
Property Sheet Handler	X	
Infotip Handler	X	
Column Handler	X	
Shell Extensions	X	
Browser Helper Object	X	X
Active X Object	X	X

Dynamic Configuration Processing

Deploying App-V packages to one machine or user is very simple. However, as organizations deploy App-V applications across business lines and geographic and political boundaries, the ability to sequence an application one time with one set of settings becomes impossible. App-V was designed for this scenario, as it captures specific settings and configurations during sequencing in the Manifest file, but also supports modification with Dynamic Configuration files.

App-V dynamic configuration allows for specifying a policy for a package either at the machine level or at the user level. The Dynamic Configuration files enable sequencing engineers to modify the configuration of a package, post-sequencing, to address the needs of individual groups of users or machines. In some instances it may be necessary to make modifications to the application to provide proper functionality within the App-V environment. For example, it may be necessary to make modifications to the `_*.config.xml` files to allow certain actions to be performed at a specified time during the execution of the application, like disabling a mailto extension to prevent a virtualized application from overwriting that extension from another application.

App-V Packages contain the Manifest file inside of the APPV package file, which is representative of sequencing operations and is the policy of choice unless Dynamic Configuration files are assigned to a specific package. Post-sequencing, the Dynamic Configuration files can be modified to allow the publishing of an application to different desktops or users with different extension points. The two Dynamic Configuration Files are the Dynamic Deployment Configuration (DDC) and Dynamic User Configuration (DUC) files. This section focuses on the combination of the manifest and dynamic configuration files.

Example for Dynamic Configuration Files

The example below shows the combination of the Manifest, Deployment Configuration and User Configuration files after publishing and during normal operation. These examples are abbreviated examples of each of the files. The purpose is show the combination of the files only and not to be a complete description of the specific categories available in each of the files. For more information

review the App-V 5 Sequencing Guide at: <http://www.microsoft.com/en-us/download/details.aspx?id=27760>

Manifest

```
<appv:Extension Category="AppV.Shortcut">
  <appv:Shortcut>
    <appv:File>[{Common Programs}]\7-Zip\7-Zip File Manager.lnk</appv:File>
    <appv:Target>[{AppVPackageRoot}]\7zFM.exe</appv:Target>
    <appv:Icon>[{AppVPackageRoot}]\7zFM.exe.0.ico</appv:Icon>
  </appv:Shortcut>
</appv:Extension>
```

Deployment Configuration

```
<MachineConfiguration>
  <Subsystems>
    <Registry>
      <Include>
        <Key Path="\REGISTRY\Machine\Software\7zip">
          <Value Type="REG_SZ" Name="Config" Data="1234"/>
        </Key>
      </Include>
    </Registry>
  </Subsystems>
```

User Configuration

```
<UserConfiguration>
  <Subsystems>
    <appv:Extension Category="AppV.Shortcut">
      <appv:Shortcut>
        <appv:File>[{Desktop}]\7-Zip\7-Zip File Manager.lnk</appv:File>
        <appv:Target>[{AppVPackageRoot}]\7zFM.exe</appv:Target>
        <appv:Icon>[{AppVPackageRoot}]\7zFM.exe.0.ico</appv:Icon>
      </appv:Shortcut>
    </appv:Extension>
  </Subsystems>
```

Figure 4: Example configuration files


```

<UserConfiguration>
  <Subsystems>
<appv:Extension Category="AppV.Shortcut">
  <appv:Shortcut>
    <appv:File>[{Desktop}]\7-Zip\7-Zip File Manager.lnk</appv:File>
    <appv:Target>[{AppVPackageRoot}]\7zFM.exe</appv:Target>
    <appv:Icon>[{AppVPackageRoot}]\7zFM.exe.0.ico</appv:Icon>
  </appv:Shortcut>
  <appv:Shortcut>
    <appv:File>[{Common Programs}]\7-Zip\7-Zip File Manager.lnk</appv:File>
    <appv:Target>[{AppVPackageRoot}]\7zFM.exe</appv:Target>
    <appv:Icon>[{AppVPackageRoot}]\7zFM.exe.0.ico</appv:Icon>
  </appv:Shortcut>
</appv:Extension>
  </Subsystems>
<MachineConfiguration>
  <Subsystems>
    <Registry>
      <Include>
        <Key Path="\REGISTRY\Machine\Software\7zip">
          <Value Type="REG_SZ" Name="Config" Data="1234"/>
        </Key>
      </Include>
    </Registry>
  </Subsystems>

```

Figure 5: Post publishing combination

Side by Side Assemblies

App-V supports the automatic packaging of side-by-side (SxS) assemblies during sequencing and deployment on the client during virtual application publishing. App-V 5 SP2 supports capturing SxS assemblies during sequencing for assemblies not present on the sequencing machine. And for assemblies consisting of Visual C++ (Version 8 and newer) and/or MSXML run-time, the sequencer will automatically detect and capture these dependencies even if they were not installed during monitoring. The Side by Side assemblies feature removes the limitations of previous versions of App-V, where the App-V Sequencer did not capture assemblies already present on the sequencing workstation, and privatizing the assemblies which limited to one bit version per package. This behavior resulted in deployed App-V applications to clients missing the required SxS assemblies, causing application launch failures. This forced the packaging process to document and then ensure that all assemblies required for packages were locally installed on the user's client operating system to ensure support for the virtual applications. Based on the number of assemblies and the lack of application documentation for the required dependencies, this task was both a management and implementation challenge.

Side by Side Assembly support in App-V has the following features.

- Automatic captures of SxS assembly during Sequencing, regardless of whether the assembly was already installed on the sequencing workstation.
- The App-V Client automatically installs required SxS assemblies to the client computer at publishing time when they are not present.
- The Sequencer reports the VC run-time dependency in Sequencer reporting mechanism.
- The Sequencer allows opting to not package the assemblies that are already installed on the sequencer, supporting scenarios where the assemblies have previously been installed on the target computers.

Automatic publishing of SxS assemblies

During publishing of an App-V package with SxS assemblies the App-V Client will check for the presence of the assembly on the machine. If the assembly does not exist, the client will deploy the assembly to the machine. Packages that are part of connection groups will rely on the Side by Side assembly installations that are part of the base packages, as the connection group does not contain any information about assembly installation.

Note: UnPublishing or removing a package with an assembly does not remove the assemblies for that package.

Client Logging

The App-V client logs information to the Windows Event log in standard ETW format. The specific App-V events can be found in the event viewer, under Applications and Services Logs\Microsoft\AppV\Client. There are three specific categories of events recorded described below.

Admin: Logs events for configurations being applied to the App-V Client, and contains the primary warnings and errors.

Operational: Logs the general App-V execution and usage of individual components creating an audit log of the App-V operations that have been completed on the App-V Client.

Virtual Application: Logs virtual application launches and use of virtualization subsystems.

Conclusion

The document has presented detailed information about how App-V performs Integration of virtual applications that present themselves like traditionally installed applications. Based on the information presented in the document, administrators and sequencing engineers have a better understanding of the assets that are part of an AppV Package and where the App-Client places those assets. The details provided enable better support and understanding of an App-V solution. Keep in mind that many of the details presented are specific integration rules that can't be modified, but understanding them assists in an overall knowledge of the App-V product and how it works.