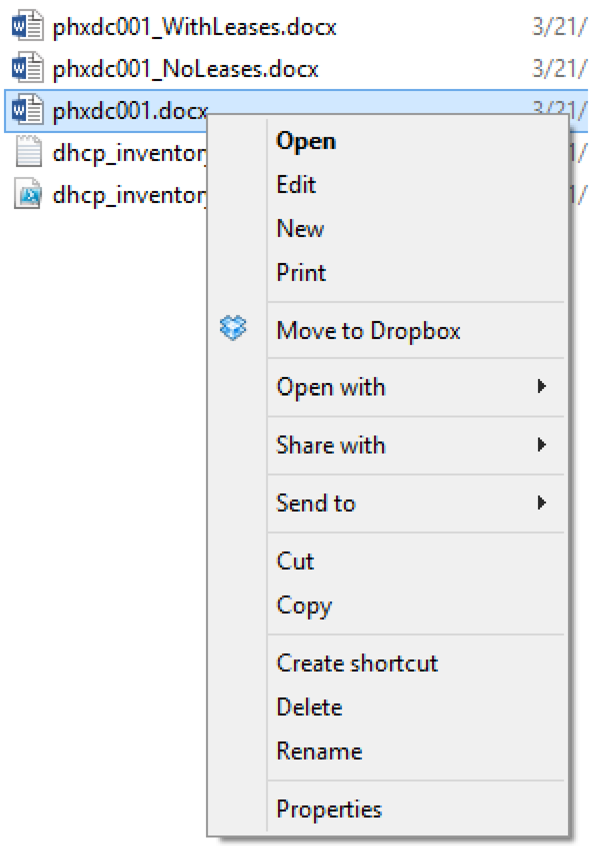
Effects of "File-Type-Associations" in App-V 5 SP2 Deployment Performance

TMurgent Performance Research Series



June, 2014

Contents

1 Introduction

The purpose of this research paper is to document the effects that file type association (FTA) shell integrations have in Microsoft App-V Virtual Application Packages.

Specifically not included in this paper are the following, closely related App-V extensions:

- Shell Extensions, while part of file type associations, are not included in any testing for this paper. Shell extensions would require a different kind of testing.
- Protocol Handlers. Are also not tested as part of this effort. It is assumed that Protocol Handlers would have a similar effect as an FTA to package performance.
- Application Capabilities Registration. It is assumed that adding capabilities registration for an FTA would slightly, but not significantly, increase the performance impact of an FTA.

The effort is squarely aimed at answering questions on how the addition of FTAs in a package affect performance. This work is part of a series of efforts to characterize the impact that different application elements have on the performance of virtual applications.

Most readers of this research will find themselves satisfied with reading the second and third section of this paper. The remaining sections detail the testing process, packages used, and provide further test details and additional findings.

2 Background on File Type Associations

A File Type Association is a registration for files ending with a specific file extension, to provide enhanced integrations supported by the windows shell (explorer.exe by default).

The FTA allows explorer to use a specific icon to display the file, to customize/extend the right-click menu options, and associate programs to be run when the user interacts with the file.

The FTA consists mostly of registry entries added to the Classes key. The application can add the information to either the HKLM\Software\Classes key or the HKCU\Software\Classes key to cause availability to either all users or only the current user. In either case, App-V will detect the FTA and generalize the information to deploy to all users or an individual user based on the deployment assignment.

In most cases, An FTA registry entry under a Classes key consists of a sub-key named with the file extension (including the leading “dot”) with additional data underneath. The FTA detail can (and usually does) include a reference to a ProgID. The ProgID will then contain the detail needed to associate the file type with an executable. The ProgID also is added under the classes key. In some commercial applications, there may be:

- No ProgIDs used
- A single FTA with a single ProgID
- A single FTA with multiple ProgIDs
- Multiple FTAs sharing a single ProgID
- Combinations of the above

Due to the extremely low impact that added registry items in general have on App-V performance, it is reasonable to assume that the combinations and complexity of an FTA are not that significant to performance, however that these extensions need processing are.

The application installer can also add shell integrations (right click menu items that point to an exe) or shell extensions (items that point to a dll) to the file type. This document will not deal with either shell integrations or extensions.

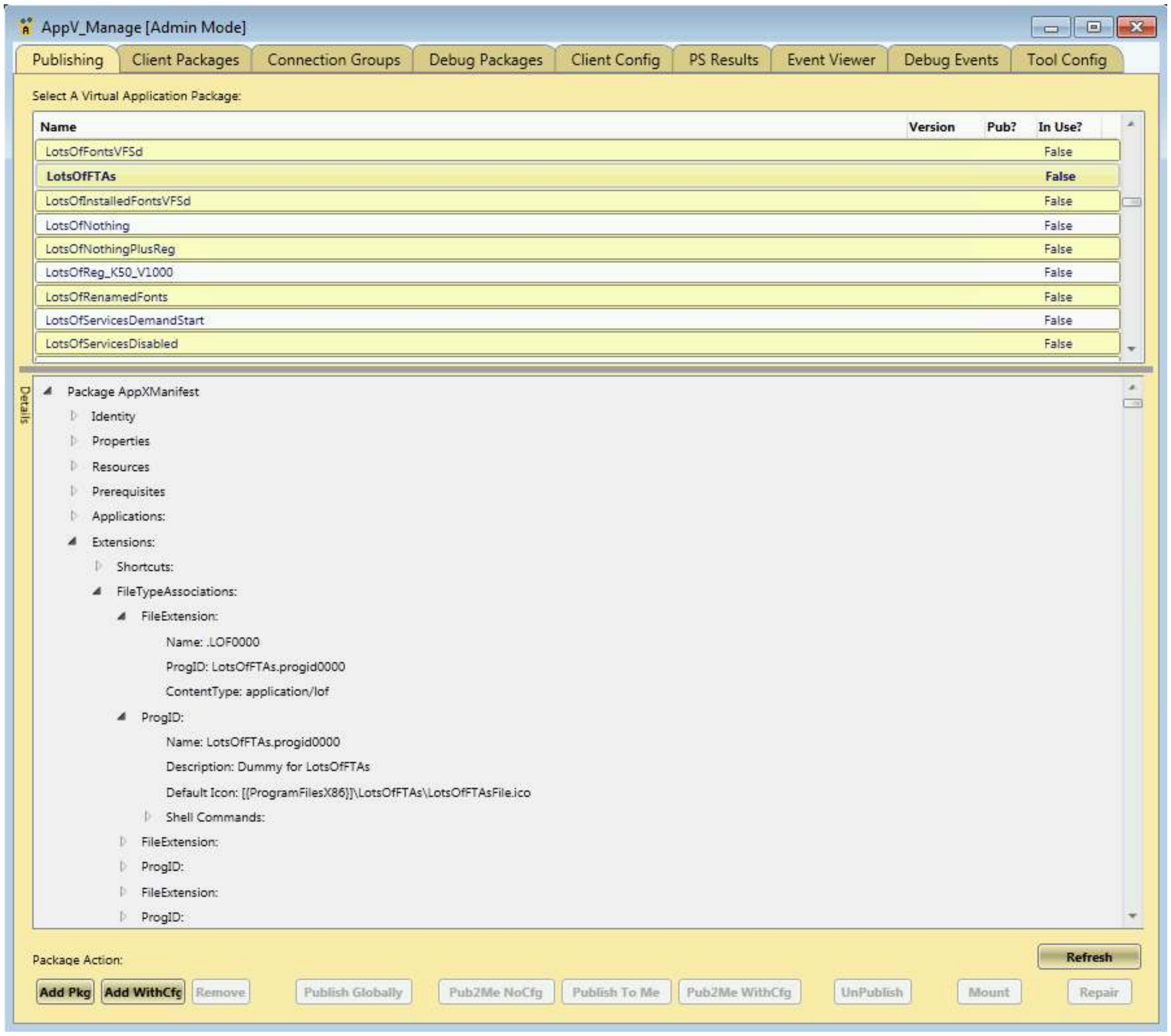
2.1 Detecting FTAs in a Package

How do you detect if a package has any FTAs?

Often, we use the “Shortcuts and File Associations” tab of the App-V Sequencer Editor pages when initially sequencing the package. **Unfortunately, this is an unreliable method to detect FTAs**! Some types of FTAs, notably when the installer does not take ownership of the FTA but adds itself to the capabilities registration to be allowed to take ownership and show up on the "open with" right-click.

Instead, inside the sequencer you may browse to the MACHINE or USER hive and look for the registry entries directly. Additionally, you can look for the associations in the DeploymentConfig or internal AppXManifest file for the App-V package.

The image below are taken from the AppV_Manage tool where you can view the AppXManifest file.

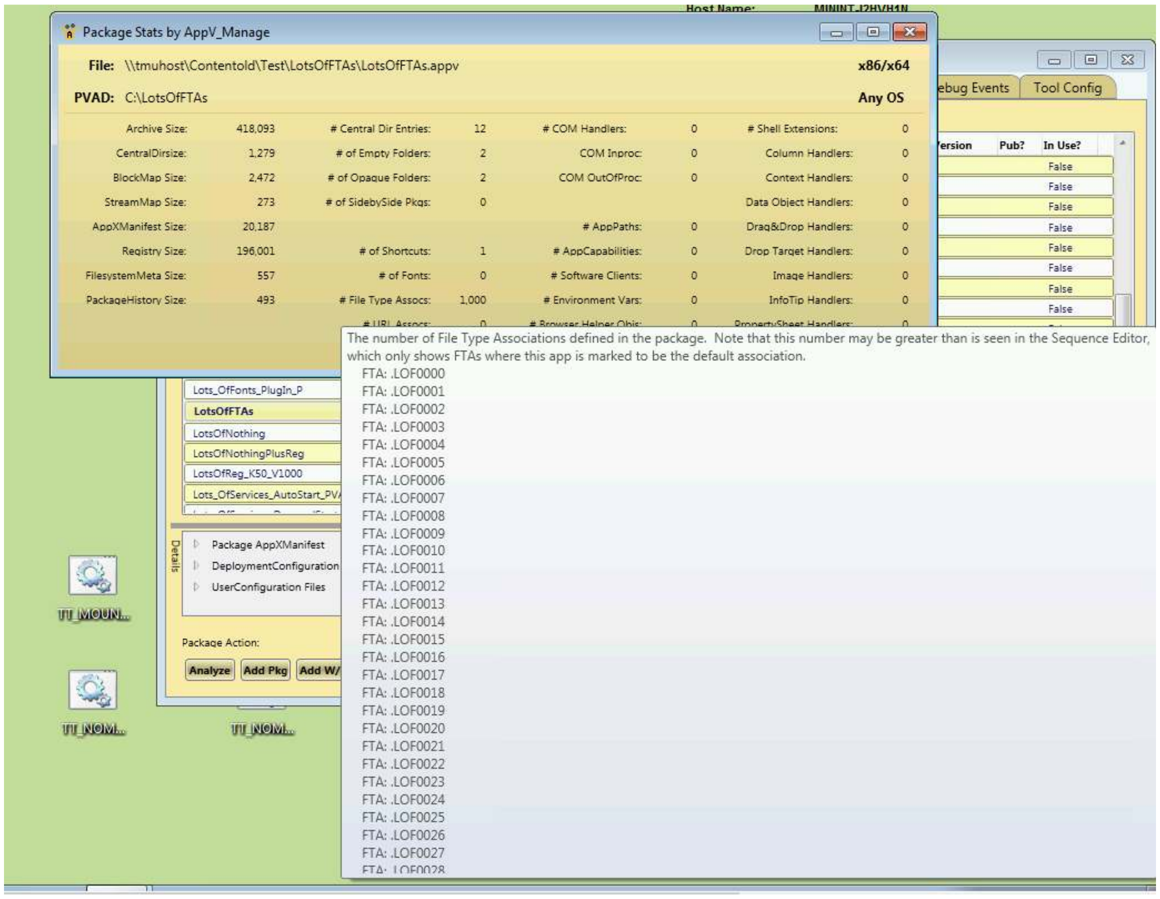


Select A Virtual Application Package:

Name	Version	Pub?	In Use?
LotsOfFontsVFSd			False
LotsOfFTAs			**False**
LotsOfInstalledFontsVFSd			False
LotsOfNothing			False
LotsOfNothingPlusReg			False
LotsOfReg_K50_V1000			False
LotsOfRenamedFonts			False
LotsOfServicesDemandStart			False
LotsOfServicesDisabled			False

```
Package AppXManifest
    Identity
    Properties
    Resources
    Prerequisites
    Applications:
    Extensions:
        Shortcuts:
        FileTypeAssociations:
            FileExtension:
                Name: .LOF0000
                ProgID: LotsOfFTAs.progid0000
                ContentType: application/lof
            ProgID:
                Name: LotsOfFTAs.progid0000
                Description: Dummy for LotsOfFTAs
                Default Icon: [{ProgramFilesX86}]\LotsOfFTAs\LotsOfFTAsFile.ico
                Shell Commands:
            FileExtension:
            ProgID:
            FileExtension:
            ProgID:
```

Additionally, you can use the package analyzer in AppV_Manage to see the FTAs:



File: \\tmuhost\Contentold\Test\LotsOfFTAs\LotsOfFTAs.appv **x86/x64**

PVAD: C:\LotsOfFTAs **Any OS**

Archive Size:	418,093	# Central Dir Entries:	12	# COM Handlers:	0	# Shell Extensions:	0
CentralDirsize:	1,279	# of Empty Folders:	2	COM Inproc:	0	Column Handlers:	0
BlockMap Size:	2,472	# of Opaque Folders:	2	COM OutOfProc:	0	Context Handlers:	0
StreamMap Size:	273	# of SidebySide Pkgs:	0			Data Object Handlers:	0
AppXManifest Size:	20,187			# AppPaths:	0	Drag&Drop Handlers:	0
Registry Size:	196,001	# of Shortcuts:	1	# AppCapabilities:	0	Drop Target Handlers:	0
FilesystemMeta Size:	557	# of Fonts:	0	# Software Clients:	0	Image Handlers:	0
PackageHistory Size:	493	# File Type Assocs:	1,000	# Environment Vars:	0	InfoTip Handlers:	0
		# URL Assocs:	0	# Browser Helper Objs:	0	PropertySheet Handlers:	0

The number of File Type Associations defined in the package. Note that this number may be greater than is seen in the Sequence Editor, which only shows FTAs where this app is marked to be the default association.

FTA: .LOF0000
FTA: .LOF0001
FTA: .LOF0002
FTA: .LOF0003
FTA: .LOF0004
FTA: .LOF0005
FTA: .LOF0006
FTA: .LOF0007
FTA: .LOF0008
FTA: .LOF0009
FTA: .LOF0010
FTA: .LOF0011
FTA: .LOF0012
FTA: .LOF0013
FTA: .LOF0014
FTA: .LOF0015
FTA: .LOF0016
FTA: .LOF0017
FTA: .LOF0018
FTA: .LOF0019
FTA: .LOF0020
FTA: .LOF0021
FTA: .LOF0022
FTA: .LOF0023
FTA: .LOF0024
FTA: .LOF0025
FTA: .LOF0026
FTA: .LOF0027
FTA: .LOF0028

3 Summary of Where Impacts of FTAs Are Felt

FTAs affect performance in three ways:

- As a set of registry entries, an FTAs increase the size of the virtual registry, however the impact of this on performance is too small to be detectable.
- As an extension, FTAs add time to Publish-AppVClientPackage.
- When an FTA points to an icon, the icon is added to the publishing block of the package. When the FTA points to an exe containing an icon, the Sequencer extracts the icon out into an ico file, which is then added to the publishing block. Items in the publishing block are streamed to the local cache (no matter what the SCS mode is set to) during the Publish-AppVClientPackage step. Even a complicated package with a lot of FTAs likely only uses a handful of icons, so this impact is minor.

4 Testing Strategy Used

This section provides details about how the testing was performed.

4.1 About the Testing Platform

The testing results depicted in this paper are based on:

App-V 5.0 SP2 with HotFix 4 running on a Windows 7 SP1 x86 virtual machine.

The testing was performed in an isolated environment using a Microsoft 2012 R2 server with Hyper-V. The server has 24 processors and 64GB or RAM. To minimize external impacts, this server utilizes local storage and contains a VM with the domain controller. App-V Package sources were located on a share on this host.

The Test VM used had 2GB of RAM and was given 2 virtual CPUs. The App-V Client is configured for Shared Content Store mode (which disables background streaming).

4.2 About Test Packages and "Streaming Configuration"

All Test packages used are specially constructed software packages that I developed. These packages are generally stripped down to a bare minimum, except for an overabundance of the one particular things we want to measure when using this package. In many cases, this means custom software that I developed for the purpose of the test.

Unless specifically noted, each package was sequenced and configured for streaming by *not* launching anything during the streaming training configuration phase of the sequencer. This means that, barring mounting operations, almost everything in the package will fault-stream (stream on demand).

4.3 About the Testing Methods

All tests are automated using significant sleep periods before each portion of the testing to allow all systems to settle down, and warm-up of the external components (hypervisor/fileshare) and within the OS (App-V Client and drivers) are performed. The test process consists of

- A ***Test Cycle*** that consists of a series of Test Passes.
- Each ***Test Pass*** consists of a number of Test Packages.
- Each ***Tested Package*** is tested using a series of actions and measurements.

A *Tested Package*, consists of a series of actions, always preceded by a significant sleep period to allow system background processes to settle down.

A *Test Pass* always starts from a freshly booted snapshot and with a dummy *Test Package* to warm up the App-V Client and Driver sub-systems. The results of this dummy package are not used.

A *Test Cycle* always starts with a *Test Pass* to warm up the external components of the Hypervisor and Windows File Share. Because the packages are relatively small compared to the amount of memory available, the packages are likely retained in memory in the Windows Standby Lists after the initial *Test Cycle.*

These are described as follows, from the bottom up.

4.3.1 Test Package

For a given ***Test Package***, the series of actions includes:

- Waiting
- Add-AppVClientPackage
- Waiting
- Publish-AppVClientPackage
- Waiting
- [Optionally Mount-AppVClientPackage[1]]
- Waiting
- First run (launch "cmd.exe[2] /c time /t" inside the virtual environment).
- Waiting
- Second run (launch "cmd.exe[3] /c time /t" inside the virtual environment).

The time required for each of the actions to complete is recorded.

4.3.2 Test Pass

A ***Test Pass*** consists of testing multiple *Test Packages* as follows:

- Reverting the test VM to a snapshot.
- Waiting for the Hypervisor to settle.
- Booting the VM and logging in.
- Waiting.

[1] With SCS enabled, mounting the package does result in the actual file content being stored in the App-V file cache. I test in SCS mode both with and without mounting to better delineate the cause of performance slowdowns on a package.

[2] This is used rather than a program in the package to produce a comparable time that varies based on special actions that the client must perform during virtual environment startup and shutdown due to the package content.

[3] The client is also known to perform special actions the first time a virtual environment is used, so the second run is used for comparison to the first run.

- A series of actions and measurements on a warm-up package. These results are never used, it is only performed to warm up the client (client service, drivers, and WMI) and to ensure that each subsequent package fairly tested under similar conditions.
- Waiting.
- A series of actions and measurements on the first package.
- Waiting.
- A series of actions and measurements on the second package.
- Etc…
- Recording results

4.3.3 Test Cycle

Finally, A ***Test Cycle*** consists of several consecutive test runs of the same *Test Pass*. The first pass is used to “warm up” external systems and achieve a relatively consistent amount of caching by the server. The results of this pass are not used, but the results of the remaining passes are averaged to produce results. A Test Cycle typically requires a full day to complete.

4.4 About the Test Results Accuracy

As careful as I attempt to be to eliminate variability in the results, there is a fair amount of variability in results between two passes.

Due to the nature of the background interruptions affecting the results, the impact on result accuracy is felt much more on measurements that are shorter in duration than those that are longer. With measurements that are sub-second, this can produce results that typically vary by as much as +/-10% from the average.

Instead, I use an approach to test with a sufficient number of test cycles and select the minimum value seen on any of the tests. The more repetitions that are made, the better this minimum value represents the time it takes for App-V to complete the task without the effects of any extraneous background interference.

5 App-V Package Definitions

This section details the packages used in testing.

5.1.1 Warm-up Package

This package is primarily used as the first package in a Test Pass, to warm up the OS and App-V Client components and dependencies[4].

5.1.2 FullOfNothing (Baseline)

This is a minimal App-V Package.

In developing this package, I discovered that there is an issue with the App-V Client in that there appears to be some sort of undocumented minimal package requirements. If you create a package with no registry entries, no files, and no integrations, the Add-AppVClientPackage cmdlet will error out with error 700002.

Therefore this package consists of one HKLM registry key, one HKCU registry key, one text file in the PVAD folder, and one shortcut (to the text file).

The package was tested to produce a baseline for “absolute minimum” of what the App-V Client can do. These numbers are useful in determining the amount of overhead that the FTAs place on the system.

5.1.3 LotsOfFTAs

This package consists of a small dummy program with a shortcut and 1000 FTA entries using 1000 ProgIDs. As with many of my test packages, this is a home grown package designed to just include the FTAs. Each FTA/ProgID pair points to the same icon for display purposes.

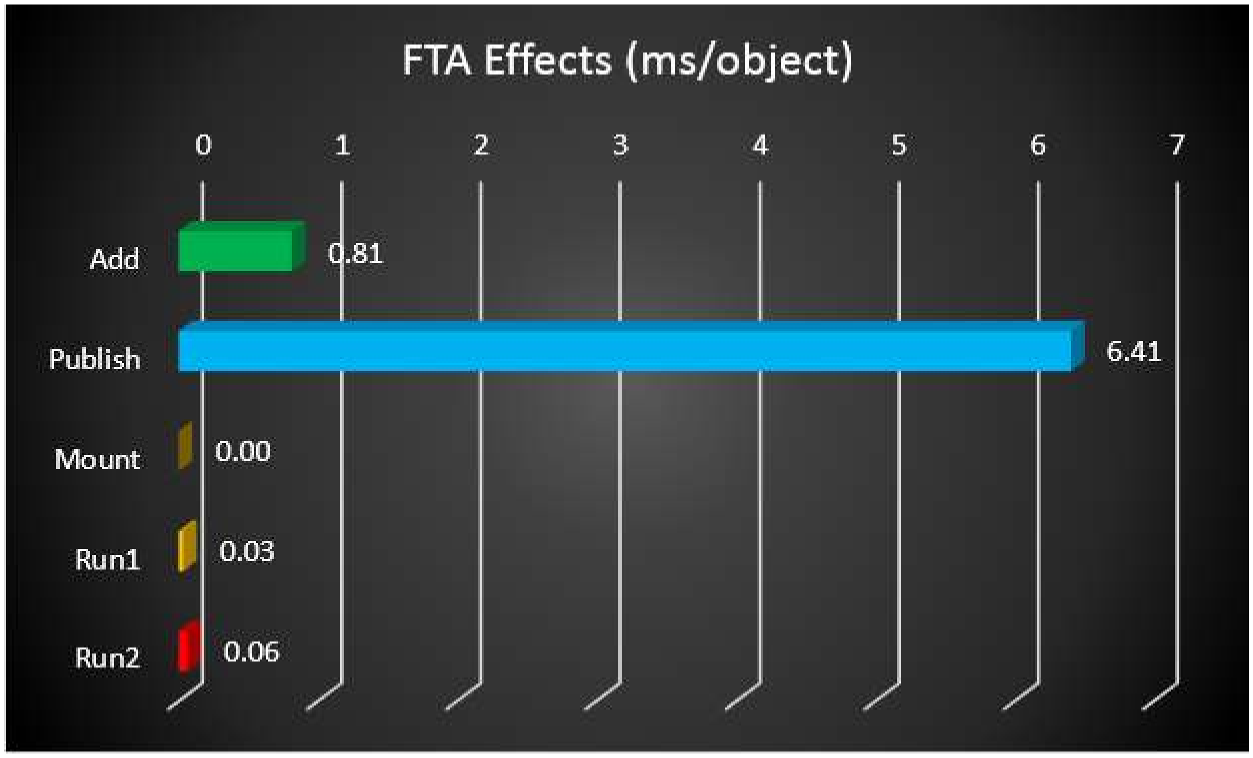
Each FTA points to a specific ProgID. The ProgID consists of a pointer to the icon, plus an “Open” command pointing to the dummy program.

[4] When conducting tests that use mounting, I found it necessary to warm up the system without mounting this package. It appears that the first client activity after boot requires additional time to warm up the client, possibly loading drivers. But I also found that mounting this package causes an odd additional 1 second hit to any subsequently Add-AppVClientPackage commands (even after settling time). This issue only seems to exist with this package, and mounting other packages does not affect subsequent Add cmdlets. The cause of this is unknown.

6 Detail Test Results

Results reported are based on an ideal test environment. Performance impacts identified in this paper will be very different in production environments. Specific numbers are *only* useful in comparison to numbers from other research papers in this series!

Tests were performed with and without Mounting, and with SCS enabled and disabled. As all impacts occur during the add and publish steps, only a single result (SCS with mounting) is presented here.



From this we determine that the effect of FTAs are felt primarily at the Publish step, and a bit at the Add step. The differences seen at runtime are likely within the error margin of the tests.

Overhead at the Add Step comes mostly from a larger AppXManifest file to be processed. Overhead at the Publish Step comes from implementing the FTAs natively on the system.

From the numbers we reach the following conclusions:

An FTA adds less than 1ms each to the Add Step.

An FTA adds about 6.5ms each to the Publish Step.

7 About This Research Paper Series

This research paper is part of a series of papers, released by TMurgent Technologies, that investigate the performance impacts that certain application contents can have in the deployment of Microsoft App-V 5 packages.

Through these papers, we can better understand what areas to focus on when packaging applications for App-V when deployment and end-user experience is important. Additionally, with an understanding of these papers you can better target a specific package that is performing poorly and prioritize your efforts to improve it.

TMurgent Technologies, LLP is based in Canton, MA, USA; just 17 miles south of the offices where Microsoft develops the App-V product. TMurgent's Tim Mangan has a long history with the product, having built the original version at Softricity more than a dozen years ago. TMurgent is well known in the App-V community as a source for the best training classes on App-V as well as an endless supply of tools and information. More information is available at the website, www.tmurgent.com