

App-V DSC and Transparency Revealed

A Research White Paper by TMurgent Technologies

DRAFT

JUNE 1, 2010 : APP-V 4.6 RTM

TIMOTHY MANGAN

Brought to you by:



TMurgent Technologies, LLP

Your Destination for Application Virtualization

Training and education resources for Microsoft App-V

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, which was acquired by Microsoft for the technology, when the platform was built. 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 is also co-author of the upcoming book on the App-V Client, to be released in the Summer of 2010. TMurgent provides training and consulting to the App-V community.

Abstract

Two important, but now well understood, features of Microsoft Application Virtualization (App-V) are Dynamic Suite Composition (DSC) and transparency. This white paper documents research carried out by TMurgent into how these features behave and integrate at a detail level. Specific operational aspects are painstakingly documented here for the first time anywhere, revealing behaviors affecting virtualized application behavior with DSC is in use.

In the conclusion section, new recommendations towards “best practices” are outlined for building App-V sequences using DSC. Included in these recommendations are:

- All Application OSD files that will be used to launch applications for any DSC component *should* contain all DSC references, and in the same order.
- Folders and Registry keys that exist in more than one DSC component *should* be screened to ensure that the same transparency setting is used in each case .

Introduction

Dynamic Suite Composition, or DSC, is a new feature added to Microsoft Application Virtualization (App-V), allowing for separate virtual application sequences to be dynamically bound together at runtime to work as a single virtual application.

When this feature was released in App-V 4.5, Microsoft indicated that the initial version of DSC was for limited use, and eventually an improved version was being planned. Release 4.6, which is the focus of this research, contained no known significant changes affecting DSC. Microsoft depicts the support for DSC as being appropriate for “simple” application integration. Examples include applications that support dll “plug-ins” (such as many Microsoft Office applications or the Internet Explorer and other http browsers), and those that reference each other by launching an executable.

To use DSC, separate sequences are created for each application, or application component. After sequencing, the OSD files are modified to list dependent components. The App-V client then reads and processes these dependencies when the application package is started.

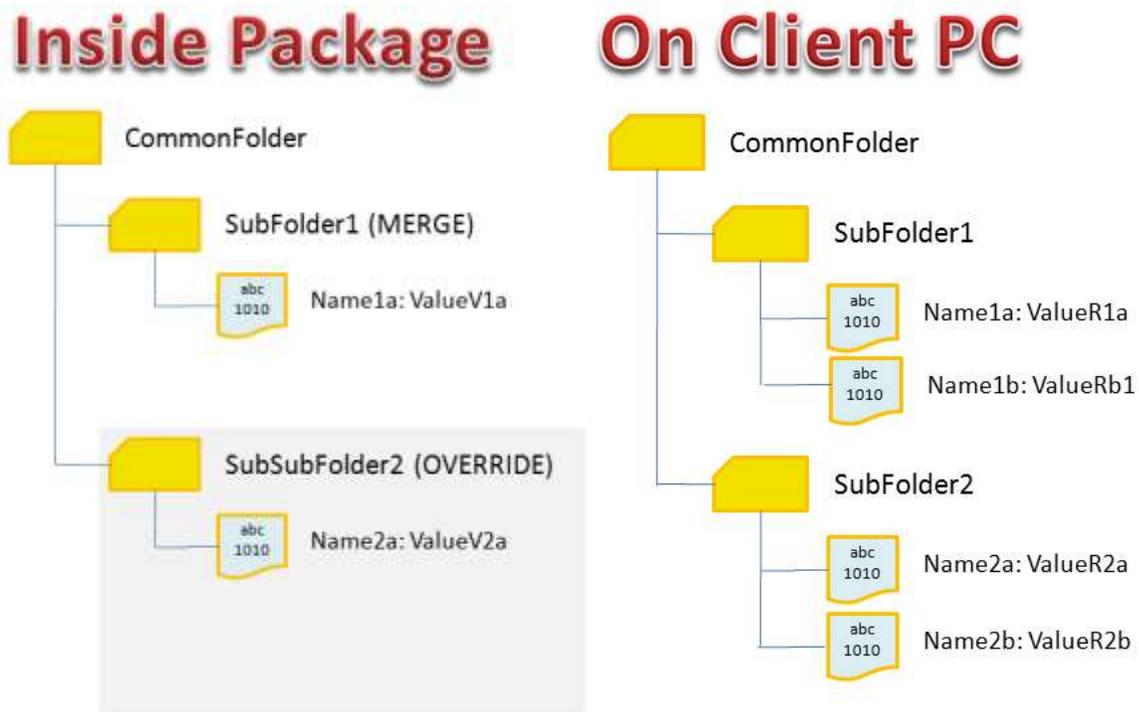
For DSC to work, folders and registry keys are layered over each other, and the focus of this research is first to understand how that layering occurs, and where runtime changes to these components are stored. This layering is not new in the product; App-V has always used layering to create an integration of package contents and user personalizations (stored in the PKG files) with the Client PCs resident registry and file system. With DSC, however, this layering becomes more important to understand.

Inside a SFT package, a folder or registry key (keys are in essence the equivalent to a directory folder inside the registry) has a setting that affects how this folder or key is layered. We refer to this as the *transparency* setting, although in the sequencer it appears as two exclusive options called “Merge with Local” and “Override Local”. Fortunately, without DSC, the sequencer automatically determines the appropriate transparency setting when processing the monitored application in almost all cases.

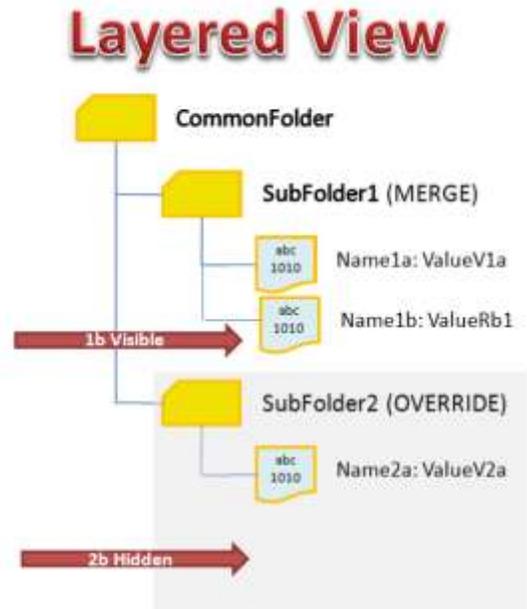
For example, if a folder or key is created new while monitoring, the sequencer will set the transparency to “Override”. This means that if the folder or key existed on the client PC, perhaps because an older version of the same application is natively installed, the virtual application will not see artifacts of the older version.

If files or folders, registry keys or name/value pairs are added to a parent folder or key, the sequencer would set the transparency to “Merge”. This allows the virtual application to see additional standard components under that parent that are part of the operating system and do not need to be included in the package.

The client uses these transparency settings to create a layered view (sometimes called the “virtual environment”) for the application. Below is an example showing the hypothetical contents of an SFT and those resident on a client PC.

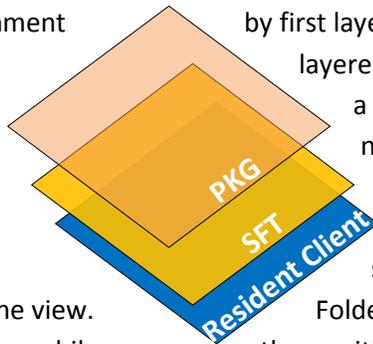


When layered, the resulting virtual environment view would appear as shown next:



Notice the difference in how the two resident client files, Name1b and Name2b, are treated in this view due to the transparency setting of the parent sub-folders.

When modifications are made to package components while the virtual application is running, these changes are generally saved in a separate PKG file¹. When the application package is next opened, the client creates the virtual environment and then the PKG contents are layered on top of this. In essence, all folders and keys of the PKG are set to a “merge” transparency setting, such that name/value pairs of the PKG are applied. As viewed from above, these layers act like panes of glass that contain smudges. Each file and registry name/value pair acts as an opaque smudge, preventing any visibility of sub-items in lower layers. But when DSC is considered two important, but unanswered (by Microsoft documentation), questions are raised that need to be answered:



by first layering the SFT over the resident client PC, layered on top of this. In essence, all folders a “merge” transparency setting, such that name/value pairs of the PKG are applied. layers act like panes of glass that registry name/value pair acts as an similarly named file or registry name in a Folders and keys marked with a transparency of “merge” allow those with a transparency of “merge”

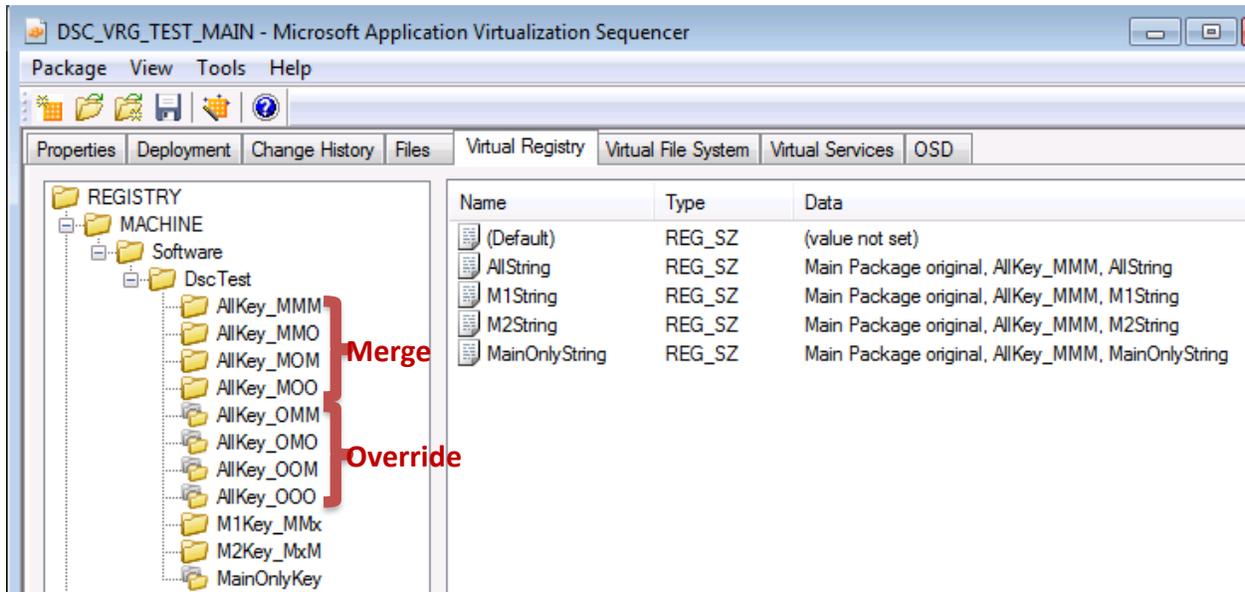
- How does the client perform layering when there are multiple SFTs and multiple PKGs?
- Once the virtual environment is created, to which PKG is a given change written to?

¹ Actually several PKG files exist for a virtual application. For simplicity in this paper, we will treat the collection of PKG files associated with an SFT as a single PKG. The research did look at these as separate layers and concluded that this simplification in our “description model”, does not misstate how the product works.

Testing Description

We devised a series of tests using a contrived set of packages to answer these two questions. The tests were conducted using DSC consisting of a main package sft and two “plugin” sfts.

Each of these packages contain a set of folders and files, registry keys and name/value pairs. Every possible combination of duplication and combination of transparency settings are created. Shown below is an example of the registry in the main application package.



The naming convention of these keys is as follows:

- “AllKey” indicates that the key exists in all three packages. “M1Key” indicates that the key exists in the main package and plugin1 but not plugin2. “M2” is main and plugin2. “MainOnlyKey” indicates that the key is not created in the plugins. “12key”, which does not exist in the main package would indicate the key exists in both plugin1 and plugin2.
- The last three characters are positional to indicate the transparency settings for the three packages. The first character for the main package, second for plugin1, etc. “MMM” means transparency is set to “merge” in each case. “OOM” means it is set to “Override” for the main application and plugin1 but “merge” in plugin2. The “x” character appears in a position when the key does not exist in that package.
- Under each of the keys, registry names appear using a similar naming for which packages have that name.

Folders and files are similarly set up for each of the packages. The contents of these files (as well as the contents of the registry strings) are unique for each package to aid in determining the source of the resulting view.

Finally, the test client is set up with a duplicate of all of these folders, files, registry keys, and name/value pairs. An additional file, registry name/value is also created labeled "ClientOnly" under each folder or key.

Each of the three packages are generated with an OSD application to the command prompt. From this command prompt, either regedit or notepad may be run inside the virtual environment to allow us to manipulate the visible

manipulate the contents of the visible files and registry items and create new ones.

The main package OSD is modified to add dependencies for both of the plug-ins, and all three packages are deployed to the client test user.

Client testing is performed as follows:

1. Running the main package with plugins, determine what is visible and the source of the file or registry item.
2. From that same view, modify each file and registry item that is visible. Also create a new file/item under each folder/key. Determine to which package PKG file the result is stored
3. After resetting all three packages (returning them to the initial state):
 - a. Running only plugin2, perform all modifications to visible items.
 - b. Running only plugin1, perform all modifications to visible items.
 - c. Running the main application with plugins, perform all modifications.
 - d. Determine to which PKG file the result is stored.

Determination of which PKG the changes are stored was performed by using the PkgView tool (a free tool that may be downloaded from the tools section of the TMurgent website).

Additional Testing Notes:

There are a few other details to note for completeness:

- All testing was performed using the 4.6 RTM release of App-V
- Files of a package stored in the "asset folder" (the folder created on the "Q:" drive) are not of interest to this test. Anything written to the asset folder tree can never have a duplicate in another package (because it must have a differently named asset folder) or on the client (because the drive is virtual). This is why the sequencer does not have transparency settings on the "Files" tab but only on the "Virtual Files System" tab.
- File/Folder, or Key/Item deletion was not tested. When running a virtual application, deletions become markings of a deletion in the appropriate PKG. We assume (but did not test) that these deletion markings act exactly as a modification would.
- In the case of files, we duplicated all files, making half as "User" and half as "Application" in the Files tab of the sequencer. This proved to be of no consequence.

- Environment variables, Policies, Scripts, are examples of possibly items that cannot conflict in the same way as folders, files, and registry items. These are contained in the OSD and are only applied when the first application of a package is started.

It is also important to note that our testing found differences in the implementation of the Virtual File System (VFS) and the Virtual Registry (VRG). It is because of these differences that we needed to disclose the complete testing results, rather than just a summary and recommendations. We attribute the differences due to different groups within the Microsoft App-V development team making independent decisions

independent implementation decisions. It is like building a house with two contractors, one that specializes in windows and another doors. If you ask them each to add a screen to their portals to keep out bugs, both will add effective screens. However, the window contractor will probably add screens that do not move (except to put up and take down) and the door contractor will probably add screens that open or slide. As you might guess, one of those lets bugs in now and then!

But this paper isn't about bugs, just differences that affect operation in different ways, leading us to need to understand this behavior and develop practices to keep the differences from causing problems.

VFS Test Results

The results of the VFS testing are best understood if you have some information regarding how folders and files properties are stored within the SFT and PKG. Unlike the VRG, all files and folders in the SFT have a unique GUID. When a file or folder change is marked in the PKG, the PKG does not store the path information, but only the name, GUID, and contents. The impact of this original design imposes a constraint that changes marked in a PKG must be relative to the SFT for which the change was made to. This is probably the behavior we all expected.

What we did not expect that was found in this testing, is how transparency is implemented, with different behavior on transparency settings in the main pkg versus those of plug-ins. In short, we saw more files than we expected to.

The VFS test results for Tests 1 and 2 are documented in the table that follows. The "Client" column indicates if the referenced files existed resident on the client. "Which visible" column indicates which sft the visible file came from (if visible). The "Modification to" column indicates which PKG the modification was written to. Notice that in these results we had both "User" and "Application" marked files causing the changes to be written to the User volume or Global volume, but in all cases to a PKG associated with the SFT that was "visible".

Folder	File	Client	Main	PlugIn1	PlugIn2	Which Visible	Modification To
DscTestMMM	ALL_VFS_Application	yes	Merge	Merge	Merge	Plugin2	P2 Global_21_x_500

Folder	File	Client	Main	PlugIn1	PlugIn2	Which Visible	Modification To
DscTestMMM	ALL_VFS_User	yes	Merge	Merge	Merge	Plugin2	P2 UserVol
DscTestMMM	M1_VFS_Application	yes	Merge	Merge		Plugin1	P1 Global_21_x_500
DscTestMMM	M1_VFS_User	yes	Merge	Merge		Plugin1	P1 UserVol
DscTestMMM	M2_VFS_Application	yes	Merge		Merge	Plugin2	P2 Global_21_x_500
DscTestMMM	M2_VFS_USER	yes	Merge		Merge	Plugin2	P2 UserVol
DscTestMMM	12_VFS_Application	yes		Merge	Merge	Plugin2	P2 Global_21_x_500
DscTestMMM	12_VFS_USER	yes		Merge	Merge	Plugin2	P2 UserVol
DscTestMMM	MainOnly_VFS_Application	yes	Merge			Main	Main Global_21_x_500
DscTestMMM	MainOnly_VFS_User	yes	Merge			Main	Main UserVol
DscTestMMM	1Only_VFS_Application	yes		Merge		Plugin1	P1 Global_21_x_500
DscTestMMM	1Only_VFS_USER	yes		Merge		Plugin1	P1 UserVol
DscTestMMM	2Only_VFS_Application	yes			Merge	Plugin2	P2 Global_21_x_500
DscTestMMM	2Only_VFS_USER	yes			Merge	Plugin2	P2 UserVol
DscTestMMM	ClientOnly_VFS	yes				Client	Client System
DscTestMMM	Runtime					once written	Client System
DscTestMMO	ALL_VFS_Application	yes	Merge	Merge	Override	Plugin2	P2 Global_21_x_500
DscTestMMO	ALL_VFS_User	yes	Merge	Merge	Override	Plugin2	P2 UserVol
DscTestMMO	M1_VFS_Application	yes	Merge	Merge		Plugin1	P1 Global_21_x_500
DscTestMMO	M1_VFS_User	yes	Merge	Merge		Plugin1	P1 UserVol
DscTestMMO	M2_VFS_Application	yes	Merge		Override	Plugin2	P2 Global_21_x_500
DscTestMMO	M2_VFS_USER	yes	Merge		Override	Plugin2	P2 UserVol
DscTestMMO	12_VFS_Application	yes		Merge	Override	Plugin2	P2 Global_21_x_500
DscTestMMO	12_VFS_USER	yes		Merge	Override	Plugin2	P2 UserVol
DscTestMMO	MainOnly_VFS_Application	yes	Merge				
DscTestMMO	MainOnly_VFS_User	yes	Merge				
DscTestMMO	1Only_VFS_Application	yes		Merge		Plugin1	P1 Global_21_x_500
DscTestMMO	1Only_VFS_USER	yes		Merge		Plugin1	P1 UserVol
DscTestMMO	2Only_VFS_Application	yes			Override	Plugin2	P2 Global_21_x_500
DscTestMMO	2Only_VFS_USER	yes			Override	Plugin2	P2 UserVol
DscTestMMO	ClientOnly_VFS	yes					
DscTestMMO	Runtime					once written	P2 UserVol
DscTestMOM	ALL_VFS_Application	yes	Merge	Override	Merge	Plugin2	P2 Global_21_x_500
DscTestMOM	ALL_VFS_User	yes	Merge	Override	Merge	Plugin2	P2 UserVol
DscTestMOM	M1_VFS_Application	yes	Merge	Override		Plugin1	P1 Global_21_x_500
DscTestMOM	M1_VFS_User	yes	Merge	Override		Plugin1	P1 UserVol
DscTestMOM	M2_VFS_Application	yes	Merge		Merge	Plugin2	P2 Global_21_x_500
DscTestMOM	M2_VFS_USER	yes	Merge		Merge	Plugin2	P2 UserVol
DscTestMOM	12_VFS_Application	yes		Override	Merge	Plugin2	P2 Global_21_x_500
DscTestMOM	12_VFS_USER	yes		Override	Merge	Plugin2	P2 UserVol

Folder	File	Client	Main	PlugIn1	PlugIn2	Which Visible	Modification To
DscTestMOM	MainOnly_VFS_Application	yes	Merge				
DscTestMOM	MainOnly_VFS_User	yes	Merge				
DscTestMOM	1Only_VFS_Application	yes		Override		Plugin1	P1 Global_21_x_500
DscTestMOM	1Only_VFS_USER	yes		Override		Plugin1	P1 UserVol
DscTestMOM	2Only_VFS_Application	yes			Merge	Plugin2	P2 Global_21_x_500
DscTestMOM	2Only_VFS_USER	yes			Merge	Plugin2	P2 UserVol
DscTestMOM	ClientOnly_VFS	yes					
DscTestMOM	Runtime					once written	P1 UserVol
DscTestMOO	ALL_VFS_Application	yes	Merge	Override	Override	Plugin2	P2 Global_21_x_500
DscTestMOO	ALL_VFS_User	yes	Merge	Override	Override	Plugin2	P2 UserVol
DscTestMOO	M1_VFS_Application	yes	Merge	Override		Plugin1	P1 Global_21_x_500
DscTestMOO	M1_VFS_User	yes	Merge	Override		Plugin1	P1 UserVol
DscTestMOO	M2_VFS_Application	yes	Merge		Override	Plugin2	P2 Global_21_x_500
DscTestMOO	M2_VFS_USER	yes	Merge		Override	Plugin2	P2 UserVol
DscTestMOO	12_VFS_Application	yes		Override	Override	Plugin2	P2 Global_21_x_500
DscTestMOO	12_VFS_USER	yes		Override	Override	Plugin2	P2 UserVol
DscTestMOO	MainOnly_VFS_Application	yes	Merge				
DscTestMOO	MainOnly_VFS_User	yes	Merge				
DscTestMOO	1Only_VFS_Application	yes		Override		Plugin1	P1 Global_21_x_500
DscTestMOO	1Only_VFS_USER	yes		Override		Plugin1	P1 UserVol
DscTestMOO	2Only_VFS_Application	yes			Override	Plugin2	P2 Global_21_x_500
DscTestMOO	2Only_VFS_USER	yes			Override	Plugin2	P2 UserVol
DscTestMOO	ClientOnly_VFS	yes					
DscTestMOO	Runtime					once written	P2 UserVol
DSCTestOMM	ALL_VFS_Application	yes	Override	Merge	Merge	Plugin2	P2 Global_21_x_500
DSCTestOMM	ALL_VFS_User	yes	Override	Merge	Merge	Plugin2	P2 UserVol
DSCTestOMM	M1_VFS_Application	yes	Override	Merge		Plugin1	P1 Global_21_x_500
DSCTestOMM	M1_VFS_User	yes	Override	Merge		Plugin1	P1 UserVol
DSCTestOMM	M2_VFS_Application	yes	Override		Merge	Plugin2	P2 Global_21_x_500
DSCTestOMM	M2_VFS_USER	yes	Override		Merge	Plugin2	P2 UserVol
DSCTestOMM	12_VFS_Application	yes		Merge	Merge	Plugin2	P2 Global_21_x_500
DSCTestOMM	12_VFS_USER	yes		Merge	Merge	Plugin2	P2 UserVol
DSCTestOMM	MainOnly_VFS_Application	yes	Override			Main	Main Global_21_x_500
DSCTestOMM	MainOnly_VFS_User	yes	Override			Main	Main UserVol
DSCTestOMM	1Only_VFS_Application	yes		Merge		Plugin1	P1 Global_21_x_500
DSCTestOMM	1Only_VFS_USER	yes		Merge		Plugin1	P1 UserVol
DSCTestOMM	2Only_VFS_Application	yes			Merge	Plugin2	P2 Global_21_x_500
DSCTestOMM	2Only_VFS_USER	yes			Merge	Plugin2	P2 UserVol
DSCTestOMM	ClientOnly_VFS	yes					

Folder	File	Client	Main	PlugIn1	PlugIn2	Which Visible	Modification To
DSCTestOMM	Runtime					once written	Main UserVol
DSCTestOMO	ALL_VFS_Application	yes	Override	Merge	Override	Plugin2	P2 Global_21_x_500
DSCTestOMO	ALL_VFS_User	yes	Override	Merge	Override	Plugin2	P2 UserVol
DSCTestOMO	M1_VFS_Application	yes	Override	Merge		Plugin1	P1 Global_21_x_500
DSCTestOMO	M1_VFS_User	yes	Override	Merge		Plugin1	P1 UserVol
DSCTestOMO	M2_VFS_Application	yes	Override		Override	Plugin2	P2 Global_21_x_500
DSCTestOMO	M2_VFS_USER	yes	Override		Override	Plugin2	P2 UserVol
DSCTestOMO	12_VFS_Application	yes		Merge	Override	Plugin2	P2 Global_21_x_500
DSCTestOMO	12_VFS_USER	yes		Merge	Override	Plugin2	P2 UserVol
DSCTestOMO	MainOnly_VFS_Application	yes	Override				
DSCTestOMO	MainOnly_VFS_User	yes	Override				
DSCTestOMO	1Only_VFS_Application	yes		Merge		Plugin1	P1 Global_21_x_500
DSCTestOMO	1Only_VFS_USER	yes		Merge		Plugin1	P1 UserVol
DSCTestOMO	2Only_VFS_Application	yes			Override	Plugin2	P2 Global_21_x_500
DSCTestOMO	2Only_VFS_USER	yes			Override	Plugin2	P2 UserVol
DSCTestOMO	ClientOnly_VFS	yes					
DSCTestOMO	Runtime					once written	P2 UserVol
DSCTestOOM	ALL_VFS_Application	yes	Override	Override	Merge	Plugin2	P2 Global_21_x_500
DSCTestOOM	ALL_VFS_User	yes	Override	Override	Merge	Plugin2	P2 UserVol
DSCTestOOM	M1_VFS_Application	yes	Override	Override		Plugin1	P1 Global_21_x_500
DSCTestOOM	M1_VFS_User	yes	Override	Override		Plugin1	P1 UserVol
DSCTestOOM	M2_VFS_Application	yes	Override		Merge	Plugin2	P2 Global_21_x_500
DSCTestOOM	M2_VFS_USER	yes	Override		Merge	Plugin2	P2 UserVol
DSCTestOOM	12_VFS_Application	yes		Override	Merge	Plugin2	P2 Global_21_x_500
DSCTestOOM	12_VFS_USER	yes		Override	Merge	Plugin2	P2 UserVol
DSCTestOOM	MainOnly_VFS_Application	yes	Override				
DSCTestOOM	MainOnly_VFS_User	yes	Override				
DSCTestOOM	1Only_VFS_Application	yes		Override		Plugin1	P1 Global_21_x_500
DSCTestOOM	1Only_VFS_USER	yes		Override		Plugin1	P1 UserVol
DSCTestOOM	2Only_VFS_Application	yes			Merge	Plugin2	P2 Global_21_x_500
DSCTestOOM	2Only_VFS_USER	yes			Merge	Plugin2	P2 UserVol
DSCTestOOM	ClientOnly_VFS	yes					
DSCTestOOM	Runtime					once written	P1 UserVol
DSCTestOOO	ALL_VFS_Application	yes	Override	Override	Override	Plugin2	P2 Global_21_x_500
DSCTestOOO	ALL_VFS_User	yes	Override	Override	Override	Plugin2	P2 UserVol
DSCTestOOO	M1_VFS_Application	yes	Override	Override		Plugin1	P1 Global_21_x_500
DSCTestOOO	M1_VFS_User	yes	Override	Override		Plugin1	P1 UserVol
DSCTestOOO	M2_VFS_Application	yes	Override		Override	Plugin2	P2 Global_21_x_500
DSCTestOOO	M2_VFS_USER	yes	Override		Override	Plugin2	P2 UserVol

Folder	File	Client	Main	PlugIn1	PlugIn2	Which Visible	Modification To
DSCTest000	12_VFS_Application	yes		Override	Override	Plugin2	P2 Global_21_x_500
DSCTest000	12_VFS_USER	yes		Override	Override	Plugin2	P2 UserVol
DSCTest000	MainOnly_VFS_Application	yes	Override				
DSCTest000	MainOnly_VFS_User	yes	Override				
DSCTest000	1Only_VFS_Application	yes		Override		Plugin1	P1 Global_21_x_500
DSCTest000	1Only_VFS_USER	yes		Override		Plugin1	P1 UserVol
DSCTest000	2Only_VFS_Application	yes			Override	Plugin2	P2 Global_21_x_500
DSCTest000	2Only_VFS_USER	yes			Override	Plugin2	P2 UserVol
DSCTest000	ClientOnly_VFS	yes					
DSCTest000	Runtime					once written	P2 UserVol
DscTest-ClientOnly	ClientOnly_VFS	yes				Client	Client System
DscTest-ClientOnly	Runtime					once written	Client System

Looking at these results for visibility, we can clearly see that when the main application has a folder set for "Override", no client files in that folder may be seen. Similarly if **either** of the plugins have the folder set for Override, the a file in that folder from either the client or main sft cannot be seen. *But surprisingly, an override setting to the folder on plugin2 does not hide a plugin1 file!* Clearly the software performing layering of plugins works differently than the layering of the main sft.

Performing Test 3 provided no unexpected insights, but confirmed our guess as the ordering of the three SFTs and three PKGs. In essence, each SFT has it's PKG layered just above it, and before the next package layer.

VRG Test Results

Unlike the virtual file system, the VRG does not use GUIDs for items. Instead, each item is stored (both in the SFT and the PKG) with a full “path” reference. Although this does should not prevent an implementation design resulting in parallel results to those seen for the VFS, we found strikingly different results.

Key	RegName	Client	Main	PlugIn1	PlugIn2	Which Visible	Modification To
DscTestMMM	ALL_String	Y	Merge	Merge	Merge	2	m
DscTestMMM	M1_String	Y	Merge	Merge		1	m
DscTestMMM	M2_String	Y	Merge		Merge	2	m
DscTestMMM	12_String	Y		Merge	Merge	2	m
DscTestMMM	MainOnly_String	Y	Merge			m	m
DscTestMMM	1Only_String	Y		Merge		1	m
DscTestMMM	2Only_String	Y			Merge	2	m
DscTestMMM	Runtime					once written	m
DscTestMMO	ALL_String	Y	Merge	Merge	Override	2	m
DscTestMMO	M1_String	Y	Merge	Merge			m
DscTestMMO	M2_String	Y	Merge		Override	2	m
DscTestMMO	12_String	Y		Merge	Override	2	m
DscTestMMO	MainOnly_String	Y	Merge				m
DscTestMMO	1Only_String	Y		Merge			
DscTestMMO	2Only_String	Y			Override	2	m
DscTestMMO	Runtime					once written	m
DscTestMOM	ALL_String	Y	Merge	Override	Merge	2	m
DscTestMOM	M1_String	Y	Merge	Override		1	m
DscTestMOM	M2_String	Y	Merge		Merge	2	m
DscTestMOM	12_String	Y		Override	Merge	2	m
DscTestMOM	MainOnly_String	Y	Merge			m	m
DscTestMOM	1Only_String	Y		Override		1	m
DscTestMOM	2Only_String	Y			Merge	2	m
DscTestMOM	Runtime					once written	m
DscTestMOO	ALL_String	Y	Merge	Override	Override	2	m
DscTestMOO	M1_String	Y	Merge	Override			m
DscTestMOO	M2_String	Y	Merge		Override	2	m
DscTestMOO	12_String	Y		Override	Override	2	m
DscTestMOO	MainOnly_String	Y	Merge				m
DscTestMOO	1Only_String	Y		Override			
DscTestMOO	2Only_String	Y			Override	2	m
DscTestMOO	Runtime					once written	m
DSCTestOMM	ALL_String	Y	Override	Merge	Merge	2	m

Key	RegName	Client	Main	PlugIn1	PlugIn2	Which Visible	Modification To
DSCTestOMM	M1_String	Y	Override	Merge		1	m
DSCTestOMM	M2_String	Y	Override		Merge	2	m
DSCTestOMM	12_String	Y		Merge	Merge	2	m
DSCTestOMM	MainOnly_String	Y	Override			m	m
DSCTestOMM	1Only_String	Y		Merge		1	m
DSCTestOMM	2Only_String	Y			Merge	2	m
DSCTestOMM	Runtime					once written	m
DSCTestOMO	ALL_String	Y	Override	Merge	Override	2	m
DSCTestOMO	M1_String	Y	Override	Merge			m
DSCTestOMO	M2_String	Y	Override		Override	2	m
DSCTestOMO	12_String	Y		Merge	Override	2	m
DSCTestOMO	MainOnly_String	Y	Override				m
DSCTestOMO	1Only_String	Y		Merge			
DSCTestOMO	2Only_String	Y			Override	2	m
DSCTestOMO	Runtime					once written	m
DSCTestOOM	ALL_String	Y	Override	Override	Merge	2	m
DSCTestOOM	M1_String	Y	Override	Override		1	m
DSCTestOOM	M2_String	Y	Override		Merge	2	m
DSCTestOOM	12_String	Y		Override	Merge	2	m
DSCTestOOM	MainOnly_String	Y	Override			m	m
DSCTestOOM	1Only_String	Y		Override		1	m
DSCTestOOM	2Only_String	Y			Merge	2	m
DSCTestOOM	Runtime					once written	m
DSCTestOOO	ALL_String	Y	Override	Override	Override	2	m
DSCTestOOO	M1_String	Y	Override	Override			m
DSCTestOOO	M2_String	Y	Override		Override	2	m
DSCTestOOO	12_String	Y		Override	Override	2	m
DSCTestOOO	MainOnly_String	Y	Override				m
DSCTestOOO	1Only_String	Y		Override			
DSCTestOOO	2Only_String	Y			Override	2	m
DSCTestOOO	Runtime					once written	m
DSCTestMMx	M1_String	Y	Merge	Merge		1	m
DSCTestMMx	MainOnly_String	Y	Merge			m	m
DSCTestMMx	1Only_String	Y		Merge		1	m
DSCTestMMx	Runtime	Y				once written	m
DSCTestMxM	M2_String	Y	Merge		Merge	2	m
DSCTestMxM	MainOnly_String	Y	Merge			m	m
DSCTestMxM	2Only_String	Y			Merge	2	m
DSCTestMxM	Runtime					once written	m

Key	RegName	Client	Main	PlugIn1	PlugIn2	Which Visible	Modification To
DSCTestMxx	MainOnly_String	Y	Merge			m	m
DSCTestMxx	Runtime	Y				once written	m
DSCTestxMx	1Only_String	Y		Merge			m
DSCTestxMx	Runtime	Y				once written	m
DSCTestxxM	2Only_String	Y			Merge		m
DSCTestxxM	Runtime					once written	m

Looking at the visibility results of test 1, we see that each layer, including the individual plugins, are applied separately and with respect to their individual transparency setting. (Compare the MMO result for "M1" name of this table to that of the MMO/M1 file of the VFS results).

In test 2, we see that all changes are written to the PKG associated with the main SFT, no matter what the source.

Furthermore, in test 3 we see that even if additional registry settings exist for plugin associated PKGs, these are ignored.

Summary of Test Results

Consider the differences in behavior of the VFS and VGR.

Test 1, where we looked at visibility of SFT contents:

DSC VFS Transparency Visibility Summary

“Highest level” wins, except...

 Plug 2	Plugin overrides only hide main/client, not each other!
 Plug 1	
 Main	Any “plugin” override hides
 Client	Any “higher” override hides.

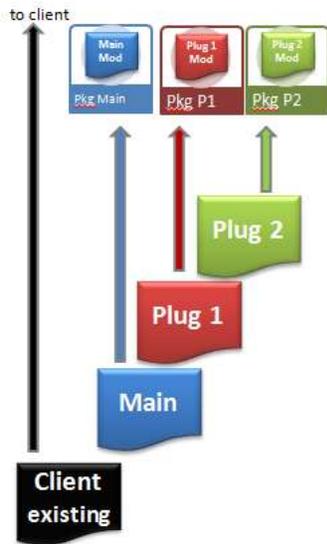
DSC VRG Transparency Visibility Summary

“Highest level” wins, except...

 Plug 2	No special plugin rule (unlike VFS)
 Plug 1	Any “higher plugin” override hides
 Main	Any “plugin” override hides
 Client	Any “higher” override hides.

Test 2, were we looked at where changes are saved:

DSC VFS Transparency Modified to PKG Summary

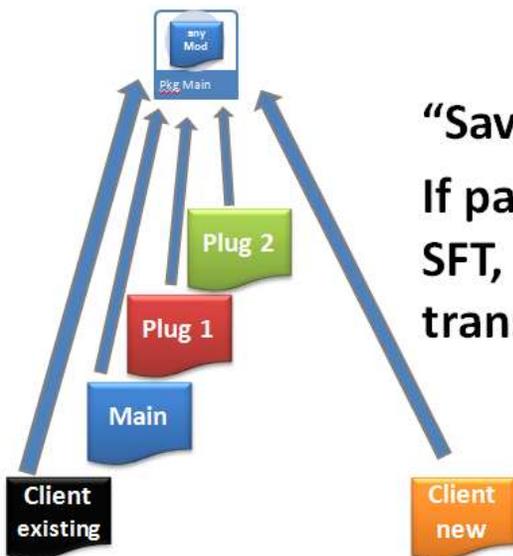


“Save mod to PKG associated with SFT it came from”.

To “highest” override folder layer,
Or client if none.

Client new ...whether or not client folder exists

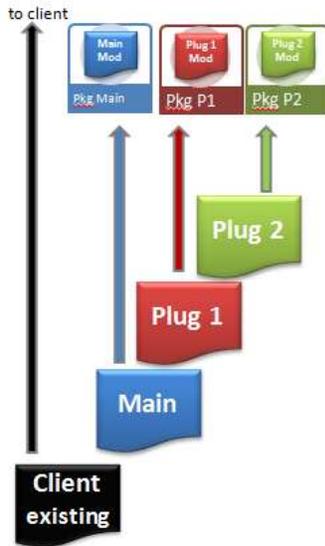
DSC VRG Transparency Modified to PKG Summary



**“Save mod to MAIN PKG
If parent key exists in any
SFT, regardless of
transparency”.**

Test 3, where we look at the effect of each SFT having an SFT:

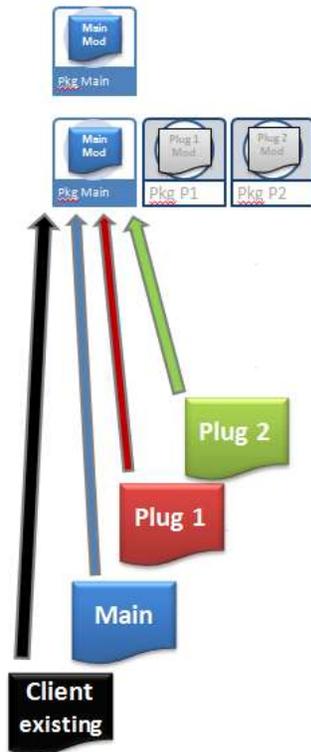
DSC VFS Transparency Modified to PKG Summary



“Save mod to PKG associated with SFT it came from”.

To “highest” override folder layer,
Or client if none.

Client new ...whether or not client folder exists



DSC VRG Transparency Multiple Pkg Summary

“PKG layers other than associated with main SFT are ignored”.

The differences highlighted in these three tests have implications to using DSC.

The first test highlight the need to ensure that transparency settings for duplicated folders are consistent between SFTs.

For applications that have simple integrations, we believe that the differences shown for tests 2 and 3 are not of consequence. In practice, these plugins (or referenced applications) do not generally duplicate file or registry names, but augment each other. These differences, however, may cause more complicated integrations to have inconsistent runtime behavior if the user ever runs a plugin without the other applications. If the user runs the plugin separately and makes both file and registry changes, and later runs the main application with the plugin, only the file changes would be seen. This could lead to erratic, unpredictable, and seemingly random application behavior.

Recommendations

- Due to differences in how VRG/VFS behave in saving modifications, DFS involving significant Registry Key AND File Folder overlaps should always be run with same ordering. This may mean that “plugin” OSDs need to be significantly modified to have the “IMPLEMENTATION” block copied from the modified main application OSD, with the FILENAME changed to that of the original plugin IMPLEMENTATION.
- If DFS “has issues”, check to ensure transparency settings are consistent for any given folder.
- Be aware that a VFS override in second or subsequent plugin for folder to hide an earlier plugin is abnormal behavior.