Understanding DISM / Servicing Stack Interaction

bsodtutorials.wordpress.com/2022/07/26/understanding-dism-servicing-stack-interaction

View all posts by 0x14c → July 26, 2022

Understanding DISM – /CheckHealth, /ScanHealth and /RestoreHealth

There appears to be much confusion about what the switches available for DISM actually do. I've seen countless occasions where people have recommended forum users who are suffering from Windows Update issues (and even BSODs) to run DISM /ScanHealth followed by /RestoreHealth. Please stop! It's pointless.

DISM is designed for servicing the Windows operating system and provides an option to ensure that the Component Store is not corrupt. There is two main options for doing this: /ScanHealth and /RestoreHealth. They actually almost do the same thing, however, the latter option will actually attempt to perform repairs using the Windows Update servers as the primary source. Although, typically, this is only for corrupted packages and payload files – I'm yet to see a case whereby DISM was able to repair registry corruption, although, it usually does an excellent job of finding it.

The /CheckHealth is the fastest option and simply checks for the presence of two different store flags which are part of the CBS subkey. These are the **Corruption** value and **Unserviceable** value. These two flags are queried by a CBS worker process called TiWorker.exe.

/ScanHealth and /RestoreHealth do largely the same as each other and perform an extensive check of the Component Store. If there is no corruption then the two previously mentioned flags are cleared, otherwise they're set to 0x1 or true. DISM does a good job of checking for registry corruption within the COMPONENTS and CBS subkey, however, tends to struggle when it comes to checking actual files. With this in mind, it is often best to run SFC as well which will only check for file corruption within the WinSxS folder. SFC does attempt to perform repairs if possible by checking the Backup directory of the WinSxS folder for a suitable replacement file, if none can be found, then the file is reported as corrupted or missing from the backup folder.

DISM and the Servicing Stack:

Now, there is an important registry key which is queried just before DISM is executed and that is the Image File Execution Options (IFEO) subkey. There is a particular value named **Debugger** and if this is set to 0x1, the DISM will not start at all and will throw DISM Error 2.

```
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution
Options\DismHost.exe" /v Debugger
```

Additionally, prior to the TrustedInstaller (Windows Module Installer) service starting, the same subkey is queried but for the TrustedInstaller.exe instead.

Eventually, DismHost.exe looks up a few different subkeys under HKEY_CLASSES_ROOT, there is quite a few important subkeys here and if they're missing or corrupted, then DISM will fail to run correctly.

```
HKEY_CLASSES_ROOT\Interface\{00020400-0000-0000-C000-000000000046}
    (Default)    REG_SZ    IDispatch

HKEY_CLASSES_ROOT\Interface\{00020400-0000-0000-C000-000000000046}\ProxyStubClsid32
    (Default)    REG_SZ    {00020420-0000-0000-C000-000000000046}
```

The default value is checked here and then used in conjunction with another subkey of the same name.

```
HKEY_CLASSES_ROOT\CLSID\{00020420-0000-0000-C000-000000000046}
    (Default)    REG_SZ    PSDispatch

HKEY_CLASSES_ROOT\CLSID\{00020420-0000-0000-C000-000000000046}\InprocServer32
    (Default)    REG_SZ    C:\Windows\System32\oleaut32.dll
    ThreadingModel    REG_SZ    Both
```

The above two subkeys aren't too interesting and quite standard. However, eventually DismHost.exe will begin to start interacting with the servicing stack – this is commonly where issues begin to arise. The servicing directory is queried to check which version of the servicing stack is installed and active. The folder in question in the following:

```
%systemroot%\servicing\version\{SSU version}\*_installed
```

The folder will always contain two files: amd64_installed and x86_installed. These can be found in the corresponding SSU component folders within the WinSxS folder. DISM uses these files to look up the appropriate servicing stack component to run.

On my server instance, it was looking up and creating a file handle to the wcp.dll file which is part of the following component:

```
%systemroot%\WinSxS\amd64_microsoft-windows-
servicingstack_31bf3856ad364e35_6.3.9600.17031_none_fa50b3979b1bcb4a\wcp.dll
```

The file attributes were examined and the file handle closed shortly afterwards; note, if this folder is corrupt or missing in any way, then DISM will crash and refuse to run. The DISM log will contain the following error message:

> 2022-07-21 15:07:10, Warning DISM DISM OS Provider: PID=6236 TID=6932 Failed to bind the online servicing stack – CDISMOSServiceManager::get_ServicingStackDirectory(hr:0x80070003)
>
> 2022-07-21 15:07:10, Error DISM DISM OS Provider: PID=6236 TID=6932 Unable to retrieve servicing stack folder for DLL search path modification. – CDISMOSServiceManager::SetDllSearchPath(hr:0x80070003)
>
> 2022-07-21 15:07:10, Error DISM DISM OS Provider: PID=6236 TID=6932 Unable to set the DLL search path to the servicing stack folder. C:\Windows may not point to a valid Windows folder. – CDISMOSServiceManager::SetWindowsDirectory(hr:0x80070003)
>
> 2022-07-21 15:07:10, Error DISM DISM.EXE: Failed to set the windows directory to 'C:\Windows'. HRESULT=80070003

The active search path will be the same value as the one under Version subkey of the CBS key. This will be mentioned further later.

DISM then eventually returns to the WinSxS folder mentioned previously and then opens the CbsCore.dll.

```
%systemroot%\WinSxS\amd64_microsoft-windows-
servicingstack_31bf3856ad364e35_6.3.9600.17031_none_fa50b3979b1bcb4a\CbsCore.dll
```

The following value is then queried:

```
reg query "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CMF\Config" /v SYSTEM

HKEY_CLASSES_ROOT\CLSID\{75207391-23F2-4396-85F0-8FDB879ED0ED}
    (Default)    REG_SZ    Component Based Servicing Session Proxy/Stub

HKEY_CLASSES_ROOT\CLSID\{75207391-23F2-4396-85F0-8FDB879ED0ED}\InProcServer32
    (Default)    REG_EXPAND_SZ    %SystemRoot%\servicing\CbsApi.dll
    ThreadingModel    REG_SZ    Both
```

The above two subkeys are then examined which seem to be used to set up a CBS session. The second subkey points to the location of the CbsApi.dll.

```
HKEY_CLASSES_ROOT\CLSID\{752073A1-23F2-4396-85F0-8FDB879ED0ED}
    (Default)    REG_SZ    Component Based Servicing Session
    AppID    REG_SZ    {752073A2-23F2-4396-85F0-8FDB879ED0ED}
```

The **AppId** value is then examined which is used to look up the value data which corresponds to the following subkey name:

```
HKEY_CLASSES_ROOT\AppID\{752073A2-23F2-4396-85F0-8FDB879ED0ED}
    (Default)    REG_SZ    Trusted Installer Service
    AccessPermission    REG_BINARY
0100048084000000940000000000000014000000020070000500000000001400070000000101000000000

    LaunchPermission    REG_BINARY
0100048070000000 8C000000000000001400000002005C000400000000001400 0B000000101000000000

    LocalService    REG_SZ    TrustedInstaller
```

As the **LocalService** value suggests, this subkey is related to the Trusted Installer Service which is the principal owner of almost all the files and registry keys required for servicing the operating system.

The services.exe then checks the services configuration for the Trusted Installer service within the following registry subkey:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\TrustedInstaller
    BlockTime    REG_DWORD    0x2a30
    BlockTimeIncrement    REG_DWORD    0x384
    Description    REG_SZ    @%SystemRoot%\servicing\TrustedInstaller.exe,-101
    DisplayName    REG_SZ    @%SystemRoot%\servicing\TrustedInstaller.exe,-100
    ErrorControl    REG_DWORD    0x1
    FailureActions    REG_BINARY
840300000000000000000000030000001400000001000000C0D4010001000000E0930400000000000000000

    Group    REG_SZ    ProfSvc_Group
    ImagePath    REG_EXPAND_SZ    %SystemRoot%\servicing\TrustedInstaller.exe
    ObjectName    REG_SZ    localSystem
    PreshutdownTimeout    REG_DWORD    0x36ee80
    ServiceSidType    REG_DWORD    0x1
    Start    REG_DWORD    0x3
    Type    REG_DWORD    0x10
```

The TrustedInstaller service is then launched from the same directory as the **ImagePath** value. The service then enumerates the values under the **Version** subkey:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\Version
```

The TrustedInstaller eventually checks the CBS log configuration by examining the following four values:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\EnableLog
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SideBySide\Configuration\CBSLogMaxInMB
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SideBySide\Configuration\CBSLogHardMaxI

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SideBySide\Configuration\NumCBSPersistL
```

The following key is then created to indicate that the Trusted Installer service is actively running:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\TiRunning
```

To ensure there is no pending reboots required or that one is actively taking place, the service will query the following two values:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based
Servicing\RebootInProgress
HKLM\Software\Microsoft\Windows\CurrentVersion\Component Based
Servicing\RebootPending
```

The TrustedInstaller then examines the default values of the following subkeys:

```
HKEY_CLASSES_ROOT\CLSID\{8F5DF053-3013-4DD8-B5F4-88214E81C0CF}
    (Default)    REG_SZ    SFP Repair Class
    AppID    REG_SZ    {752073A2-23F2-4396-85F0-8FDB879ED0ED}

HKEY_CLASSES_ROOT\CLSID\{3C6859CE-230B-48A4-BE6C-932C0C202048}
    (Default)    REG_SZ    Sxs Store Class
    AppID    REG_SZ    {752073A2-23F2-4396-85F0-8FDB879ED0ED}

HKEY_CLASSES_ROOT\CLSID\{F556F9B2-C810-44A2-BA7A-3AB8C24E666D}
    (Default)    REG_SZ    GenValObject Outer Class
    AppID    REG_SZ    {D8D4249F-A8FB-44A7-8AA0-564E8C385BD6}
```

The **AppId** value for the final subkey corresponds to the software protection platform:

```
HKEY_CLASSES_ROOT\AppID\{D8D4249F-A8FB-44A7-8AA0-564E8C385BD6}
    (Default)    REG_SZ    Microsoft Software Protection Platform Admin Object
(outer)
    AccessPermission    REG_BINARY
0100148088000000980000001400000030000000002001C000100000011001400010000000101000000000C

    LaunchPermission    REG_BINARY
0100148088000000980000001400000030000000002001C000100000011001400010000000101000000000C

    LocalService    REG_SZ    TrustedInstaller
```

Another CBS session interface related subkey is then examined:

```
HKEY_CLASSES_ROOT\Interface\{F568C899-AF4F-4EAA-B12A-B8E5F1B219DE}
    (Default)    REG_SZ    ICbsSession8

HKEY_CLASSES_ROOT\Interface\{F568C899-AF4F-4EAA-B12A-B8E5F1B219DE}\ProxyStubClsid32
    (Default)    REG_SZ    {75207391-23F2-4396-85F0-8FDB879ED0ED}
```

The **InProcServer32** value is then looked up from the same subkey mentioned earlier, this is the same one as the default value for **ProxyStubClsid32** subkey. The CbsApi.dll is then has a file handle opened to by the Trusted Installer service:

```
%systemroot%\servicing\CbsApi.dll
```

The image is then loaded by the Trusted Installer, DismHost.exe then queries our CBS session stub subkey from earlier a few times. After some time of DismHost.exe repeatedly examining a CLSID subkey which doesn't exist; we eventually get to a very important subkey which will throw DISM Error 87 if it doesn't exist or is corrupt:

```
HKEY_CLASSES_ROOT\CLSID\{0823B6F8-F499-4D5E-B885-EA9CB4F43B24}
    (Default)    REG_SZ    Component Based Servicing Worker
    AppID    REG_SZ    {8D15A4F3-1BE5-4120-8A4D-2EF92A5DD58D}

HKEY_CLASSES_ROOT\CLSID\{0823B6F8-F499-4D5E-B885-EA9CB4F43B24}\LocalServer32
    (Default)    REG_EXPAND_SZ    %SystemRoot%\winsxs\amd64_microsoft-windows-
servicingstack_31bf3856ad364e35_6.3.9600.17031_none_fa50b3979b1bcb4a\TiWorker.exe
```

This subkey is used to store the location of the TiWorker process which is a child process of the Trusted Installer service. The TiWorker process will create a number of threads which are sometimes referred to in the CBS log as a CBS worker. If this subkey is missing or corrupted then you will experience DISM Error 87 or an error message stating that the /Cleanup-Image switch is not recognised. Additionally, if you disable the Trusted Installer service, the same error code will be produced. Thanks to Maxstar for discovering the cause of this error.

```
%systemroot%\WinSxS\amd64_microsoft-windows-
servicingstack_31bf3856ad364e35_6.3.9600.17031_none_fa50b3979b1bcb4a\TiWorker.exe
```

The TiWorker.exe is then launched and the subsequent subkey is then examined:

```
HKEY_CLASSES_ROOT\AppID\{8D15A4F3-1BE5-4120-8A4D-2EF92A5DD58D}
    (Default)    REG_SZ    TiWorker
    AccessPermission    REG_BINARY
0100048074000000840000000000000014000000020060000400000000000140007000000010100000000000

    LaunchPermission    REG_BINARY
0100048070000000800000000000000140000000200500004000000000000140008000000010100000000000
```

The **AppId** corresponds to the same **AppId** value as the Component Based Servicing Worker CLSID subeky. The TiWorker thread then opens a file handle to the following file and reads it.

```
%systemroot%\WinSxS\amd64_microsoft-windows-
servicingstack_31bf3856ad364e35_6.3.9600.17031_none_fa50b3979b1bcb4a\wdscore.dll
```

The TiWorker.exe eventually loads the CbsApi.dll from the servicing folder and the Trusted Installer service once again examines two other subkeys:

```
HKEY_CLASSES_ROOT\Interface\{A70DBECC-3734-4B22-B2D1-648C0E43E177}
    (Default)    REG_SZ    ICbsWorker

HKEY_CLASSES_ROOT\Interface\{A70DBECC-3734-4B22-B2D1-648C0E43E177}\ProxyStubClsid32
    (Default)    REG_SZ    {75207391-23F2-4396-85F0-8FDB879ED0ED}

HKEY_CLASSES_ROOT\Interface\{365DE52E-EE7E-4975-AEC8-06588234BB3C}
    (Default)    REG_SZ    ITrustedInstallerService

HKEY_CLASSES_ROOT\Interface\{365DE52E-EE7E-4975-AEC8-06588234BB3C}\ProxyStubClsid32
    (Default)    REG_SZ    {75207391-23F2-4396-85F0-8FDB879ED0ED}
```

This leads to the TiWorker thread loads a few other images from the servicing component within the WinSxS folder.

```
%systemroot%\WinSxS\amd64_microsoft-windows-
servicingstack_31bf3856ad364e35_6.3.9600.17031_none_fa50b3979b1bcb4a\CbsCore.dll
%systemroot%\WinSxS\amd64_microsoft-windows-
servicingstack_31bf3856ad364e35_6.3.9600.17031_none_fa50b3979b1bcb4a\USERENV.dll
%systemroot%\WinSxS\amd64_microsoft-windows-
servicingstack_31bf3856ad364e35_6.3.9600.17031_none_fa50b3979b1bcb4a\profapi.dll
```

The TiWorker ensures that the following subkeys don't exist:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SideBySide\Configuration\DisablePSRL
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SideBySide\DisableKernelTransactions
```

A few other images are then loaded:

```
%systemroot%\WinSxS\amd64_microsoft-windows-
servicingstack_31bf3856ad364e35_6.3.9600.17031_none_fa50b3979b1bcb4a\dpx.dll
%systemroot%\WinSxS\amd64_microsoft-windows-
servicingstack_31bf3856ad364e35_6.3.9600.17031_none_fa50b3979b1bcb4a\wcp.dll
%systemroot%\WinSxS\amd64_microsoft-windows-
servicingstack_31bf3856ad364e35_6.3.9600.17031_none_fa50b3979b1bcb4a\DrUpdate.dll
```

The service host (svchost.exe) then queries the following subkey:

```
HKEY_CLASSES_ROOT\CLSID\{D5978620-5B9F-11D1-8DD2-00AA004ABD5E}
    (Default)    REG_SZ    SENS Network Events

HKEY_CLASSES_ROOT\CLSID\{D5978620-5B9F-11D1-8DD2-00AA004ABD5E}\InprocServer32
    (Default)    REG_EXPAND_SZ    %systemroot%\system32\ES.DLL
    ThreadingModel    REG_SZ    Both
```

This causes the TiWorker to look for the following file which never appears to exist, seems to be related to the Software Quality Management component:

```
%systemroot%\servicing\SQM\*_all.sqm
```

The TiWorker then sets the session times for the latest CBS session and then begins to either check for the corruption flags (/CheckHealth) or check the Component Store for corruption (/ScanHealth).

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based
Servicing\SessionIdHigh
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\SessionIdLow
```

References:

https://docs.microsoft.com/en-us/answers/questions/179052/the-source-files-not-found-running-dism-restorehea.html

Process Monitor Trace from Windows Server 2012 R2