Enforce Windows Registry Settings Worklet

This worklet describes how to manage device configurations through various registry settings, including 64-bit vs 32-bit registry access.

It is often valuable to change or enforce the behavior of a device by defining a particular registry value. The following is an example that can be extended to just about any setting that is useful to your situation.

Evaluate Current Setting

Before changing a setting, it is best to evaluate its current state before acting. Additionally, this allows the worklet to report Compliance to the desired state.

In this example, all you need to edit are the three variables you want to monitor/change. Set the path to the property, the name of the property, and the desired value. Use this as your evaluation code in your worklet:

```
# Define Registry Key and sub-value to evaluate
#############################################
$regPath = "HKLM:\SYSTEM\CurrentControlSet\Control\Terminal Server"
$regProperty = "fDenyTSConnections"
$desiredValue = '1'
#############################################

# Retrieve current value for comparison
$currentValue = (Get-ItemProperty -Path $regPath -Name $regProperty -ErrorAction Silen
tlyContinue).$regProperty

# Compare current with desired and exit accordingly.
# 0 for Compliant, 1 for Non-Compliant
if ($currentValue -eq $desiredValue) {
   Exit 0
} else { Exit 1 }
```

The policy is marked as compliant/non-compliant based on the exit code. So a simple IF statement that exits accordingly is all that is needed to determine that state of the policy.

Remediating the Setting

After determining compliance, now it's time to act on that information. A non-compliant evaluation flags the device as needing remediation. So when the worklet's scheduled time arrives the device runs the remediation code. Here's the example that goes along with the evaluation above. Use this as your remediation code:

```
# Define Registry Key and sub-value to modify
############################################
$regPath = "HKLM:\SYSTEM\CurrentControlSet\Control\Terminal Server"
$regProperty = "fDenyTSConnections"
$desiredValue = '1'
############################################

try {
   Set-ItemProperty -Path $regPath -Name $regProperty -Value $desiredValue -ErrorActio
n Stop
   Exit 0
} catch {
   Write-Output "Unable to update $regProperty"
   Exit 1
}
```

Again, just set the 3 variables to match the evaluation code. This will set the property to the your desired value. Success/failure here is determined again by the exit code. So a try/catch block is well suited to this scenario.

64-Bit Registry Workaround

Automox runs as a 32-bit process and looks at the 32-bit registry keys by default on a 64-bit device. This doesn't affect the current example, as it isn't looking in an affected registry locations, but this is worth noting for other scenarios.

For instance, all registry keys with a WOW6432Node counterpart (most commonly *HKLM:\SOFTWARE)* the 32-bit WOW6432Node counterpart will be read unless using the following workaround.

So *HKLM:\SOFTWARE* is interpreted as *HKLM:*\SOFTWARE\WOW6432Node if it is run as previously defined.

To get around this behavior, we can use the*sysnative* PowerShell.exe on the device. The following example uses a ScriptBlock to accomplish this. Note that some variables need to be defined inside the ScriptBlock as that code is run in its own context.

Evaluation

```
#### Check Registry with ScriptBlock

$scriptBlock = {
    # Define Registry Key and sub-value to evaluate
    ###############################################
    $regPath = "HKLM:\SYSTEM\CurrentControlSet\Control\Terminal Server"
    $regProperty = "fDenyTSConnections"
    ###############################################

    # Retrieve current value for comparison
    $currentValue = (Get-ItemProperty -Path $regPath -Name $regProperty).$regProperty

    return $currentValue
}

# Define Desired Value here
$desiredValue = '1'

# Execute the ScriptBlock in a 64-bit shell,
# No need to assign to a variable if you don't
# have a return value. In most cases, you will.
$currentValue = & "$env:SystemRoot\sysnative\WindowsPowerShell\v1.0\powershell.exe" -E
xecutionPolicy Bypass -WindowStyle Hidden -NoProfile -NonInteractive -Command $scriptB
lock

# Compare current with desired and exit accordingly.
# 0 for Compliant, 1 for Non-Compliant
if ($currentValue -eq $desiredValue) {
   Exit 0
} else { Exit 1 }
```

Remediation

```
#### Remediate Registry with Scriptblock

$scriptBlock = {
    # Define Registry Key and sub-value to modify
    ###############################################
    $regPath = "HKLM:\SYSTEM\CurrentControlSet\Control\Terminal Server"
    $regProperty = "fDenyTSConnections"
    $desiredValue = '1'
    ###############################################

    try {
       Set-ItemProperty -Path $regPath -Name $regProperty -Value $desiredValue

       return 0
    } catch {
       return 1
    }
}


# Execute the ScriptBlock in a 64-bit shell,
## Since this block returns 0 for success and 1 for fail
## Use that value for exit code to determine success
$returnCode = & "$env:SystemRoot\sysnative\WindowsPowerShell\v1.0\powershell.exe" -Exe
cutionPolicy Bypass -WindowStyle Hidden -NoProfile -NonInteractive -Command $scriptBlo
ck

Exit $returnCode
```

Note: The remediation examples included here assume that the registry key in question already exists. In the case where it does not, additional logic could be added to create the key/value as needed.