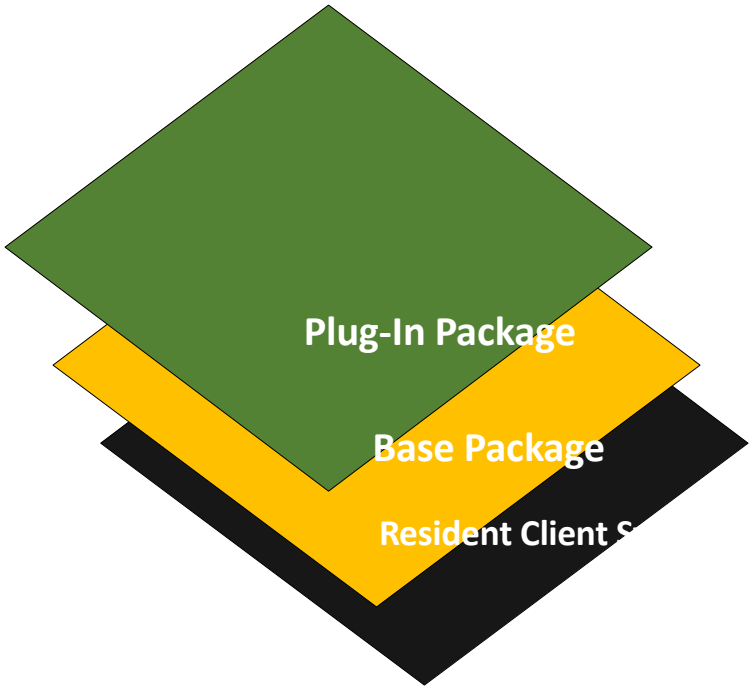
Pellucidity and Deletion Markers:
Connection Group Layering in App-V 5

A Research White Paper by TMurgent Technologies



Tim Mangan,
The Kahuna at TMurgent Technologies
November 7, 2013

About the Author

Tim Mangan, and his company TMurgent Technologies LLP, are dedicated to supporting enterprises, consultants, resellers, integrators, and Independent Software Vendors, in their understanding of Microsoft App-V. Tim originally ran the development group at Softricity when the platform was built. Microsoft acquired the company and renamed the product App-V. He is recognized by Microsoft as a "Most Valuable Professional" (MVP) and by Citrix as a "Citrix Technology Professional" (CTP), and is well known and sought-after speaker around the world. Tim has written several books and his website is a frequent bookmark for those looking for information and tools related to App-V. TMurgent provides training and consulting to the App-V community.

Background

In November 2012 Microsoft released a completely new version of their Application Virtualization product, Microsoft App-V. This new version was, for the most part, a complete rewrite of the virtualization stack. This new version behaves quite differently, and we are still learning the differences. One striking, but unexpected, difference is how layering is performed when more than one package is grouped together.

I authored a white paper "*App-V DSC and Transparency Revealed*"[1] in June of 2010. This research paper examined how layering was performed in App-V 4.6 using Dynamic Suite Composition (DSC). That deep look into the details of how the layering was implemented was incredibly valuable to troubleshooting large packages being grouped together with DSC.

While Microsoft has not documented the new behavior of DSC's replacement, Connection Groups, it turns out it is implemented differently when you examine the details. That 4.6 white paper is now not only obsolete, but misleading if you assume the details work similarly in App-V 5. Additionally, the way deletion works in packages has never been properly documented (by Microsoft or myself), so it is high time that I did so.

How important is this? Not at all. Well, most of the time. I was working with the product for over a year before I noticed the differences! But if you try to bring together two large packages that share a number of registry keys and file folders, you may run into an issue that you will not understand if you do not have a solid base of knowledge on how the product actually works.

So here is a replacement to that white paper.

[1] Available online at http://www.tmurgent.com/WhitePapers/AppV_DSC_Transparency.pdf

1 Folder and Registry Key Pellucidity in a Package

Ultimately, a folder and a registry key are both just containers with some attributes. Folders are containers that hold files and subfolders. Registry Keys are containers that hold registry items and sub-keys. Attributes are usually things like permissions, but they can be much more. For example, a junction point is a folder with a special attribute that says "look over here"[2].

When dealing with virtual applications, we have the additional concept of overlaying one container inside of a virtual application package over another located on the base operating system. An important ability of a virtual package is to be able to control what the virtual application can or cannot see. Often, we want the package to be able to see its own contents transparently overlaid on top of whatever is present on the client system. But sometimes we want to hide what might be on the client system, as in an opaque layer. A common example of the latter is when we package an application that might have an older version natively installed on the client system[3]. To make this happen, the virtual folders and keys have an added attribute that I refer to as the **Pellucidity** of the virtual folder or key.

For the most part, the sequencer "does the right thing" in deciding when to make things transparent and when to make them opaque, and setting the Pellucidity correctly for us. Inside the sequencer, Microsoft exposes this attribute as two mutually exclusive properties called "Merge with local" and "Override local" that allow us to override the default behavior.

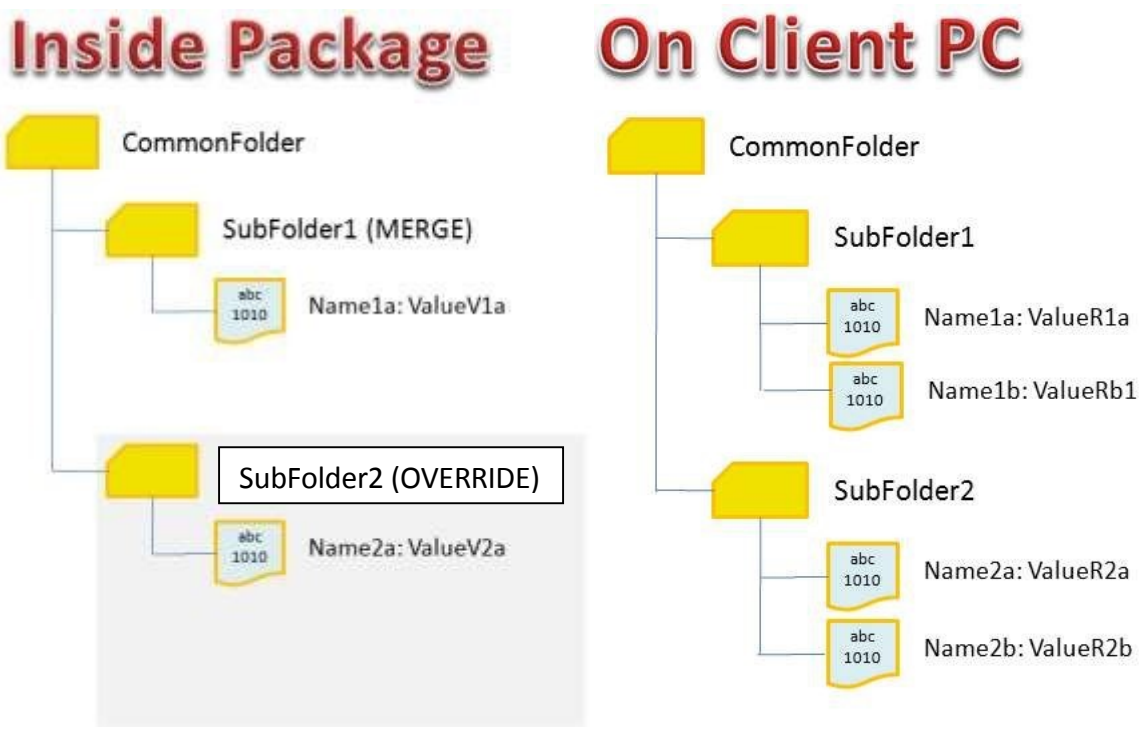
- When the overlay container is marked "Merge with Local", the pellucidity of the overlay container is transparent, allowing the application to see additional sub-items that are not part of the overlay container.
- With "Override Local", the pellucidity of the overlay container is opaque, hiding the items in the underlying container.

Individual files, or registry items, do not have a pellucidity setting because they are not containers.

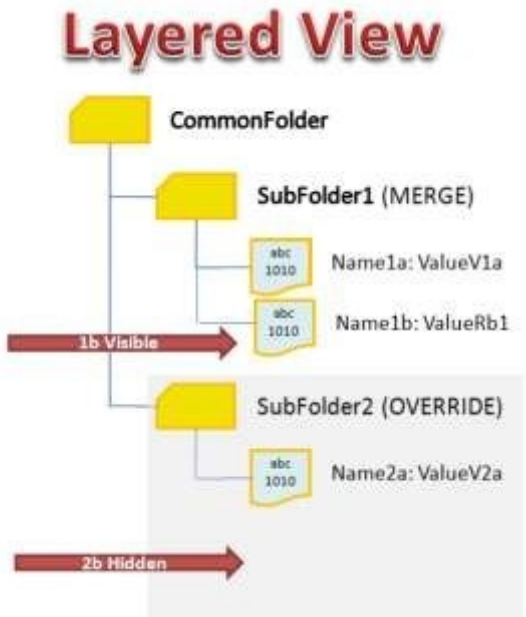
An example of this is in the illustration below. The Subfolder1 and Subfolder2 objects are containers that exist both inside the package and on the native OS. Inside the package, SubFolder1 is marked "Merge with Local", while SubFolder2 is marked "Override Local". These could represent not only a file folder, but also a Registry key.

[2] OK. Junction points are way cooler than that, but we don't need to get into that here.

[3] Microsoft is quick to point out that they do not support this scenario, but they use this as a selling point and most of the time it works anyway.



When brought together on the client, the virtual application would see a virtualized file or registry system that looks like this view:



To make the right choice (most all of the time), the sequencer follows a set of very simple rules when capturing during monitoring mode:

1. Ignore anything that is under the configurable exclusion containers.
2. Ignore anything that was done by a pre-existing process.

3. Only put containers in the package that have activity during monitoring (adding, removing, or modifying) somewhere underneath the container, either directly or many levels down.
4. If the container existed before monitoring began, mark the container “Merge with Local” (transparent pellucidity), otherwise mark it “Override Local” (opaque pellucidity).

This behavior allows (for example) the application to add additional dlls to the System32 folder. Because the System32 folder pre-existed on the OS prior to sequencing, it is marked transparent, allowing the application to see the packaged dlls plus any native dlls under that folder. The only native thing blocked from view of the application would be identically named dlls, which allows the package to not only add dlls but to “replace them”.

Meanwhile, a folder like C:\Program Files\VendorApp” would normally be created by the installer for the application during sequencing, and thus marked is marked “Override Local” by default. This is how App-V prevents the packaged version of an app from seeing the older (or possibly newer) native version of the same app that is natively installed.

These settings can be seen in the Sequence Editor of the App-V Sequencer. Folders and Key containers have a different look depending on their pellucidity setting[4], and you can change the default behavior by right-clicking on the container.

[4] In my view, the icons used to show this are inconsistent (backwards) between those used in the virtual registry and those used in the virtual file system. When in doubt, I always right-click the object to confirm which setting is active.

2 Deletion Markers in a Package

A package folder, key, individual file, or individual registry item may also have a special attribute that is indicates that the item is marked as deleted.

This occurs when the item exists before monitoring mode begins and is removed during monitoring. The idea is that this is a signal to the client to hide the application from seeing this item if it exists in a lower layer.

When an item in a package has a deletion marker, you simply do not see the item in the sequence editor, it unfortunately does not show any indication of the deletion marker. Using a tool such as GridMetric's Application Virtualization Explorer, we can see the item and marker, which is helpful in troubleshooting issues that might arise.

3 Pellucidity in App-V 4.6 DSC

When we start dealing with Connection Groups, we now have to think in terms of more than two layers, and the possibility of the Pellucidity being different on the different layers.

In App-V 4.6, In terms of what the application would see, this situation was a fairly straight-forward extension of the Package layered over Client case. The setting on any given layer affects what can be seen that is “below it” (in the sense of the cover art for this white paper).

The real purpose of the old white paper was to examine what happens when the virtual application changes one of the items inside the container (files or registry items). In App-V 4.6 this was a complex equation because the modified entity had to be stored in the PKG file associated with one of the packages (base package or plug-in package). The results, when I tested this were non-obvious. Even worse, the results were completely different depending on if you were testing the registry or the file system.

You can still read that white paper if you need to see the details.

4 Pellucidity in App-V 5

In App-V 5, Microsoft replaced DSC with a similar concept called “Connection Groups”. As a simplification, we can think of Connection Groups as being manageable DSC connections.

Importantly, Connection Groups are a defined object, one that is assignable and understood as an object at the client. The Microsoft implementation of Connection Groups significantly also solved the problems pointed out by the original paper in handling changes made by a virtual application. Ultimately, when virtual applications run inside a Connection Group, any changes made are associated with the Connection Group object and not individual packages.

This change, along with no documented changes by Microsoft in how the layering worked in App-V 5, led me to believe that the Application view inside a connection group would be a similar extension to the single package situation.

Until I finally got around to testing it. It turns out it is different. And maybe has a bug.

Before I get into the tests and results, let me emphasis that most of the time, it probably doesn’t matter. Except, of course, for when it does.

5 Pellucidity Test Setup

Rather than deal with real applications, containing unknown source code and doing who knows what, I devised a test setup that did not rely on application software. Only OS components, such as the windows explorer, the "type" command, and registry editor are used.

I created a set of eight packages. Four packages contained folders and files, and four others contained registry keys and items.

Within either group of four, two are designated as "base packages" and two are designated as "plug-in" packages.

Within any group of two, one has the container marked as "Merge with Local" and the other "Override Local".

Eight Connection Groups are then created, four for the folder/file packages and four for the registry key/item packages. Each connection group would have a unique combination that contains both a base and plug-in package, forming pairs of "Merge/Merge", "Merge/Override", "Override/Merge", and "Override/Override".

The base and plug-in packages use the same named container (folder or registry key), but have different sets of sub-items (files or registry items) under the container. The test client also has the same named container present on the base OS, and a different combination of sub-items present under the container. This allows us to develop a complete view of all of the possible combinations[5].

The file based packages were created by creating a folder (named "Folder") at the root of the C: drive. The PVAD for each package was a differently named folder. Inside the folder were placed a subset of files named from "A.txt" to "G.txt". The contents of each text file uniquely identify the file and package.

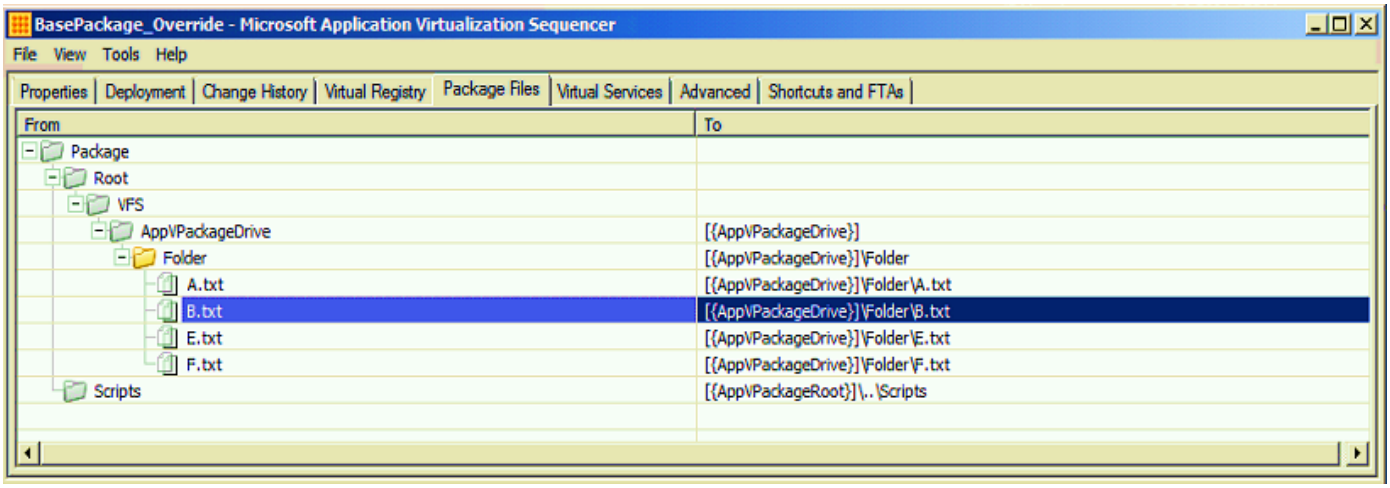
The registry based packages were created by creating a key (named "TestKey") under HKLM\Software. Under the key were placed a subset of string items named from "A" to "G". The contents of each reg string uniquely identify the file and package.

The contents of the packages are shown in the following images that follow.

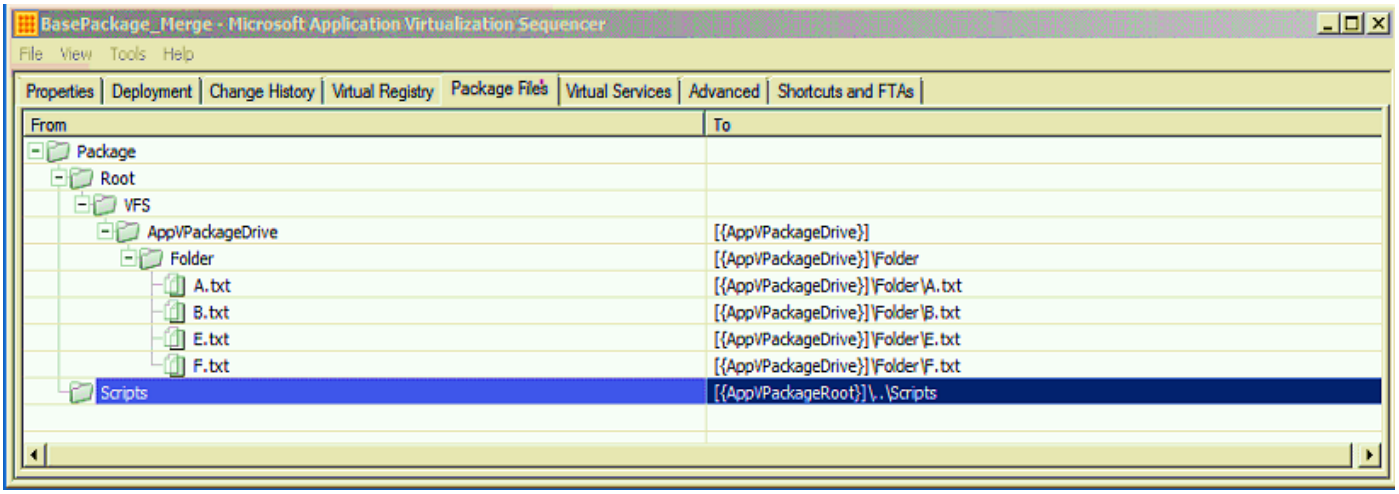
[5] Except for "deletion objects" that can exist in packages. That will be a test covered later in this paper.

5.1 File based Test Setup

Here is a look at the file system of the base package with Folder set to Override Local. The package was created by creating the folder in monitoring mode, and then adding the four files.



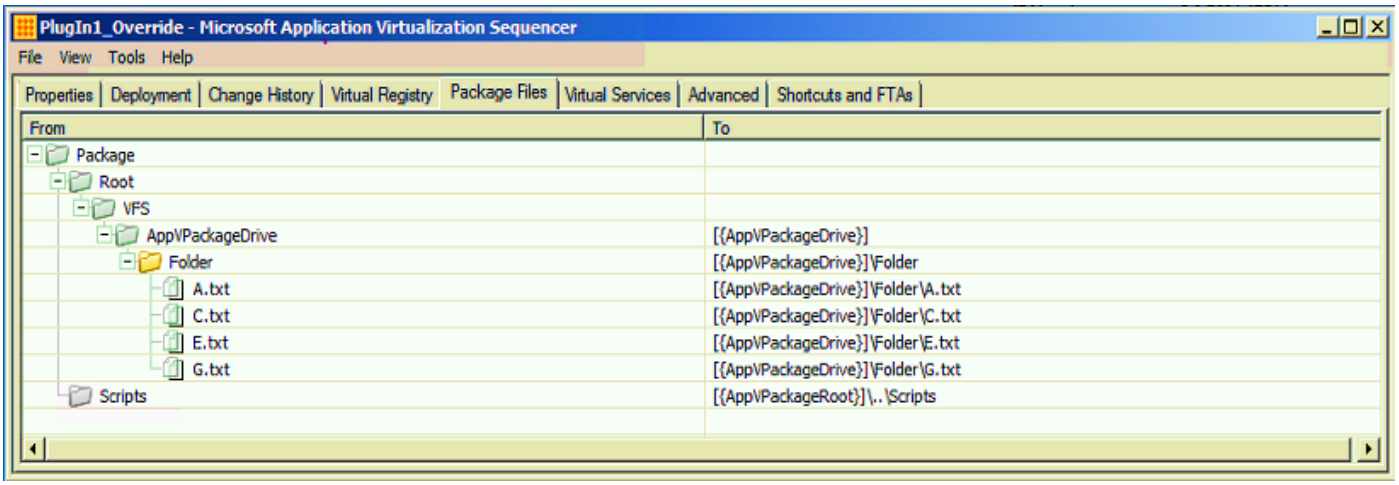
Here is the base package with the Folder set to Merge with Local. Notice that the coloring of the folder named Folder is different with the different Pellucidity setting.



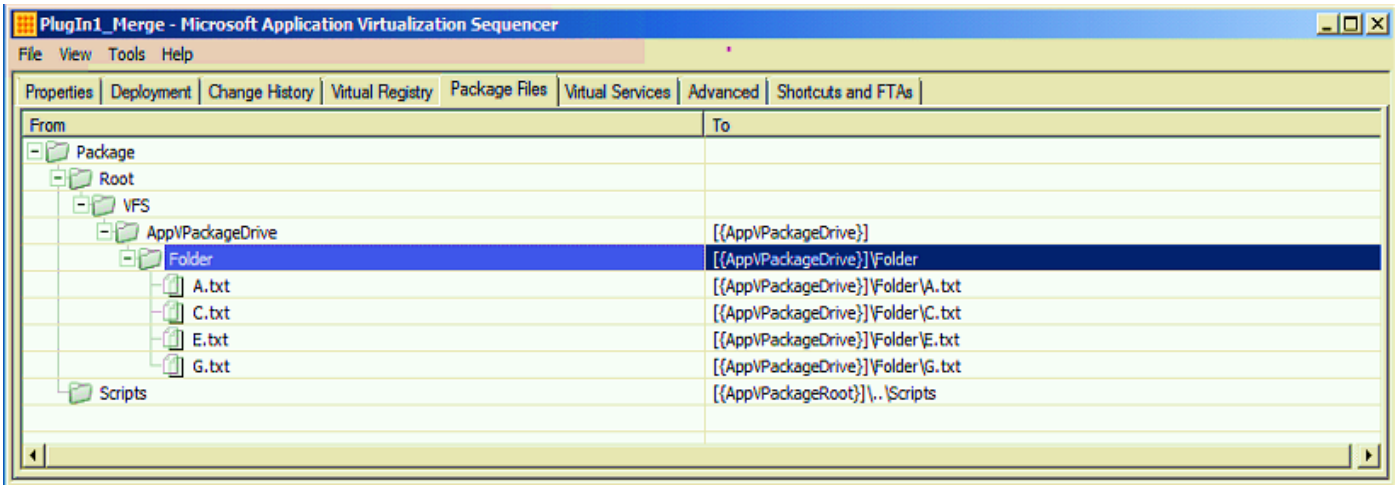
That package can be created in one of two ways:

- Pre-create Folder before monitoring. Then just add the files while in monitoring mode.
- Create the folder and files while monitoring. Continue to the sequence editor, right click on Folder, and select “Merge with Local”

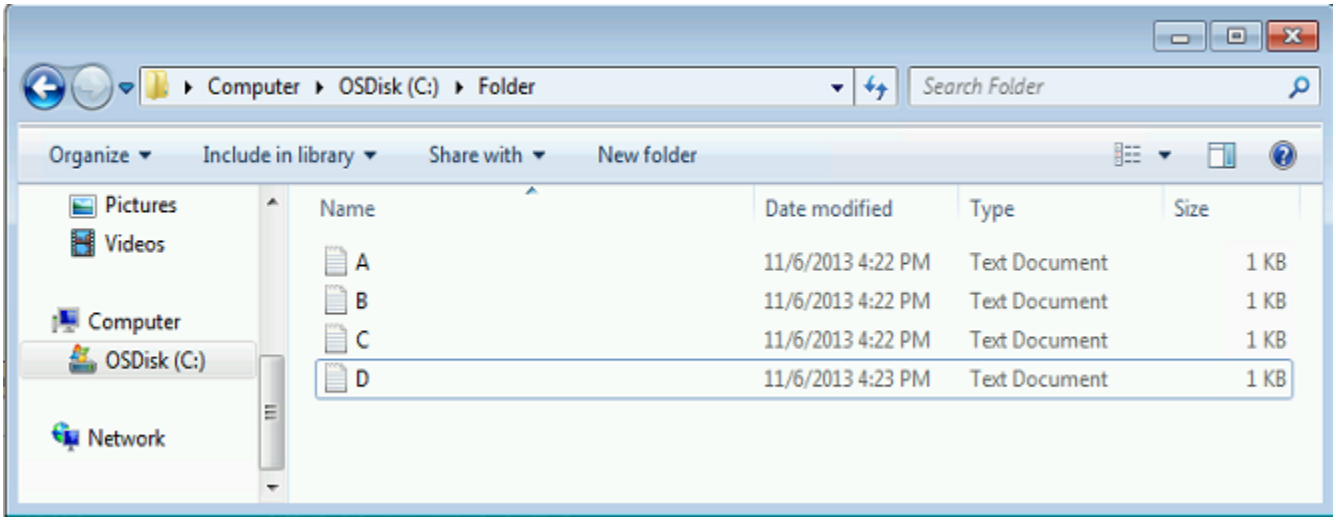
Now here is the PlugIn package with Folder marked Override Local. Following normal processes for building PlugIns this wouldn't normally occur in a Plug-In package, but if you combine two packages into a Connection Group that you didn't originally plan to group, this could happen.



And here is the Plug-In package with Folder marked Merge with Local.

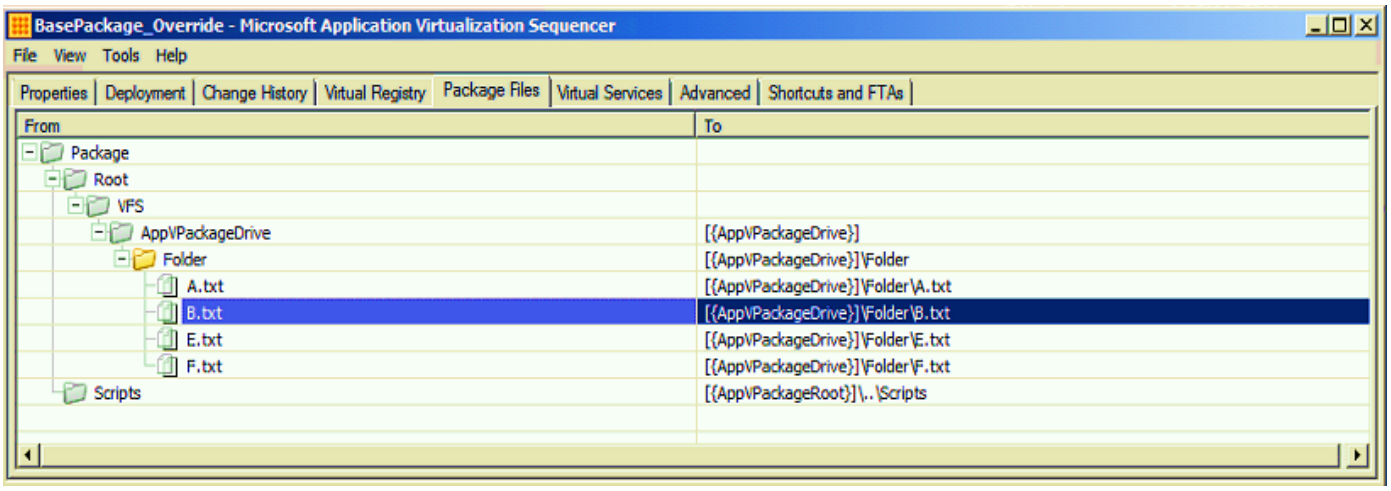


And then on the Client, the following files exist natively on the system.

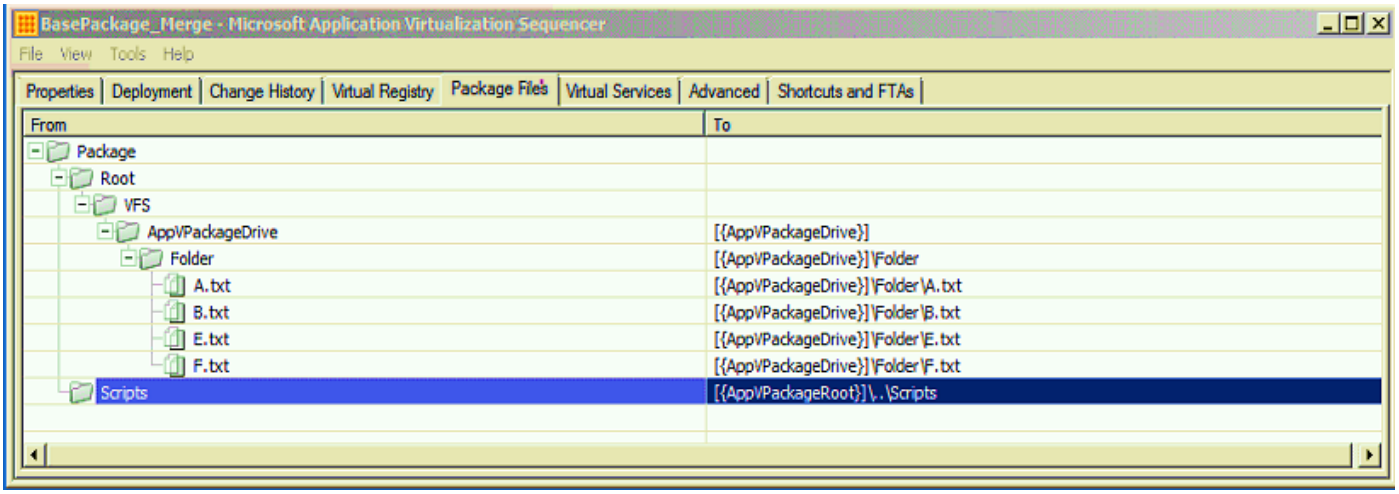


5.2 Registry Based Test Setup

Here is a look at the file system of the base package with Folder set to Override Local. The package was created by creating the folder in monitoring mode, and then adding the four files.



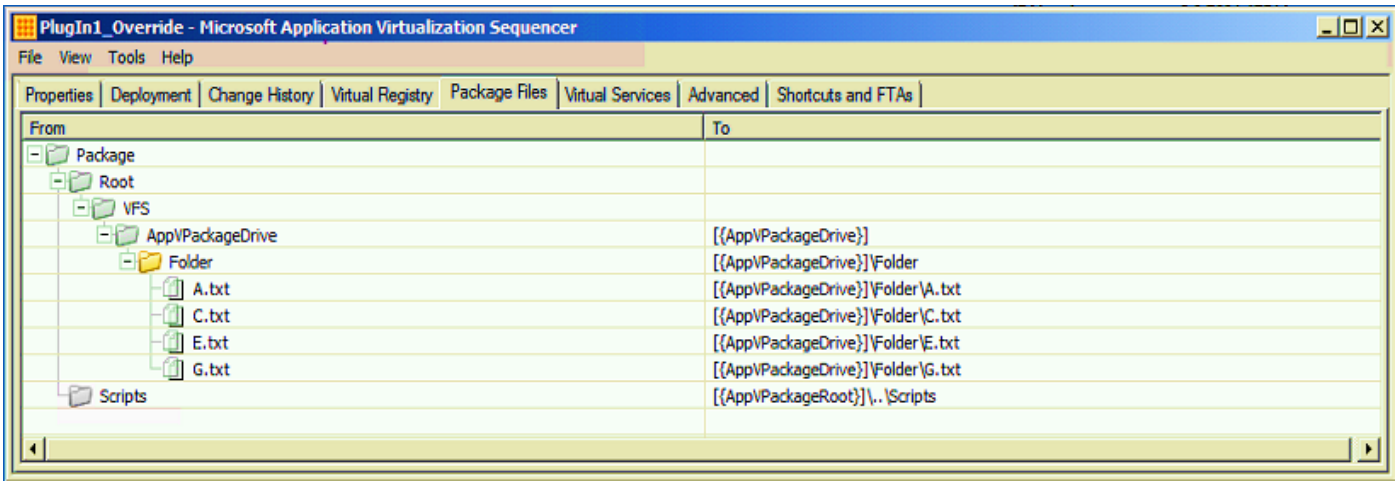
Here is the base package with the Folder set to Merge with Local. Notice that the coloring of the folder named Folder is different with the different Pellucidity setting.



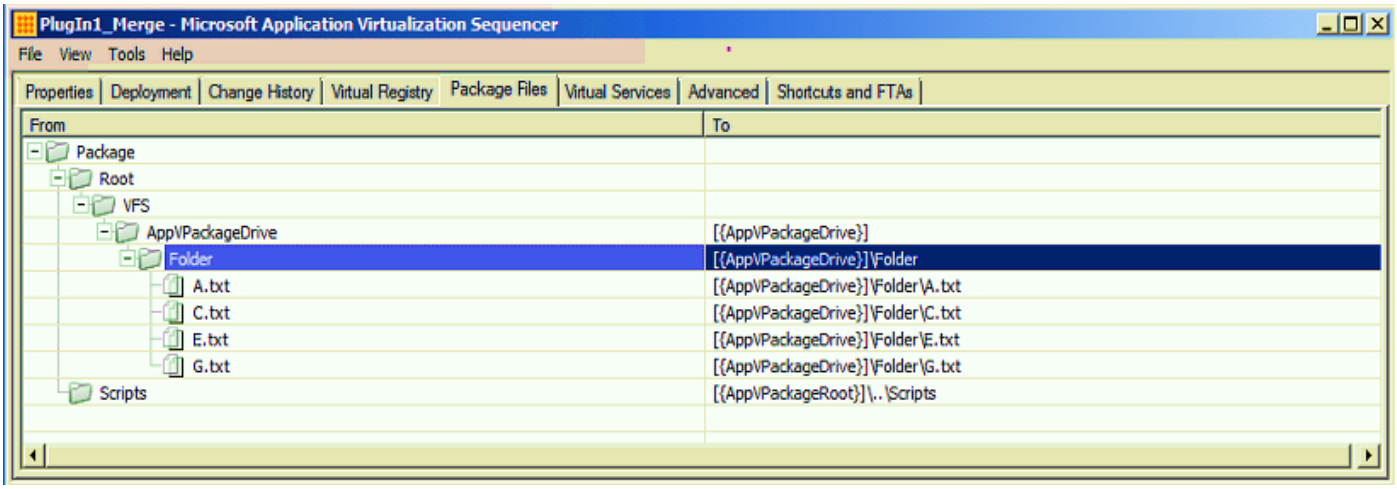
That package can be created in one of two ways:

- Pre-create Folder before monitoring. Then just add the files while in monitoring mode.
- Create the folder and files while monitoring. Continue to the sequence editor, right click on Folder, and select “Merge with Local”

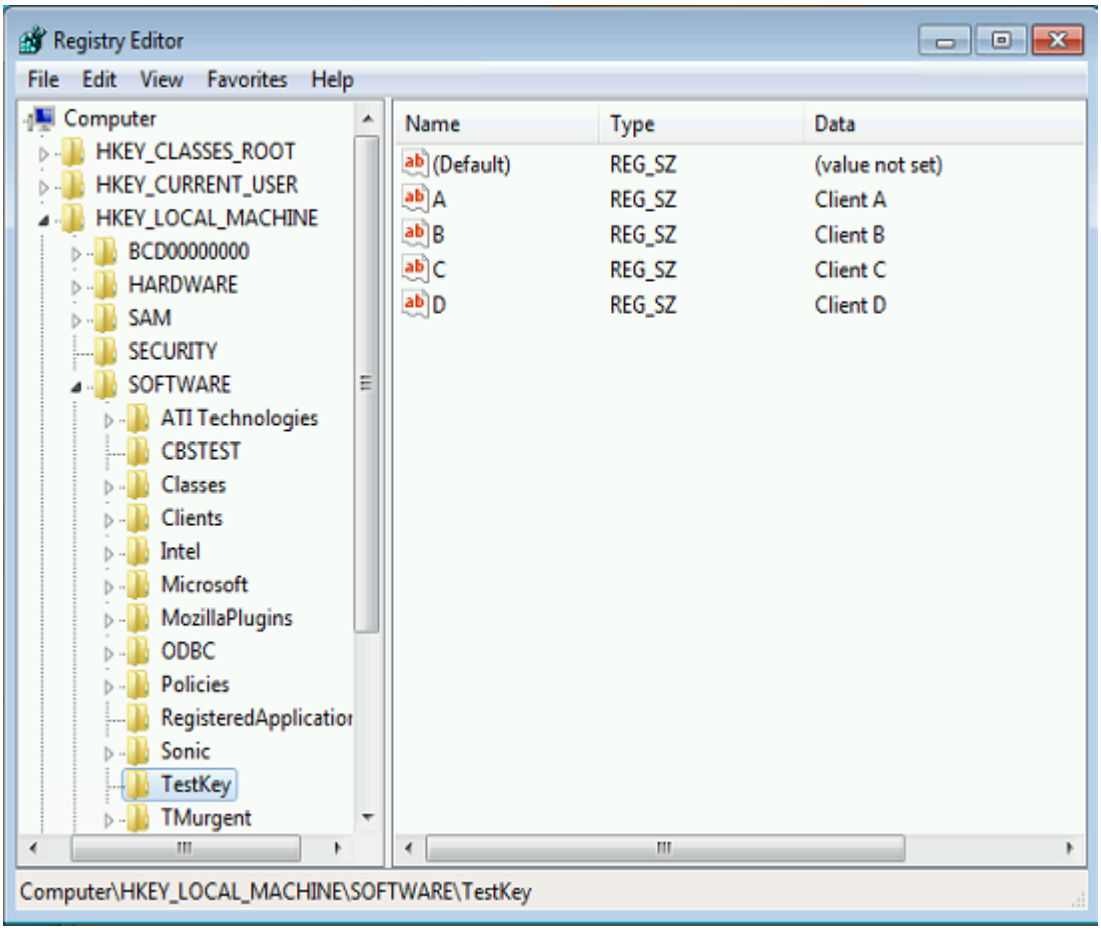
Now here is the PlugIn package with Folder marked Override Local. Following normal processes for building PlugIns this wouldn’t normally occur in a Plug-In package, but if you combine two packages into a Connection Group that you didn’t originally plan to group, this could happen.



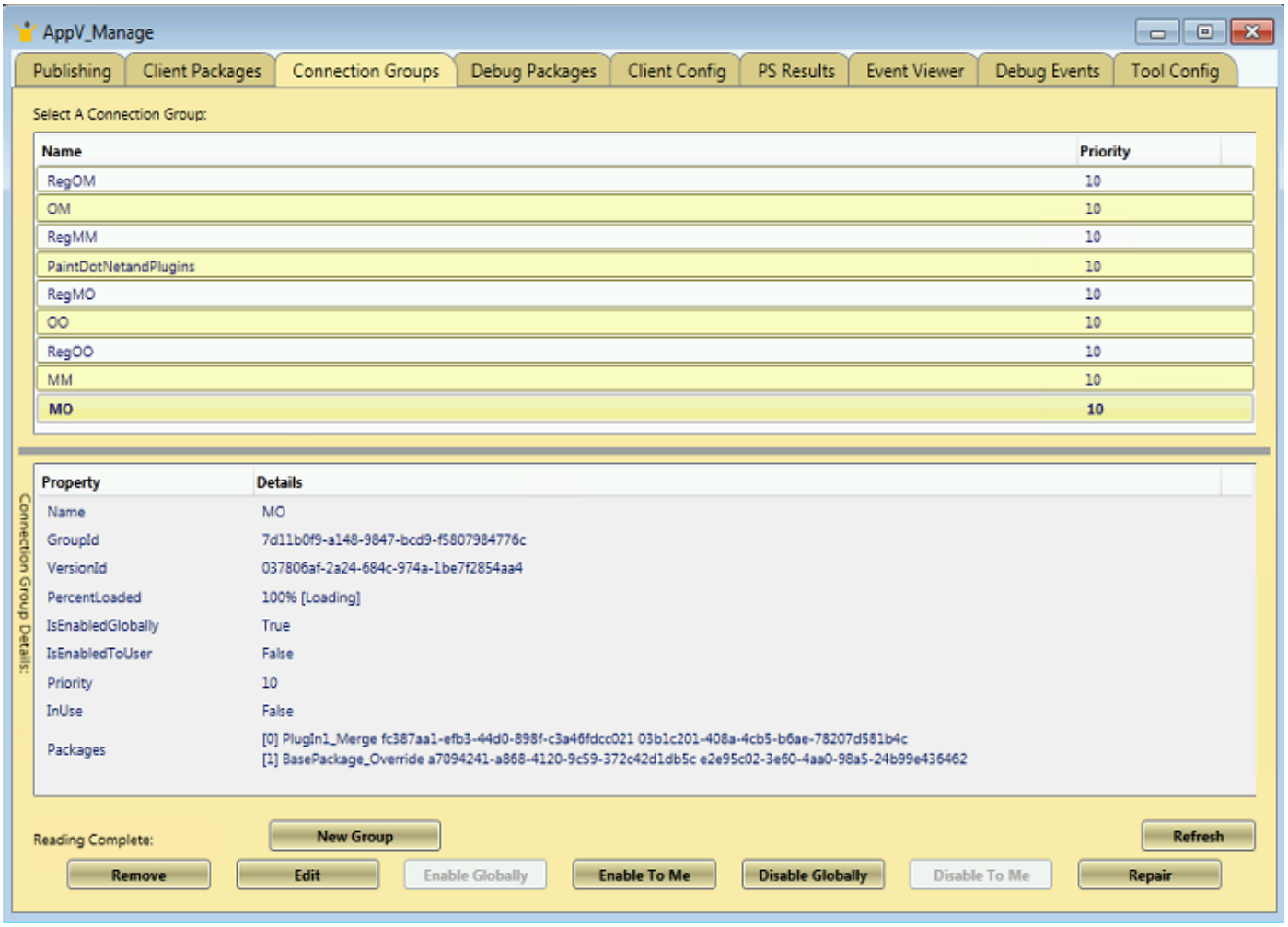
And here is the Plug-In package with Folder marked Merge with Local.



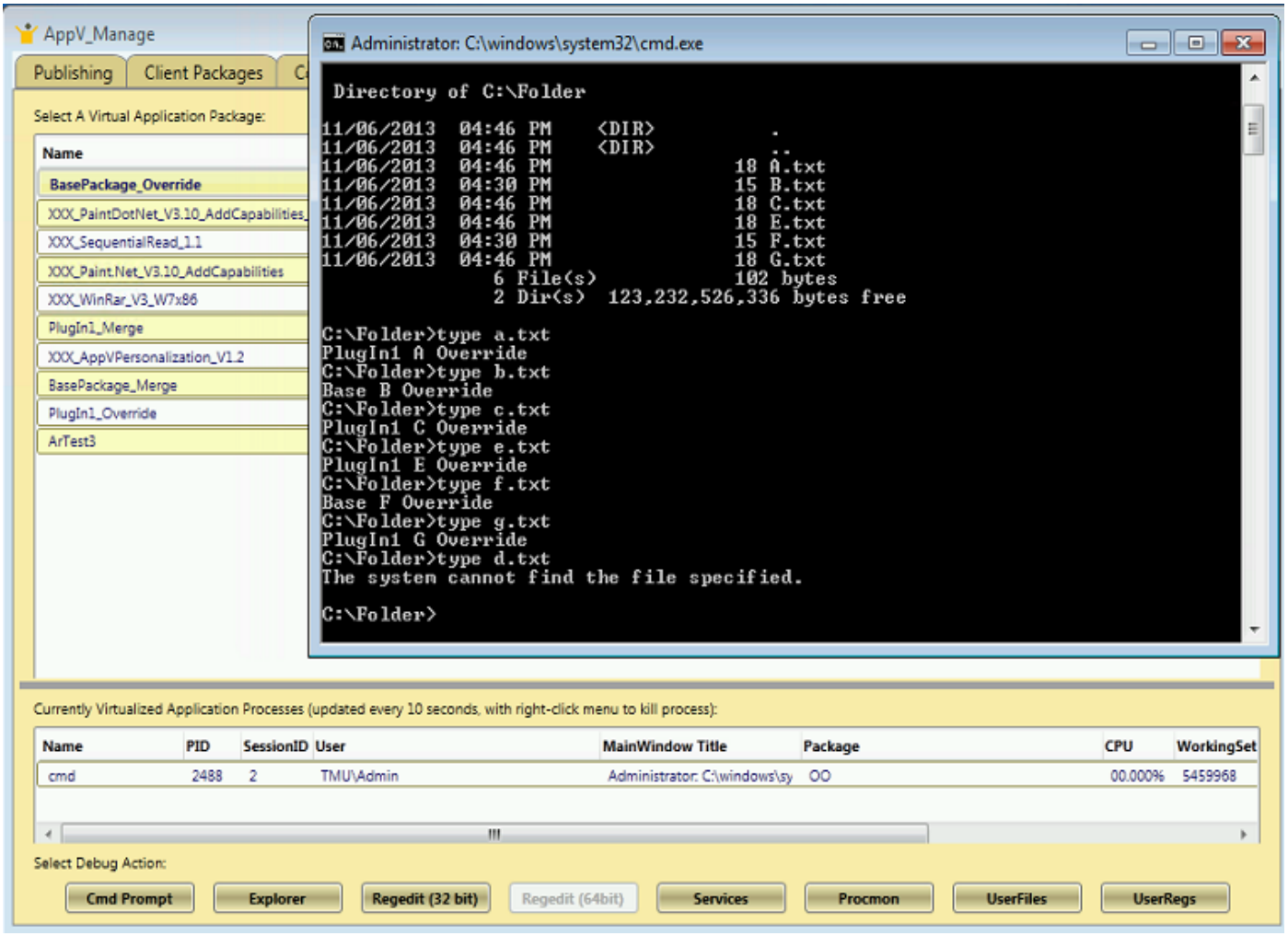
And then on the Client, the native registry on the system looks like this:



At this client, the Connection Groups are created to show the Application would see inside the virtual environment when it looks at Folder or Key. I used AppV_Manage locally on the client to create the connection groups and enable them one by one. Below is a screen shot of a typical connection group:



The screen shot below is from a test with a connection group consisting of the override version of the base and plugin packages. Notice that the virtual process monitor in the AppV_Manage tool shows the connection group name in place of the Package name; this is very helpful in ensuring which environment a virtual application is running in.



The test results are summarized in the next chapter. Some of the results, such as the one shown above, are different than would have been seen in an identical test against App-V 4.6. In addition, the file system and registry tests are different in only one case (potentially a bug in the virtual registry implementation at the client).

6 Pellucidity Test Result Summaries

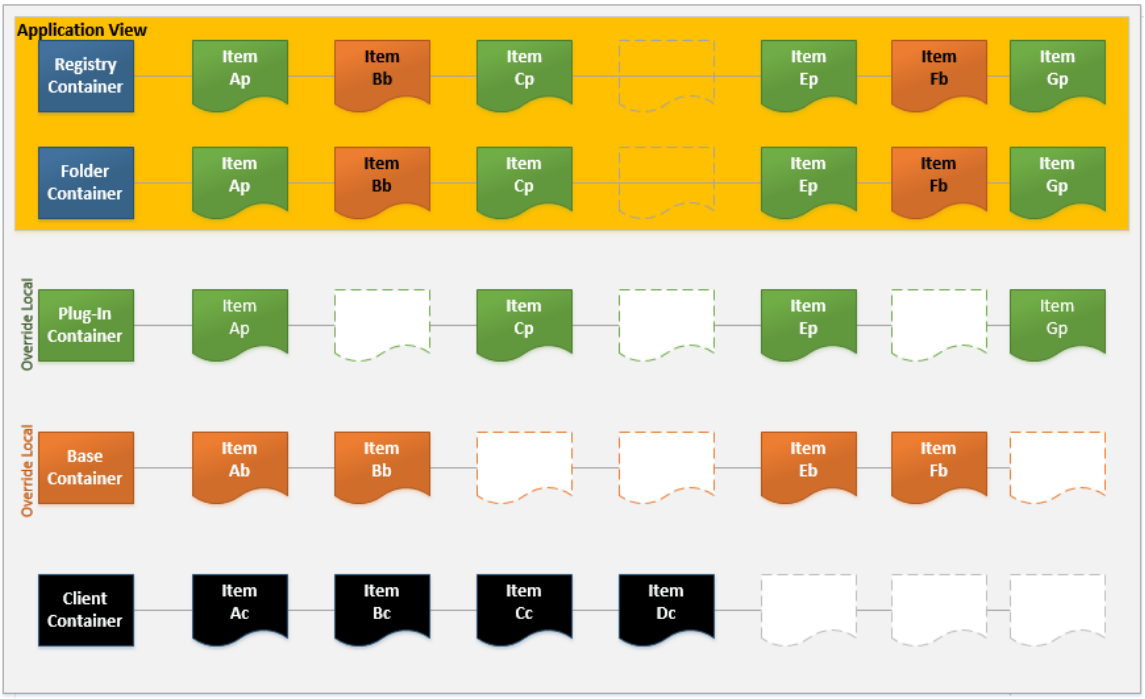
Both Merge with Local

Base Override, Plug-In Merge

Note: Item **Dc** has different results for file/reg. Reg result on that item may be a bug and is different result than for App-V 4.6

Base Merge, Plug-In Override

Both Override Local



Note: Items with **black** text are different results than for App-V 4.6

7 Pellucidity[6] Summary

Usually, a package with an add-on package or plug-in constructed properly would only cause the following situations:

- Both Merge: A, C, D, E, and G
- Base Override, Plug-in Merge: A, C, D, E, and G

Except for the virtual registry in the mixed case and item D, these "expected" situation results are consistent with App-V 4.6 behavior.

Additionally, if two packages are independently sequenced without regard to connection groups and then brought into a group, the likely additional possible situations would be:

- Both Override: all cases

While some of the test results may be unexpected, other than the potential virtual registry client bug they should not cause problems for *most* connection groups.

[6] OK. Did you look up that word yet? I realized that I needed a better term than I used in the past for the concept of how transparent or opaque a container was. So I'm going with Pellucidity because there is no chance that the reader will confuse the word to mean one setting or the other.

8 Deletion Marker Tests

To test deletion markers, a similar approach was taken in creating a base and a plugin packages.

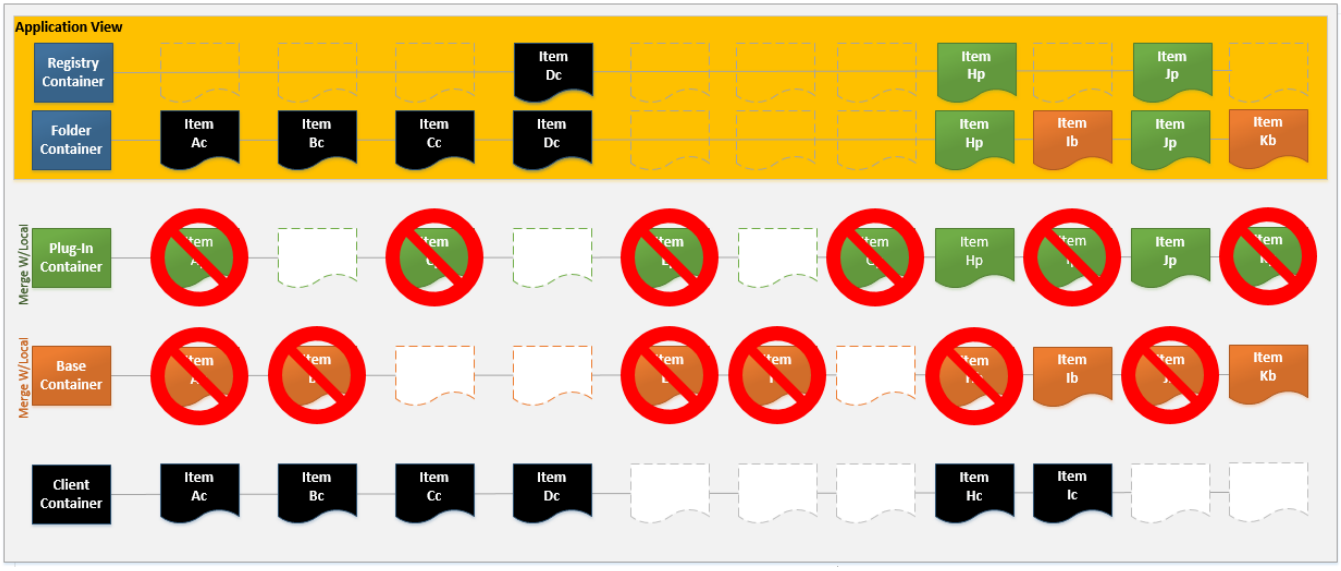
By combining both file and registry into a single package, I cut the number of packages in half. As I will explain in a moment, I did need to add an additional package, and ended up with eight connection groups.

In the first four packages, prior to running the sequencer the folder and registry containers were created, and appropriate file and registry items added (only those that needed deletion markers). During monitoring mode on those packages, the all items in each container were deleted (without deleting the container), and then any additional items added.

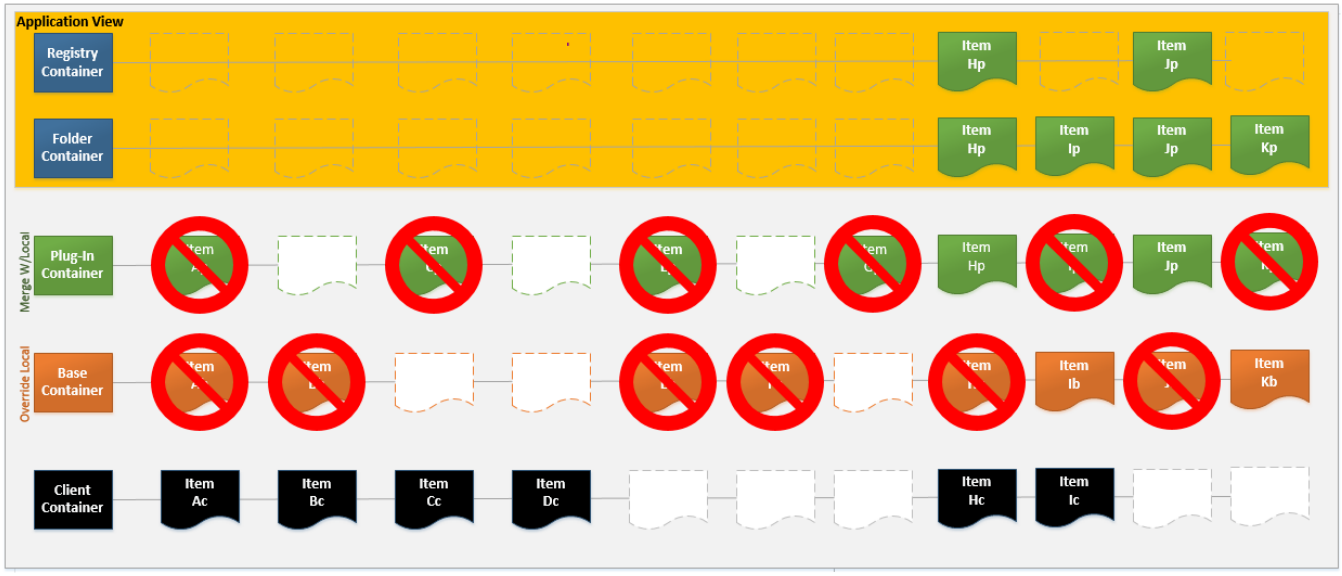
For the fifth package, prior to running the sequencer the folder and registry containers were created and left empty. During monitoring mode on this package the containers were deleted.

9 Deletion Marker Test Results

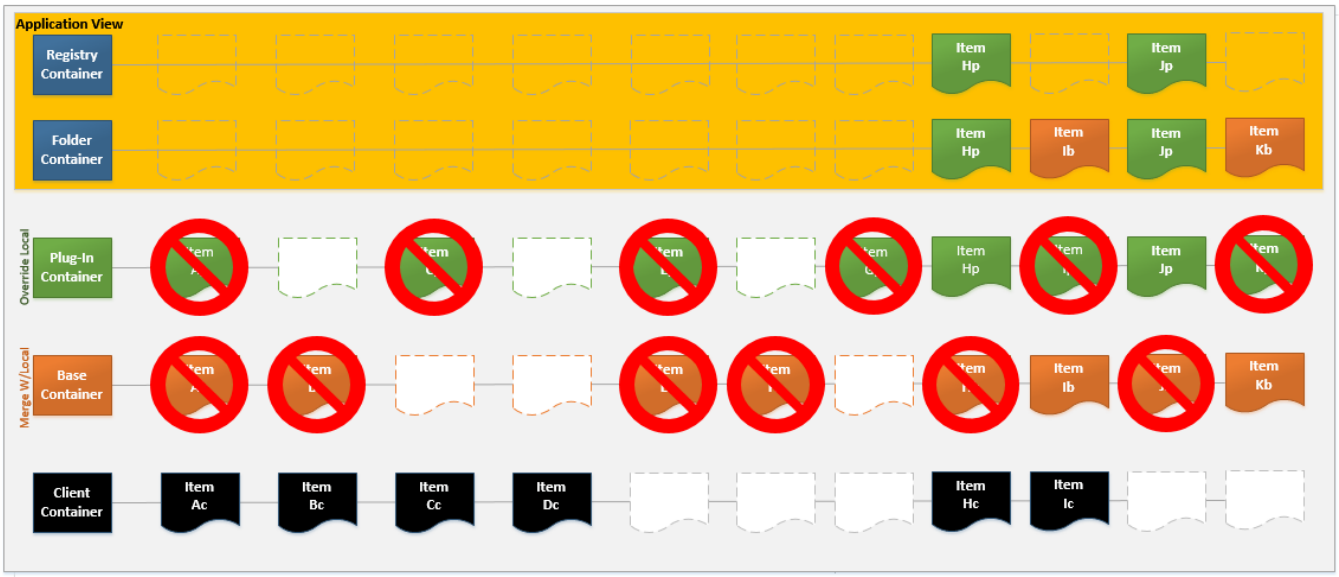
Both Merge with Local



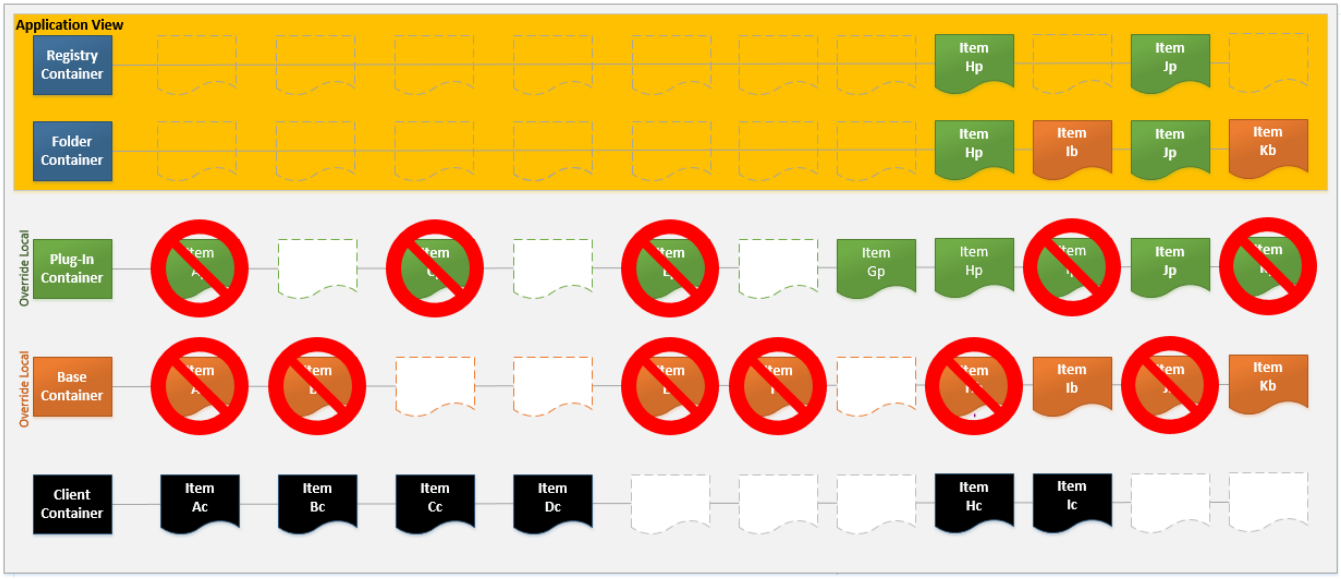
Base Override, Plug-In Merge



Base Merge, Plug-In Override

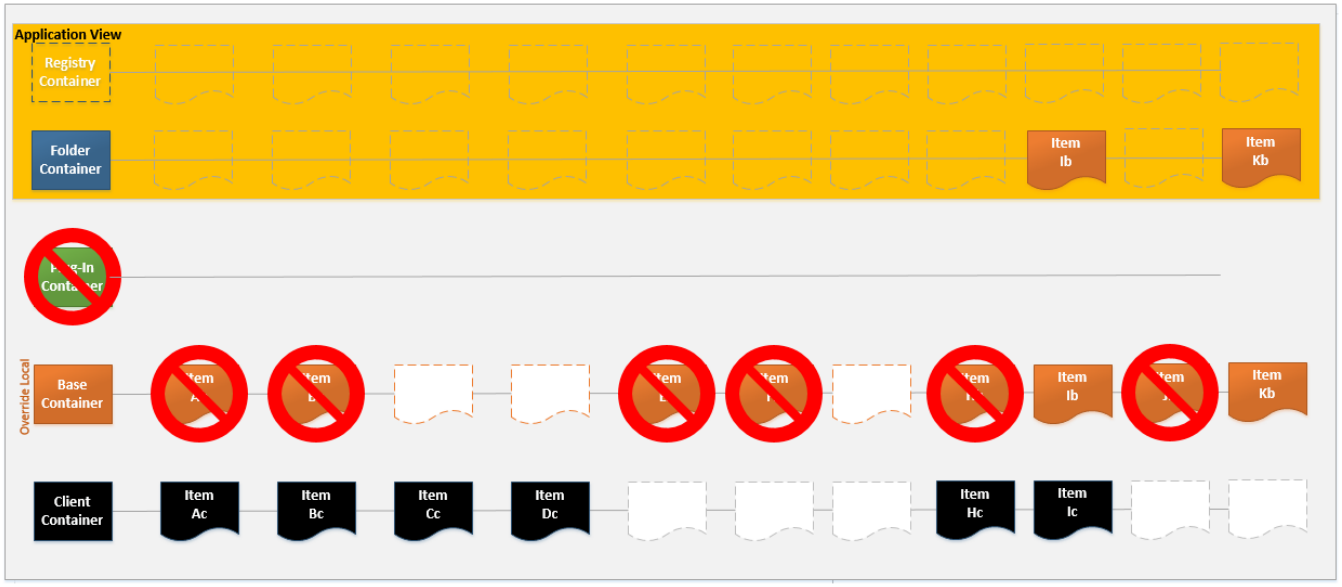


Both Override

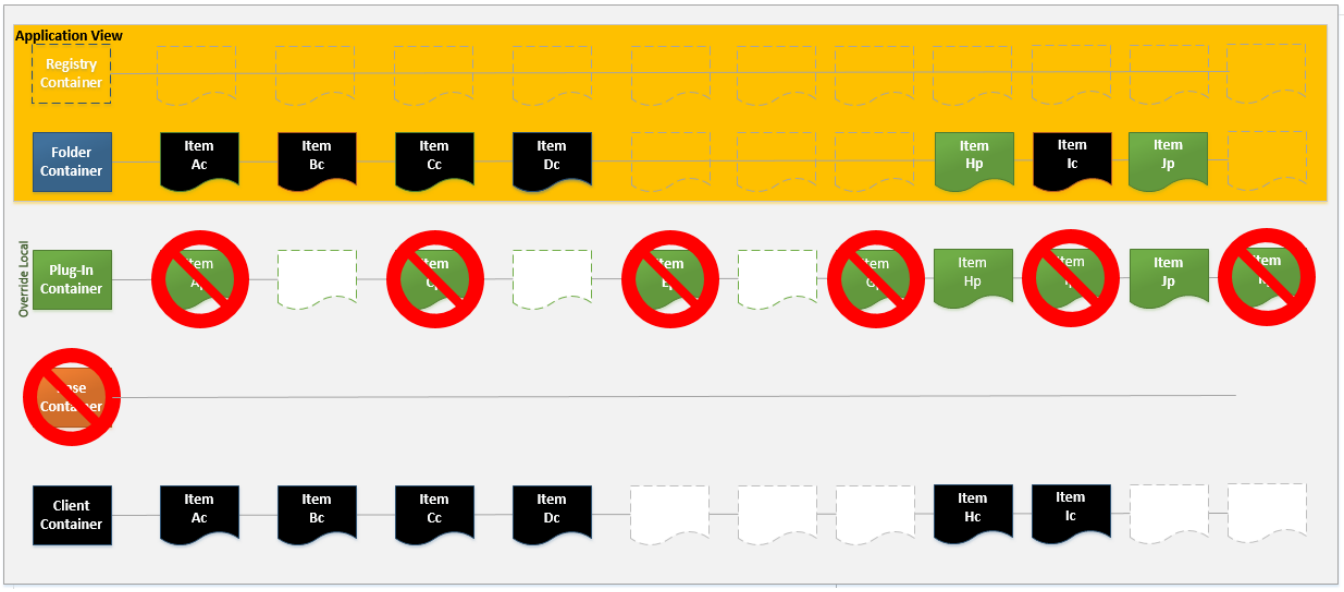


Base Merge, Plug-in Delete Container

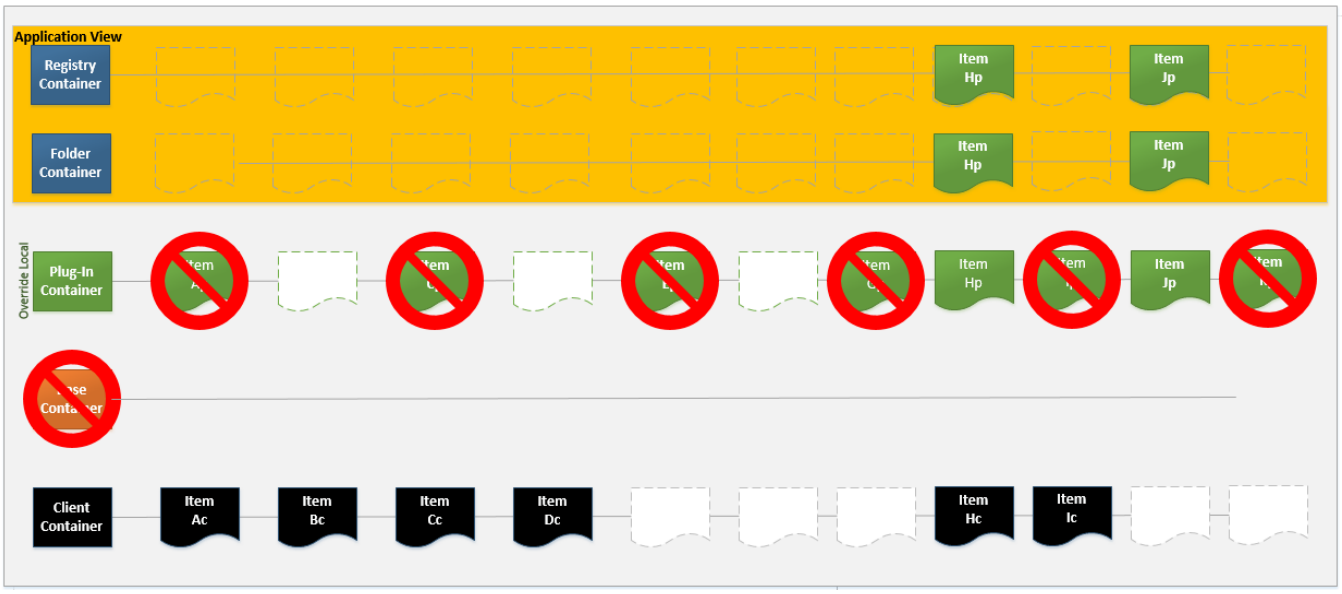
Base Override, Plug-in Delete Container



Base Delete Container, Plug-In Merge



Base Delete Container, Plug-In Override



10 Deletion Marker Summary

Usually, a package with an add-on package or plug-in constructed properly would only cause the following situations:

- Both Merge: all
- Base Override, Plug-in Merge: all

Additionally, if two packages are independently sequenced without regard to connection groups and then brought into a group, all of the container deletion scenarios are possible.

The dramatic differences in behavior of the virtual registry and virtual file systems of the client are very apparent in these deletion tests. The virtual registry implementation comes the closest to what I expected prior to testing, which looks to be the same as in 4.6.

It would seem that the virtual file system implementation for deletions is ripe to potentially cause connection group issues. When they occur it is something you should consider as part of your troubleshooting.

11 Final Summary

So the bottom line summary is as follows:

- Use connection groups when it makes sense.
- Understand how the system works and be aware of what can happen.
- Be careful when creating secondary packages for a connection group.
- *If* you have a problem, check what the application is seeing and verify any deletion markers and the pellucidity settings in the packages.
- If quick solutions do not become available, you can always package the items together in a single package.