Get MSSQL serial

```
Function Get-SqlServerKeys {
    <#
     .SYNOPSIS
        Gets SQL Server Product Keys from local and remote SQL Servers.
Works with SQL Server 2005-2014

     .DESCRIPTION
        Using a string of servers, a text file, or Central Management Server
to provide a list of servers, this script will go to each server and get the
product key for all installed instances. Clustered instances are supported
as well. Requires regular user access to the SQL instances, SMO installed
locally, Remote Registry enabled and acessible by the account running the
script.

        Uses key decoder by Jakob Bindslet (http://goo.gl/1jiwcB)

     .PARAMETER Servers
        A comma separated list of servers. This can be the NetBIOS name, IP,
or SQL instance name

     .PARAMETER CentralMgmtServer
        Compiles list of servers to inventory using all servers stored
within a Central Management Server. Requires having SQL Management Studio
installed.

     .PARAMETER ServersFromFile
        Uses a text file as input. The file must be formatted as such:
        sqlserver1
        sqlserver2

     .NOTES
        Author  : Chrissy LeMaire
        Requires:     PowerShell Version 3.0, SQL Server SMO,  Remote
Registry
        Version: 0.8.3
        DateUpdated: 2015-June-2

     .LINK
https://gallery.technet.microsoft.com/scriptcenter/Get-SQL-Server-Product-4b
5bf4f8

     .EXAMPLE
     Get-SqlServerKeys winxp, sqlservera, sqlserver2014a, win2k8
        Gets SQL Server versions, editions and product keys for all
instances within each server or workstation.

     .EXAMPLE
```

```
     Get-SqlServerKeys -CentralMgmtServer sqlserver01
           Gets SQL Server versions, editions and product keys for all
instances within sqlserver01's Central Management Server

     .EXAMPLE
     Get-SqlServerKeys -ServersFromFile C:\Scripts\servers.txt
      Gets SQL Server versions, editions and product keys for all instances
listed within C:\Scripts\servers.txt
    #>
    #Requires -Version 3.0
    [CmdletBinding(DefaultParameterSetName="Default")]

    Param(
        [parameter(Position=0)]
        [string[]]$Servers,
        # Central Management Server
        [string]$CentralMgmtServer,
        # File with one server per line
        [string]$ServersFromFile
        )

    BEGIN {

        Function Unlock-SQLServerKey {
            param(
                [Parameter(Mandatory = $true)]
                [byte[]]$data,
                [int]$version
            )
            try {
                if ($version -ge 11) { $binArray = ($data)[0..66] } else {
$binArray = ($data)[52..66] }
                $charsArray =
"B","C","D","F","G","H","J","K","M","P","Q","R","T","V","W","X","Y","2","3",
"4","6","7","8","9"
                for ($i = 24; $i -ge 0; $i--) {
                    $k = 0
                    for ($j = 14; $j -ge 0; $j--) {
                        $k = $k * 256 -bxor $binArray[$j]
                        $binArray[$j] = [math]::truncate($k / 24)
                        $k = $k % 24
                    }
                    $productKey = $charsArray[$k] + $productKey
                    if (($i % 5 -eq 0) -and ($i -ne 0)) {
                        $productKey = "-" + $productKey
                    }
                }
            }
            catch { $productkey = "Cannot decode product key." }
            return $productKey
        }
```

```
    }

    PROCESS {
        if ((Get-Host).Version.Major -lt 3) { throw "PowerShell 3.0 and
above required." }

        if
([Reflection.Assembly]::LoadWithPartialName("Microsoft.SqlServer.SMO") -eq
$null )
        { throw "Quitting: SMO Required. You can download it from
http://goo.gl/R4yA6u" }

        if ($CentralMgmtServer) {
            if
([Reflection.Assembly]::LoadWithPartialName("Microsoft.SqlServer.Management.
RegisteredServers") -eq $null )
            { throw "Can't load CMS assemblies. You must have SQL Server
Management Studio installed to use the -CentralMgmtServer switch." }

            $server = New-Object Microsoft.SqlServer.Management.Smo.Server
$CentralMgmtServer
            $sqlconnection = $server.ConnectionContext.SqlConnectionObject

            try { $cmstore = new-object
Microsoft.SqlServer.Management.RegisteredServers.RegisteredServersStore($sql
connection)}
            catch { throw "Cannot access Central Management Server" }
            $dbstore = $cmstore.DatabaseEngineServerGroup
            $servers = $dbstore.GetDescendantRegisteredServers().servername
            # Add the CM server itself, which can't be stored in the CM
server.
            $servers += $CentralMgmtServer
            $basenames = @()
            foreach ($server in $servers) { $basenames +=
$server.Split("\")[0] }
            $servers = $basenames | Get-Unique
        }

        If ($ServersFromFile) {
            if ((Test-Path $ServersFromFile) -eq $false) { throw "Could not
find file: $ServersFromFile" }
            $servers = Get-Content $ServersFromFile
        }

        if ([string]::IsNullOrEmpty($servers)) { $servers =
$env:computername }

        $basepath = "SOFTWARE\Microsoft\Microsoft SQL Server"
        # Loop through each server
        $objectCollection = @()
        foreach ($servername in $servers) {
```

```
            $servername = $servername.Split("\")[0]

            if ($servername -eq "." -or $servername -eq "localhost" -or
$servername -eq $env:computername) {
                $localmachine = [Microsoft.Win32.RegistryHive]::LocalMachine
                $defaultview = [Microsoft.Win32.RegistryView]::Default
                $reg =
[Microsoft.Win32.RegistryKey]::OpenBaseKey($localmachine,$defaultview)
            } else {
                # Get IP for remote registry access. It's the most reliable.
                try { $ipaddr =
([System.Net.Dns]::GetHostAddresses($servername)).IPAddressToString }
                catch { Write-Warning "Can't resolve $servername. Moving
on."; continue }

                try {
                    $reg =
[Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey("LocalMachine", $ipaddr)
                } catch { Write-Warning "Can't access registry for
$servername. Is the Remote Registry service started?"; continue }
            }

            $instances = $reg.OpenSubKey("$basepath\Instance
Names\SQL",$false)
            if ($instances -eq $null) { Write-Warning "No instances found on
$servername. Moving on."; continue }
            # Get Product Keys for all instances on the server.
            foreach ($instance in $instances.GetValueNames()) {
                if ($instance -eq "MSSQLSERVER") { $sqlserver = $servername
} else { $sqlserver = "$servername\$instance" }

                $subkeys = $reg.OpenSubKey("$basepath",$false)
                $instancekey =  $subkeys.GetSubKeynames() | Where-Object {
$_ -like "*.$instance" }
                if ($instancekey -eq $null) { $instancekey = $instance } #
SQL 2k5

                # Cluster instance hostnames are required for SMO connection
                $cluster =
$reg.OpenSubKey("$basepath\$instancekey\Cluster",$false)
                if ($cluster -ne $null) {
                    $clustername = $cluster.GetValue("ClusterName")
                    if ($instance -eq "MSSQLSERVER") { $sqlserver =
$clustername } else { $sqlserver = "$clustername\$instance" }
                }

                Write-Verbose "Attempting to connect to $sqlserver"
                $server = New-Object
Microsoft.SqlServer.Management.Smo.Server $sqlserver
                try { $server.ConnectionContext.Connect() } catch { Write-
Warning "Can't connect to $sqlserver or access denied. Moving on."; continue
```

```
}
                $servicePack = $server.ProductLevel

                    switch ($server.VersionMajor) {
                         9 {
                            $sqlversion = "SQL Server 2005 $servicePack"
                            $findkeys =
$reg.OpenSubKey("$basepath\90\ProductID",$false)
                                foreach ($findkey in
$findkeys.GetValueNames()) {
                                    if ($findkey -like "DigitalProductID*")
{ $key = "$basepath\90\ProductID\$findkey"}
                                }
                        }
                        10 {
                            $sqlversion = "SQL Server 2008 $servicePack"
                            $key = "$basepath\MSSQL10"
                            if ($server.VersionMinor -eq 50) { $key +=
"_50"; $sqlversion = "SQL Server 2008 R2 $servicePack" }
                            $key += ".$instance\Setup\DigitalProductID"
                        }
                        11 { $key =
"$basepath\110\Tools\Setup\DigitalProductID"; $sqlversion = "SQL Server 2012
$servicePack" }
                        12 { $key =
"$basepath\120\Tools\Setup\DigitalProductID"; $sqlversion = "SQL Server 2014
$servicePack" }
                        default { Write-Warning "SQL version not currently
supported."; continue }
                    }
                if ($server.Edition -notlike "*Express*") {
                    try {
                        $subkey = Split-Path $key; $binaryvalue = Split-Path
$key -leaf
                        $binarykey =
$($reg.OpenSubKey($subkey)).GetValue($binaryvalue)
                    } catch {$sqlkey = "Could not connect." }
                    $sqlkey = Unlock-SQLServerKey $binarykey
$server.VersionMajor
                } else { $sqlkey = "SQL Server Express Edition"}
                $server.ConnectionContext.Disconnect()

                $object = New-Object PSObject -Property @{
                    "SQL Instance" = $sqlserver
                    "SQL Version" = $sqlversion
                    "SQL Edition" = $server.Edition
                    "Product Key" = $sqlkey
                }
                $objectCollection += $object
            }
            $reg.Close()
```

```
        }
        $objectCollection | Select "SQL Instance", "SQL Version", "SQL
Edition", "Product Key"
    }

    END {
        #Write-Host "Script completed" -ForegroundColor Green
    }
}
```