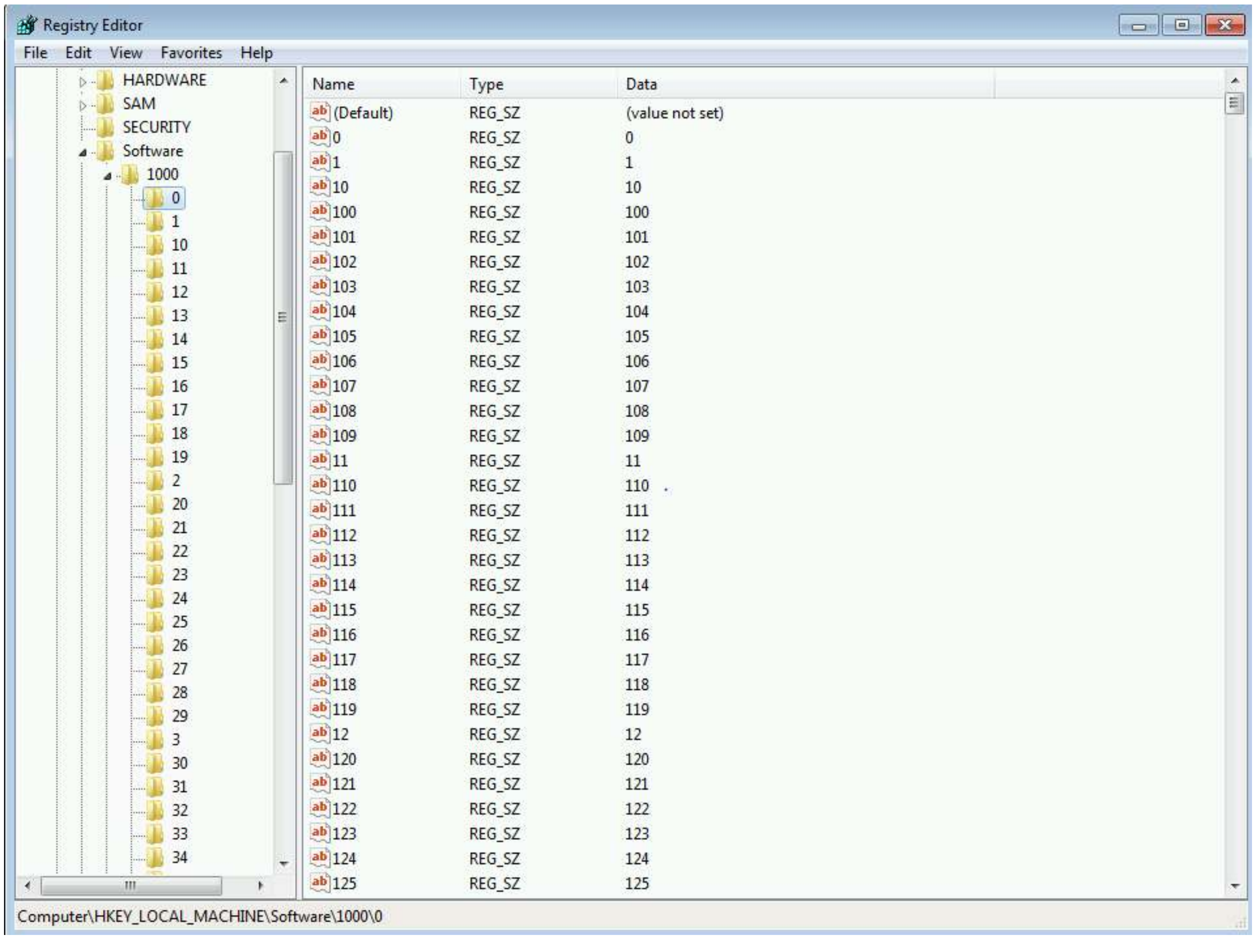
Effects of “Registry” in App-V 5 SP2 Deployment Performance

TMurgent Performance Research Series



HARDWARE
SAM
SECURITY
Software
1000
0
1
10
11
12
13
14
15
16
17
18
19
2
20
21
22
23
24
25
26
27
28
29
3
30
31
32
33
34

Name	Type	Data
(Default)	REG_SZ	(value not set)
0	REG_SZ	0
1	REG_SZ	1
10	REG_SZ	10
100	REG_SZ	100
101	REG_SZ	101
102	REG_SZ	102
103	REG_SZ	103
104	REG_SZ	104
105	REG_SZ	105
106	REG_SZ	106
107	REG_SZ	107
108	REG_SZ	108
109	REG_SZ	109
11	REG_SZ	11
110	REG_SZ	110
111	REG_SZ	111
112	REG_SZ	112
113	REG_SZ	113
114	REG_SZ	114
115	REG_SZ	115
116	REG_SZ	116
117	REG_SZ	117
118	REG_SZ	118
119	REG_SZ	119
12	REG_SZ	12
120	REG_SZ	120
121	REG_SZ	121
122	REG_SZ	122
123	REG_SZ	123
124	REG_SZ	124
125	REG_SZ	125

Computer\HKEY_LOCAL_MACHINE\Software\1000\0

June, 2014

Contents

1 Introduction

The purpose of this research paper is to document the effects that registry items have in Microsoft App-V Virtual Application Packages.

The effort is squarely aimed at answering questions on how the detection/deployment of these components 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 Virtual Registry

The App-V 5 Virtual registry is delivered as a “Dat” file inside the App-V package. This is based on a standard format, and may be imported onto a registry key for debug viewing.

Registry items have a “type” field, which indicates whether the item is a key or certain kinds of data types, including DWORD (a four byte integer), REG_SZ (a string), and REG_BINARY (a variable length binary blob).

App-V extends the registry items types a little, for example with types that signify that it is a deletion marker, an entry that indicates that a local client value should not be seen. Microsoft has not yet documented these types, but we can see them when we import the .Dat file for debugging as unknown types.

When an App-V package is published, the Dat file is pre-staged. This means that the entire file is streamed from the source and a copy is placed in C:\ProgramData\Microsoft\AppV\Client\VREF folder as a DAT file using the name of the package Version GUID.

The first time that a package is run by any user on the OS, the complete registry staging occurs. It is at this time, delaying the app launch, which the pre-staged DAT file is read and a copy of its contents is placed under the key:

HKLM/SOFTWARE/Microsoft/App-V/Client/Packages/PkgGUID/Versions/PkgVerGuid/REGISTRY

This is a read-only copy used by the client.

3 Summary of Where Impacts of Registry Are Felt

Of all of the App-V subsystems that I have tested, the virtual registry is the most efficient of all of the sub-systems.

Package cleanup to remove unnecessary registry entries from App-V packages is *not* recommended for performance as the amount of cleanup is likely to not be detectable by the end user. Additionally, thanks to virtualization, the need to perform the sorts of cleanups that must be used in other application re-packaging techniques is generally non-existent in App-V. Removal of entries when sequencing is much more likely to break the package than be of any benefit.

A large virtual registry within a package affect performance in several ways:

- An increase in size of the .AppV file directory index causes more data to be streamed during the Add-AppVClientPackage step. One individual item probably makes no difference, but a large number of them does.
- During Publishing Phase, the DAT file is streamed and a copy made (pre-staging).
- During first run operations, staging is performed.

Additional performance depredation at runtime, by actively using the virtual registry is likely, but not tested as part of this effort.

The testing did not attempt to look at different kinds of registry entries, assuming that the deployment overhead would be fairly consistent and independent of content.

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 Test Packages Utilized

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 LotsOfNothing (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 VC Runtimes place on the system.

5.1.3 LotsOfReg

This package consists of a package consisting of 50 registry keys each containing 1000 REG_SZ entries, for a total of 50,000 entries. Each entry contains one to three characters. These entries are not referenced when launching the package.

[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.

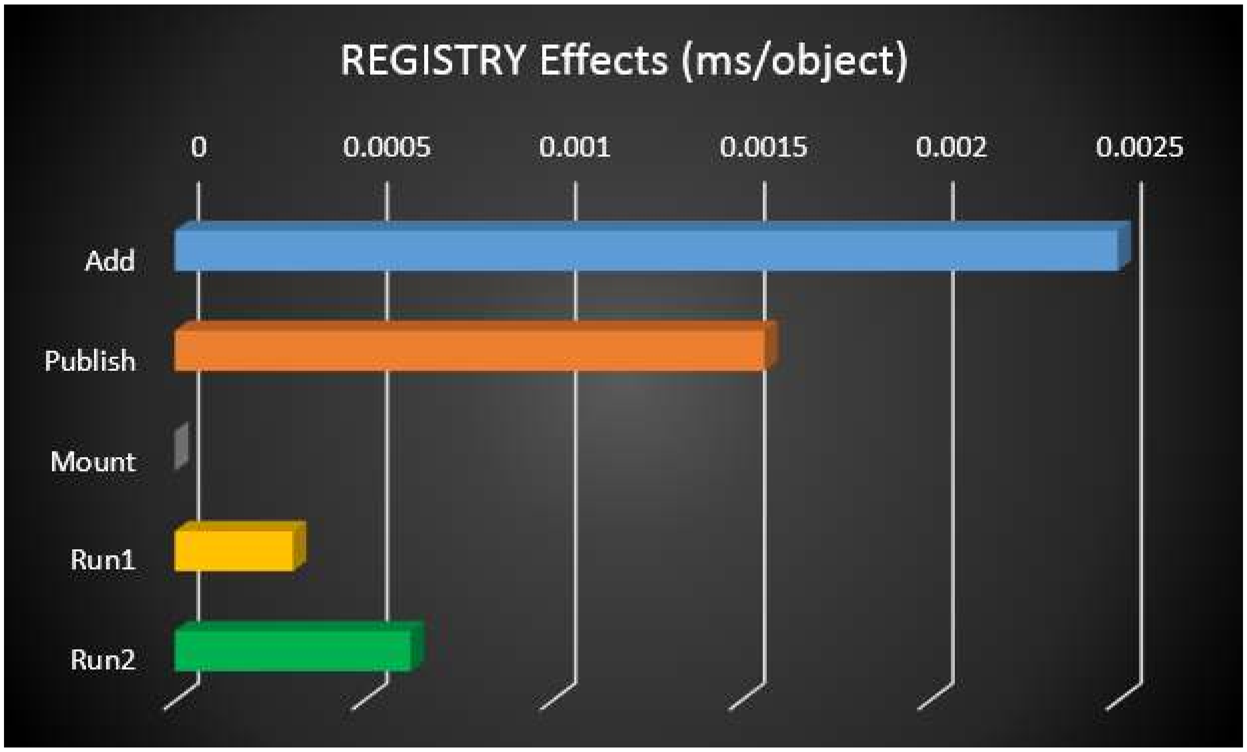
6 Detail Test Results

This section provides additional details of the testing results not reported in the summary.

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!

6.1 SCS Mode Testing with Mounting

Tests were performed with and without Mounting and with and without SCS Mode enabled, however the effects of the registry are independent of SCS mode; only the mounting results are shown below.



From these measurements, I conclude that the size of the virtual registry is pretty much irrelevant.

From the numbers we can reach the following conclusions:

IT TAKES ABOUT 400 ENTRIES IN THE VIRTUAL REGISTRY ADD 1MS TO THE ADD STEP, 667 TO ADD 1MS TO THE PUBLISH STEP.

IT TAKES ABOUT 1500 REGISTRY VALUES IN THE VIRTUAL REGISTRY TO ADD 1MS TO THE FIRST USER LAUNCH OF THE PACKAGE.

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