Table of Contents

RPS v3.1.0

Last updated on June 14, 2021.

Welcome to the RPS v3.1.0 Documentation Landing Page

Rapid Provisioning System (**RPS**) is a flexible and powerful automation tool for managing software installation, updates, and configuration.

RPS Vision Statement

To build a robust automation framework that empowers organizations to quickly and securely build, configure, and maintain their systems' desired state over any network.

⚠ **IMPORTANT**

Documentation bundled with RPS v3.1.0 is accurate as of **1/15/2021**.

Updated documentation can be found at: https://reactr.azurewebsites.us

What's new in 3.1.0

Introduction

This document describes what is new in the Rapid Provisioning System (RPS). RPS is a flexible and powerful automation tool for managing software installation and configuration. This document focuses on what is new in 3.1.0 and is written from a User's point of view.

What's new in RPS Version 3.1.0

RPS Version 3.1 contains new features for PowerSTIG 4.1 and a revamped patching system. The sections below show the technical updates to the system.

New PowerShell CMDLETS

This section contains a list of new PowerShell CMDLETS to support patching for the 3.1 release.

NAME	DESCRIPTION
New-RpsGroupCondition	Creates a Condition object to add to a Resource Group or Target Group to dynamically add Members.
Get-RpsGroupFilter	Gets the filter object on a Resource Group or Target Group
Remove-RpsGroupCondition	Removes a condition from a Resource Group or Target Group
Get-RpsResolvedParameter	Resolves a Parameter against a Target Item
New-RpsPackageStream	Creates new Package Stream and Packages
New-RpsPackage	Creates a new Package object and adds it to a Package Stream
Update-RpsPackageStream	Updates an existing Package Stream
Remove-RpsPackage	Removes an existing Package from a Package Stream and optionally removes all assignments
Remove-RpsPackageStream	Removes an existing Package Stream, Packages, and optionally removes all assignments
Get-RpsPackage	Gets an existing Package
Get-RpsPackageStream	Gets an existing Package Stream
Get-RpsMaintenanceWindow	Gets an existing Maintenance Window
New-RpsMaintenanceWindow	Creates a new Maintenance Window
Remove-RpsMaintenanceWindow	Removes an existing Maintenance Window
Set-RpsMaintenanceWindow	Sets a Maintenance Window if it exists or creates a new Maintenance Window if it doesn't exist
Get-RPSPackagingManifestSchema	Gets the xml schema for the manifest file for a RPS Package.

NAME	DESCRIPTION
Test-RPSPackagingManifestSchema	Test the inputted XML file to validate it against the packaging schema and conditional requirments.

Table 1: PowerShell CMDLETS

Updated PowerShell CMDLETS

This section includes PowerShell CMDLETS that were updated for the support of patching in version 3.1.

NAME	UPDATE
New-RpsResourceGroup: New parameters added	Operator Condition
New-RpsTargetGroup: New parameters added	Operator Condition

Table 2: PowerShell CMDLETS Updates

Updated DSC Utilities

This section describes the updates for resolving PowerShell Parameters to take a TargetItem and the ParameterMetedata object.

- You now do not need to have a resource assignment to a DSC partial or the parameter imported into the CMDB. You would use this in an external PowerShell script to resolve properties from the CMDB.

Updated PowerShell Functions

This section describes the updates to the PowerShell functions in RPS 3.1.

Name Module Description

Resolve RpsNode (**Breaking change**) The function now requires user to be in an active session context or connected to a SQL database. It no longer uses hard-coded strings to resolve the names of nodes. Resolve RpsNode Can optionally return the Node object versus just the string name Resolve RpsNode Added ability to filter out default node

Table 3: PowerShell functions

Updated RPS Installer

- Added ability to deploy a preconfigured RHEL virtual machine with the RHELTemplateFilename switch parameter
- Group ContentCreators that has permission to the CDN folder for adding content (e.g. Packages)

Sync Service Changes

- Item Properties have a SyncScope to define how the property should be synchronized
    - Public - synchronize
    - Private - do not synchronize
    - Internal - synchronize only to internal nodes

    - InternalDownstream - synchronize only to internal children
    - InternalUpstream - synchronize only to internal parents

Content Delivery Network (CDN Changes

- New properties on Node to define the protocol: ParentCdnProtocol and ChildCdnProtocol - This will be BITS or DFSR so the nodes know how to communicate with one another
- Set-RpsResourceType: Add parameters:
    - CDNDirection (Upstream or Downstream) - This can be used by content (e.g. Packages) to specify the direction it should be synchronized
    - IsContentDistribution - This indicates whether a resource refers to content that will be synchronized

New RPS Type Definitions

- Package: Defines a Package and its properties
- Package Stream: Defines a Package Stream and its properties

New DSC Resources

- RPS Package Manager: DSC Resource for RPS Package Streams and Packages. Provides tools to Get, Set, and Test Packages for a Target

New PowerShell Modules

- RPS Package Provider: Provides methods to Find, Install, or Uninstall packages; Get list of installed packages; Add or Remove Packages Sources

New RPS Web UI Features

- Packaging section added (RPS Menu > Distribution > Packaging)
    - Approvals tab: Approve or Reject Package Streams
    - Scheduling tab: Create, Edit, and Delete Maintenance Windows
    - History tab: View the deployment status of Package Streams, Packages, and Assignments

Rapid Provisioning System Release Notes

Last updated on April 20, 2021.

> Released on February 28, 2020

What's new in 3.1.0 (Feb 28)

- New PowerShell CMDLETS
  - New-RpsGroupCondition: Creates a Condition object to add to a Resource Group or Target Group to dynamically add Members.
  - Get-RpsGroupFilter: Gets the filter object on a Resource Group or Target Group
  - Remove-RpsGroupCondition: Removes a condition from a Resource Group or Target Group
  - Get-RpsResolvedParameter: Resolves a Parameter against a Target Item.
  - New-RpsPackageStream: Creates new Package Stream and Packages
  - New-RpsPackage: Creates a new Package object and adds it to a Package Stream
  - Update-RpsPackageStream: Updates an existing Package Stream
  - Remove-RpsPackage: Removes an existing Package from a Package Stream and optionally removes all assignments
  - Remove-RpsPackageStream: Removes an existing Package Stream, Packages, and optionally removes all assignments
  - Get-RpsPackage: Gets an existing Package
  - Get-RpsPackageStream: Gets an existing Package Stream
  - Get-RpsMaintenanceWindow: Gets an existing Maintenance Window
  - New-RpsMaintenanceWindow: Creates a new Maintenance Window
  - Remove-RpsMaintenanceWindow: Removes an existing Maintenance Window
  - Set-RpsMaintenanceWindow: Sets a Maintenance Window if it exists or creates a new Maintenance Window if it doesn't exist
  - Enable-RpsCdn: Turns On/Off Bits and/Or Dfsr communication.
- Updated PowerShell CMDLETS
  - New-RpsResourceGroup: Add parameters Operator and Condition
  - New-RpsTargetGroup: Add parameters Operator and Condition
- Updated DSC Utilities for resolving PowerShell Parameters to take a TargetItem and the ParameterMetedata object. You now do not need to have a resource assignment to a DSC partial or the parameter imported into the CMDB. You would use this in an external PowerShell script to resolve properties from the CMDB.
- Updated PowerShell Functions
  - Resolve-RpsNode: (**Breaking change**) The function now requires user to be in an active session context or connected to a SQL database. It no longer uses hard-coded strings to resolve the names of nodes.
  - Resolve-RpsNode: Can optionally return the Node object versus just the string name
  - Resolve-RpsNode: Added ability to filter out default node
- Updated RPS Installer
  - Added ability to deploy a preconfigured RHEL virtual machine with the RHELTemplateFilename switch parameter
  - Group ContentCreators that has permission to the CDN folder for adding content (e.g. Packages)
- Sync Changes
  - Item Properties have a SyncScope to define how the property should be synchronized
    - Public - synchronize
    - Private - do not synchronize

        - Internal - synchronize only to internal nodes
        - InternalDownstream - synchronize only to internal children
        - InternalUpstream - synchronize only to internal parents

- CDN Changes

    - New properties on Node to define the protocol: ParentCdnProtocol and ChildCdnProtocol - This will be BITS or DFSR so the nodes know how to communicate with one another
    - Set-RpsResourceType: Add parameters:
        - CDNDirection (Upstream or Downstream) - This can be used by content (e.g. Packages) to specify the direction it should be synchronized
        - IsContentDistribution - This indicates whether a resource refers to content that will be synchronized

    - New Resource Item call CdnSettings that gets globally created with two Internal properties, IsBitsEnabled and IsDfsrEnabled. The two properties only sync within internal nodes. i.e. within a unit.

- New RPS Type Definitions

    - Package: Defines a Package and its properties
    - Package Stream: Defines a Package Stream and its properties

- New DSC Resources

    - RPS Package Manager: DSC Resource for RPS Package Streams and Packages. Provides tools to Get, Set, and Test Packages for a Target

- New PowerShell Modules

    - RPS Package Provider: Provides methods to Find, Install, or Uninstall packages; Get list of installed packages; Add or Remove Packages Sources

- New RPS Web UI Features

    - Packaging section added (RPS Menu > Distribution > Packaging)
        - Approvals tab: Approve or Reject Package Streams
        - Scheduling tab: Create, Edit, and Delete Maintenance Windows
        - History tab: View the deployment status of Package Streams, Packages, and Assignments

- Added new function Test-DscModuleConflict to use for testing for required DSC Module conflicts across all assigned partials for a single Target Item.

- Added Get-AdminRoleCredential to Rps-Api-Utilities; used to determine which credential role to use in a task

Known Issues in 3.1.0

- Web UI

    - Closing the Remove Resource modal resets filters
        - From the Resources screen, if you have a filter added for Packages and you confirm and remove a package it will clear your filters
        - Workaround: Manually re-apply your filters

    - Cannot assign Target Items to an Item Group from the Web UI

        - From the RPS Web UI menu: Targeting > Item Groups
        - Open the patchable target group by clicking on the name
        - Open the Members accordion
        - When you try to click “+ Add New Item” you’ll receive an error

        - Workaround: PowerShell can be used to add Target items to an Item G

        - Workaround: PowerShell can be used to add Target Items to an Item Group:

          ```
          $ti = Get-RpsTargetItem -Id IdOfMyTargetItem
          $tg = Get-RpsTargetGroup -Id IdOfMyGroup
          $tg.AddChildItem($ti)
          $tg.Update()
          ```

    - glyphicons halflings are not signed and do not display in Web UI when deployed due to STIG
        - Some icons (e.g. status icons for Packages) do not display in Internet Explorer when deployed due to STIG
        - Workaround: The status can be easily discerned by font color and wording of the statuses.

- API
    - Dates are parsed out of Properties as strings and we always lose timezone specifications
        - This could cause some times to be slightly incorrect depending on how they were stored and/or retrieved
        - Workaround: No known workaround.

    - Currently there is no support for Maintenance Windows that span across 2 or more days
        - Workaround: Create 2 maintenance windows - one for each day so it covers the entire period of time desired.

- PowerShell cmdlets

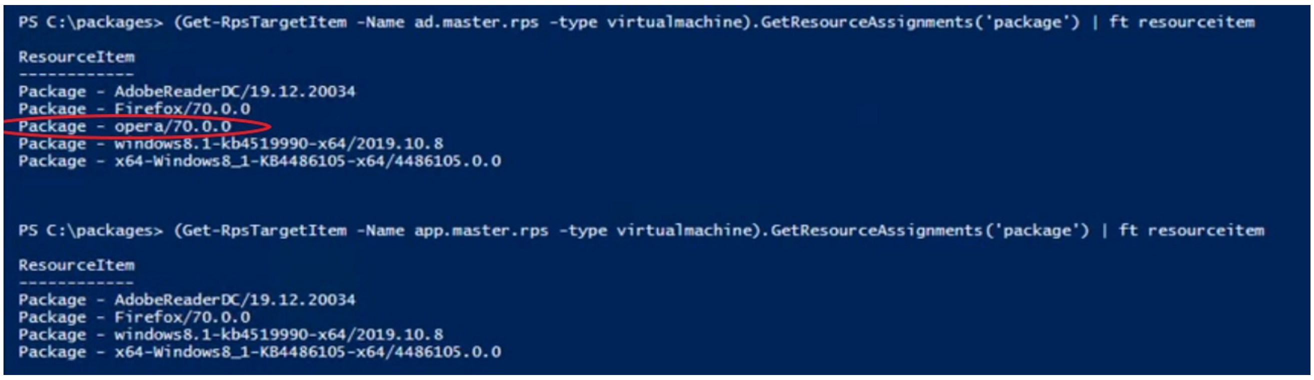
    - DateTime for Start/End date in Get-RpsMaintenanceWindow displays inaccurate time
        - Get-RpsMaintenanceWindow returns the Start/End date which includes a time, however, the time is inaccurate and it conflicts with the start/end times that are also returned to the user
        - Workaround: No known workaround. This is a visual/display issue. The time portion included alongside the Start/End dates that are shown are not what is being used. The actual times being used are displayed separately.

- Package Manifest **Conditions** element **Value** field does not support multiple values separated by the pipe delimiter **|**

    - **Error Details:** The following PackageManifest code snippet is an example using pipe delimiter **|** in **Conditions**, which will fail:

      ```
      <InstallerFileName>opera.msi</InstallerFileName>
      <Conditions>
        <PackageAssignmentCondition>
          <Property>Name</Property>
          <Operator>Eq</Operator>
          <Value>AD.master.rps|APP.master.rps</Value>
        </PackageAssignmentCondition>
      </Conditions>
      ```

      The resulting behavior: Only the first Value listed will be assigned to; all other Values after the pipe delimiter **|** are ignored.



```
PS C:\packages> (Get-RpsTargetItem -Name ad.master.rps -type virtualmachine).GetResourceAssignments('package') | ft resourceitem

ResourceItem
------------
Package - AdobeReaderDC/19.12.20034
Package - Firefox/70.0.0
Package - opera/70.0.0
Package - windows8.1-kb4519990-x64/2019.10.8
Package - x64-Windows8_1-KB4486105-x64/4486105.0.0


PS C:\packages> (Get-RpsTargetItem -Name app.master.rps -type virtualmachine).GetResourceAssignments('package') | ft resourceitem

ResourceItem
------------
Package - AdobeReaderDC/19.12.20034
Package - Firefox/70.0.0
Package - windows8.1-kb4519990-x64/2019.10.8
Package - x64-Windows8_1-KB4486105-x64/4486105.0.0
```

      In this particular example, AD.master.rps is assigned the opera Package, because it was listed before the pipe delimiter **|** . APP.master.rps is not assigned the opera Package, because it was listed after the pipe delimiter **|** .

- **Current Workaround for pipe delimiter | :** Utilize the Match Operator `<Operator>Match</Operator>`, with each value in the Value field wrapped in parentheses **()** and with a trailing question mark **?** . Example:

  ```
  <Conditions>
      <PackageAssignmentCondition>
          <Property>ComputerName</Property>
          <Operator>Match</Operator>
          <Value>(NFA)?(WNM)?(WNMA)?</Value>
      </PackageAssignmentCondition>
  </Conditions>
  ```

  In this particular example, a Target with a Property of ComputerName will be assigned if its Value contains NFA, WNM, **and/or** WNMA. This implementation only requires a **partial** Value match.

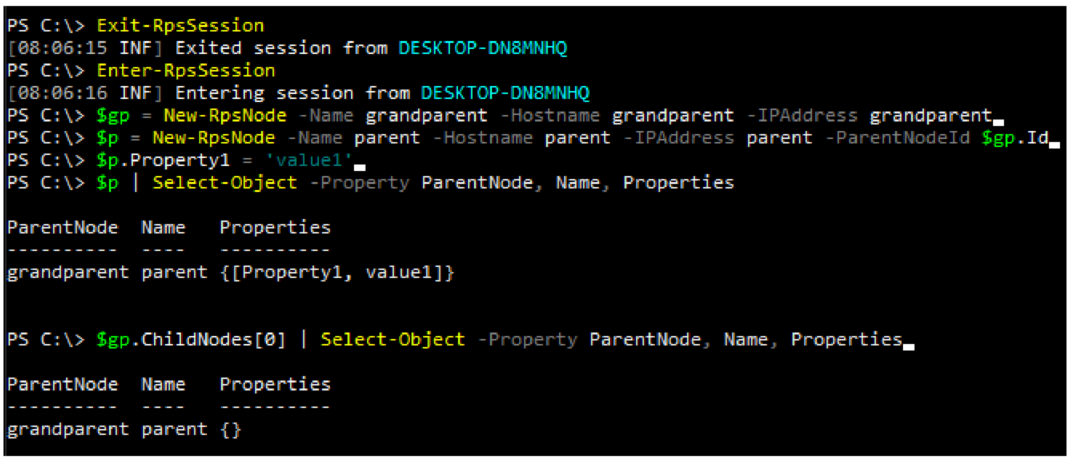
  For an **exact** Value match, the full string in the Value field must be enclosed with a caret **^** and a dollar sign **$** . Example:

  ```
  <Conditions>
      <PackageAssignmentCondition>
          <Property>ComputerName</Property>
          <Operator>Match</Operator>
          <Value>^(NFA)?(WNM)?(WNMA)?$</Value>
      </PackageAssignmentCondition>
  </Conditions>
  ```

  In this particular example, a Target with a Property of ComputerName will be assigned if its Value contains NFA, WNM, **and** WNMA.

- Making changes to child node prevents properties from being visible from ancestor nodes when in session **Error Details:** The following PowerShell example shows how making changes to a child node fails to display properties from ancestor nodes (e.g. parent node):

```
$parent = New-RpsNode -Name parent -hostname parent -IPAddress parent
$child = New-RpsNode -Name child -HostName child -IPAddress child -ParentNodeId $parent.Id
$child.Property1 = 'value1'
$child.Properties # This will show Property1
$child.Update() # Calling the Update method will commit the property changes but they still won't be
visible on the parent object
$parent.ChildNodes[0].Properties # This will show the child node without Property1
```



```
PS C:\> Exit-RpsSession
[08:06:15 INF] Exited session from DESKTOP-DN8MNHQ
PS C:\> Enter-RpsSession
[08:06:16 INF] Entering session from DESKTOP-DN8MNHQ
PS C:\> $gp = New-RpsNode -Name grandparent -Hostname grandparent -IPAddress grandparent
PS C:\> $p = New-RpsNode -Name parent -Hostname parent -IPAddress parent -ParentNodeId $gp.Id
PS C:\> $p.Property1 = 'value1'
PS C:\> $p | Select-Object -Property ParentNode, Name, Properties

ParentNode  Name   Properties
----------  ----   ----------
grandparent parent {[Property1, value1]}


PS C:\> $gp.ChildNodes[0] | Select-Object -Property ParentNode, Name, Properties

ParentNode  Name   Properties
----------  ----   ----------
grandparent parent {}
```

NOTE

The .Update() API method (in 3.1; not available in 4.x) and Update-RpsNode (in 3.1; not available in 4.x) have no effect.

Current Workaround for displaying child node properties at parent node level: After making changes to any node that has a Parent Node use the following PowerShell command to fix the issue:

~~~
```PowerShell
$parent.AddChildNode($child)
```
![Workaround Fix Example](../../../Images/v3.1.0/ChildParentWorkaroundFix.png)
~~~

NOTE

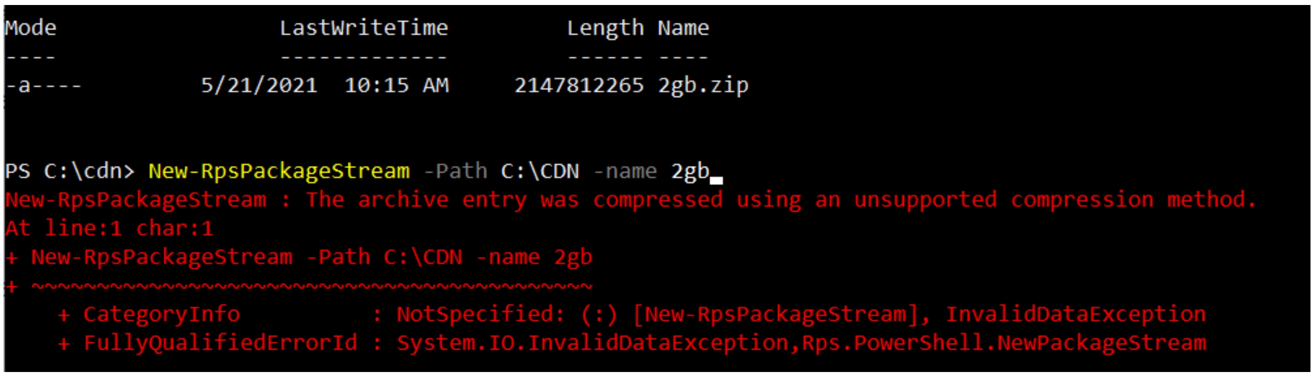
This method works regardless of the ancestry depth. Re-adding a grandchild to its parent makes the properties visible when accessed from the parent and the grandparent.

- 2GB RPS Package size limitation

  The maximum supported RPS Package size is 2GB. Any RPS Package zip file that is larger than 2GB will throw an exception when RPS tries to open the package and read the manifest file from the package zip file.

  - This exception can occur in two scenarios:
    - When creating a new package stream with a package where the zip file size is greater than 2GB.
    - When adding a new package to an existing package stream where the package zip file size is greater than 2GB.

  The resulting behavior:



```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         5/21/2021  10:15 AM     2147812265 2gb.zip


PS C:\cdn> New-RpsPackageStream -Path C:\CDN -name 2gb
New-RpsPackageStream : The archive entry was compressed using an unsupported compression method.
At line:1 char:1
+ New-RpsPackageStream -Path C:\CDN -name 2gb
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [New-RpsPackageStream], InvalidDataException
    + FullyQualifiedErrorId : System.IO.InvalidDataException,Rps.PowerShell.NewPackageStream
```

  Figure: Example of the error encountered when an RPS Package zip file greater than 2GB is used.

> Released on September 20, 2019

What's new in 3.0.3 (Sep 20)

The primary update in this Hotfix release is to address:

- #23726 Fix: Provisioning Node App Server fails to configure DSC. Login failed for user.

> Released on September 19, 2019

What's new in 3.0.2 (Sep 19)

The following are work items completed in support of the 3Q19 Release and delivered in Hotfix 3.0.2. These fixes include Critical and High Risk Factor mitigations.

- #23542 Fix: Address .NET 2.0 Vulnerabilities

- #23543 Fix: Address .NET 3.0 Vulnerabilities
- #23556 Fix: Address CVE-2017-8529 mitigation for Internet Explorer vulnerability
- #23555 Fix: Address 'Memory Management\FeatureSettingsOverride' mitigations
- #23547 Fix: Address Visual C++ 2008 SP1 Vulnerability
- #23548 Fix: Address GPO setting "Hardened UNC Paths" (KB3000483)
- #23544 Fix: Address .NET 3.5 Vulnerabilities
- #23545 Fix: Address SQL Server 2012 SP4 Vulnerability (KB4057116)
- #23550 Fix: Address SSL Version 2 and 3 Protocol Detection

⚠ **IMPORTANT**

The ContentStore has been updated to include various "binary" patches, such as for SQL Server, and Microsoft .NET. Additionally, the 3Q19 **.vhdx** / **.iso** have been updated to include additional Windows patches. Please ensure the latest ContentStore and the 3Q19-2012R2-0919 from the Release are used.

What's new in 3.0.1 (Sep 05)

- #23286 Fix: Task Management Service will throw errors after running for a long time

What's new in 3.0.0 (Aug 23)

- #19833 Timestamp Logic for BITS
- #23135 Certificates issued by RPSRoot do not have FQDN in the SAN
- #22992 Test DCA deployment without a PFX certificate
- #23147 Resource Group membership cannot be updated by subsequent node imports
- #23230 RpsDomainJoin account doesnt get the correct permissions to add a computer to the domain that is prestaged
- #23229 RpsProvisioning dns record is set to Interanl nic instead of 996 and 59 NIC
- #23227 Web Config files are being overwritten by RPSGUI, RPSProvisioning, and TrustedElementRepository DSC Partials
- #23093 xDFSR uses Domain Admin account
- #23049 RVP configured with specific registry settings for compliance
- #21370 Master-Controller fails to resume after service or machine restart
- #23210 SQL SA account name conflict
- #23026 Content Delivery Network Partial is missing a mandatory parameter

What's new in 3.0.0-beta (Aug 16)

- #22496 Update PSScriptAnalyzer to 1.18.1
- #22311 When installing the content store to a directory other than c:/contentstore certificates are installed in the wrong path
- #22432 Remove RVPS GUI files and install and powerstig from the release
- #23035 RpsDomainJoin accounts are set to Create = False within RpsAccounts.csv file
- #22647 Configure a new Packaging Repository and migrate our code out of Core.
- #22684 Updates needed to the Ports and Protocols section of the RPS Install Guide.pdf
- #21495 RPS currently does not have a way to continually re-publish DSC partials
- #22537 Update OSS registrations and Third Party Notices file
- #22913 Files located under the folder c:\ContentStore\Export are not encrypted (on the APP VM)
- #22948 Failing resource on RVP - [xPackage]ACCM
- #19973 Deploy PowerSTIG 3.3.0
- #22541 Add a script resource RpsDomainController.ps1 to execute certutil.exe - installdefaulttemplates
- #22243 Automate Axway Desktop Validator Enterprise
- #22483 Need to update SkipRules for DSC PowerStig configuration
- #22433 Access Database partial is assigned in colorless baseline data for RVP

- #22683 RPS Install does not work per published installation directions
- #22694 Export-NodeData Runbook variable $TargetItem is not correctly referenced
- #21970 Unable to export taskmap definitions with Export-RpsData
- #22015 Update RPS logging during deployment to better characterize issues
- #21971 Lot 7 NOSC NIPR RVP ActivClientAppInstall patch fails (RPS 2.4.5)
- #22607 Failing resource - [xPackage]ActivClient71
- #21560 DSC Partials should only require OSCore when necessary
- #22549 Update DCA Assignments.psd1
- #22557 Update source Certificate locations and update the Certificates.psd1 and CertificationAuthority partial.
- #20605 TrustElementRootPath gets set to wrong path
- #21458 APP and AD VMs do not have PowerStig configurations
- #22246 Any website on the c:\ drive is a CAT II finding
- #22248 RPSAdmin domain account password should be user configurable for APP and AD
- #22417 Newly generated self-signed certificates sometimes not loaded into the CMDB
- #22426 Import-NodeData fails on APP VM when importing node data
- #22429 Duplicate Import-RPSNode Functions
- #22436 OCSP website has request filters that need to be removed
- #22449 RVP - CdnPath points to C$ instead of share
- #22458 Registry resource failing to add registry keys for TER authorization
- #22593 Update to only install McAfee agent 5.5
- #22594 RVP - CdnPath points to C$ instead of share
- #22598 Registry resource failing to add registry keys for TER authorization
- #22670 DomainJoinAdmin gets access denied when joining RVP to the domain
- #22677 Remove Install-MNRps.ps1 as it is no longer used.
- #22682 RpsProvisioning folder path creation should not use the FQDN for folder name
- #22691 Generated certificates are missing FQDN for subject name
- #22743 RpsGUI not reaching desired state due to a certificate error
- #22936 Existing WinRm settings on a target cause the set-winrm runbook to fail.
- #23001 Missing SSL binding reg key in trust element repository partial
- #23015 Provisioning node configuration has missing master key encryption role on several accounts
- #23034 Copy-BaseImages references the wrong local account for credentials.
- #22454 Failing resource on RVP - [AdcsOnlineResponder]OnlineResponder
- #22464 OSCore New Computername Timing
- #22666 Deploy the RPS 3.x codebase in Hyper-V
- #22725 Install-Rps.ps1 does not update MN node target items' VhdTemplateFileName property
- #22542 Replace $DomainAdmin with $CAServiceAccount in the CertificationAuthority.ps1 partial.
- #22555 Inhibit and restart Tumbleweed service to the DesktopValidatorStandardAppInstall.ps1
- #22587 Add the Certificate partial dependency to the CertificationAuthority partial.
- #22945 Create a partial for the Exit Module
- #23073 Update the CA Partial to use RPSadmin
- #23074 DomainJoin resource fails on DCA
- #23075 RVP needs to be a member of TPKI Writes AD Group

What's New in 3.0.0-beta (Aug 9)

- #22432 Remove RVPS GUI files and install and powerstig from the release
- #22311 When installing the content store to a directory other than c:/contentstore certificates are installed in the wrong path
- #22684 Updates needed to the Ports and Protocols section of the RPS Install Guide.pdf
- #22483 Need to update SkipRules for DSC PowerStig configuration
- #22433 Access Database partial is assigned in colorless baseline data for RVP
- #22694 Export-NodeData Runbook variable $TargetItem is not correctly referenced

- #22549 Update DCA Assignments.psd1
- #21458 APP and AD VMs do not have PowerStig configurations
- #22670 DomainJoinAdmin gets access denied when joining RVP to the domain
- #22429 Duplicate Import-RPSNode Functions
- #22691 Generated certificates are missing FQDN for subject name
- #22426 Import-NodeData fails on APP VM when importing node data
- #22458 Registry resource failing to add registry keys for TER authorization
- #22598 Registry resource failing to add registry keys for TER authorization
- #22677 Remove Install-MNRps.ps1 as it is no longer used.
- #22682 RpsProvisioning folder path creation should not use the FQDN for folder name
- #22449 RVP - CdnPath points to C$ instead of share
- #22594 RVP - CdnPath points to C$ instead of share
- #20605 TrustElementRootPath gets set to wrong path
- #22593 Update to only install McAfee agent 5.5
- #22725 Install-Rps.ps1 does not update MN node target items' VhdTemplateFileName property
- #22496 Update PSScriptAnalyzer to 1.18.1
- #22647 Configure a new Packaging Repository and migrate our code out of Core.
- #22537 Update OSS registrations and Third Party Notices file
- #22541 Add a script resource RpsDomainController.ps1 to execute certutil.exe - installdefaulttemplates
- #22243 Automate Axway Desktop Validator Enterprise
- #22015 Update RPS logging during deployment to better characterize issues
- #21560 DSC Partials should only require OSCore when necessary
- #22555 Add inhibit and restart Tumbleweed service to the DesktopValidatorStandardAppInstall.ps1
- #22587 Add the Certificate partial dependency to the CertificationAuthority partial.
- #20407 Configure BITS/DFSR per node type

Files excluded from the drop!

In order to improve the speed with which RPS artifacts can be integrated with other code repositories, the decision was made to exclude files from Core which required modification later in the integration process for Mission Network. These files and folders are below:

- DSC\Modules\NetworkingDSC\6.1.0.0\DSCResources\MSFT_HostsFile\MSFT_HostsFile.psm1
- DSC\Modules\MN_OfficeDSC\
- DSC\Modules\MN_SchemaExtensionDSC\
- DSC\Modules\MN_xWinEventLog\
- DSC\Modules\WINT_NetworkPolicyServer\
- Modules\MN-AnalyzerRules\
- Modules\MN-Automation\
- Modules\MN-ISO\
- Modules\MN-Rps-Api\
- Modules\MN-Ssh\
- Modules\MN-VMWare-Utilities\
- DSC\PartialConfigurations\ActivClientAppInstall.ps1
- DSC\PartialConfigurations\AdobeReaderAppInstall.ps1
- DSC\PartialConfigurations\AdSchemaExtension.ps1
- DSC\PartialConfigurations\ClientPki.ps1
- DSC\PartialConfigurations\DesktopValidatorStandardAppInstall.ps1
- DSC\PartialConfigurations\DoDInstallRootAppInstall.ps1
- DSC\PartialConfigurations\FirefoxAppInstall.ps1
- DSC\PartialConfigurations\Firewall.ps1
- DSC\PartialConfigurations\GpoWmiFilter.ps1

- DSC\PartialConfigurations\GroupPolicy.ps1
- DSC\PartialConfigurations\McAfeeHBSSAppInstall.ps1
- DSC\PartialConfigurations\MsftAppLocker.ps1
- DSC\PartialConfigurations\MsftDnsServer.ps1
- DSC\PartialConfigurations\NetBannerAppInstall.ps1
- DSC\PartialConfigurations\OcspResponder.ps1
- DSC\PartialConfigurations\Office2013AppInstall.ps1
- DSC\PartialConfigurations\OpenSSLAppInstall.ps1
- DSC\PartialConfigurations\OracleJDKAppInstall.ps1
- DSC\PartialConfigurations\OracleJREAppInstall.ps1
- DSC\PartialConfigurations\PuTTYAppInstall.ps1
- DSC\PartialConfigurations\RvpsGUI.ps1
- DSC\PartialConfigurations\SmartCardManager90MeterAppInstall.ps1
- DSC\PartialConfigurations\SoftphoneAppInstall.ps1
- DSC\PartialConfigurations\SolarWindsETAppInstall.ps1
- DSC\PartialConfigurations\TeraTermAppInstall.ps1
- DSC\PartialConfigurations\TigerVNCAppInstall.ps1
- DSC\PartialConfigurations\TrustElementRepository.ps1
- DSC\PartialConfigurations\VMWareClientIntegrationPlugInAppInstall.ps1
- DSC\PartialConfigurations\VMWareRemoteConsoleAppInstall.ps1
- DSC\PartialConfigurations\VMWareToolsAppInstall.ps1
- DSC\PartialConfigurations\VMWarevSphereClientAppInstall.ps1
- DSC\PartialConfigurations\VMWarevSpherevCLIAppInstall.ps1
- DSC\PartialConfigurations\WaveDesktopCommunicatorAppInstall.ps1
- Images\ESX\Grangeville.cfg
- iPXE Distro\
- Provisioning\Provisioning Vlan Address Space.csv
- Runbooks\Copy-BaseImages.ps1
- Runbooks\Get-TargetDhcpIPAddress.ps1
- Runbooks\Import-VMWareVirtualAppliance.ps1
- Runbooks\New-VMWareVirtualMachine.ps1
- Runbooks\Remove-VMWareVirtualMachine.ps1
- Setup\
- Utilities\

What's New in 3.0

PowerShell

This release contains the following PowerShell enhancements:

- Added Import-RpsInstanceDefinition and Export-RpsInstanceDefinition to Import/Export InstanceDefinitions as Json.
- Added Import-RpsDataMapping and Export-RpsDataMapping to Import/Export Data Mappings as Json.
- Added Import-RpsResourceItemJson and Export-RpsResourceItemJson to Import/Export Resource Items as Json.
- Added Set-RpsDataImportMapping to the API from Rps-DataMapping module.
- Added Set-RpsDataFilter to the API from Rps-DataMapping module.
- Added Set-RpsDataCondition to the API from Rps-DataMapping module.
- Added Set-RpsDataProperty to the API from Rps-DataMapping module.
- Added Set-RpsDataAssociation to the API from Rps-DataMapping module.
- Added Set-RpsMappingFilter to the API from Rps-DataMapping module.
- Added Set-RpsDataVariable to the API from Rps-DataMapping module.
- Added Set-RpsDataMapping to the API from Rps-DataMapping module.
- Added Set-RpsDataFile to the API from Rps-DataMapping module.

Rps-Installer

This release contains the following Rps-Installer enhancements:

DSC

This release contains the following DSC enhancements:

- MofStore location is now located within C:\ContentPath\DSC.
  - OutputPath parameter is no longer set on the node, or set statically.
  - Publish-DSCConfiguration now creates, and sets the OutputPath parameter for all assigned partials.
  - Removed Mandatory flag from OutputPath parameter on all partials.
  - Updated runbooks to pass OutputPath within calls to LCM functions.
  - Default value for LCM functions within the RPS-DSC module is now the present working directory for stand alone use.

RPS API

This release contains the following API enhancements:

- Adding the following Type Property constants:
  - IsContentDistribution
  - IsSoftwareDistribution
  - IsColumnDisplay
- Updated the Set-RpsResourceType cmdlet to indicate if the Resource Type is for software or content distribution. The IsContentDistribution and IsSoftwareDistribution switches are mutually exclusive in the cmdlet, however, setting the IsSoftwareDistribution switch will also set the IsContentDistribution flag on the Resource Type.
- Updated the Set-RpsTypeProperty cmdlet to indicate if the Property Type can be used for Column Displays within the Admin UI.
- Updated the New-RpsResourceGroup cmdlet to allow for properties to be provided at creation
- Updated the Write-RpsLogItem cmdlet to write to the appropriate PS stream. Tokenization in MessageTemplate is also more forgiving.
- Added Json support for InstanceDefintitions and Data Mappings to help make creation and updating easier.
- Added Json support for Resource Items to have a readable and organized way of import/exporting resource items and sharing between nodes.
- When TaskMaps are included in an export, their TaskMapSteps are also exported by default
- Added an optional Certificate parameter for passing a certificate file (.cer) that will encrypt the resulting configuration file in the Exit-RpsSession and Export-RpsData cmdlets.
- Added optional Certificate and password parameters for passing certificate file (.pfx) and password that will be used to decrypt a configuration file in the Enter-RpsSession and Import-RpsData cmdlets.
- Added support to encrypt/decrypt export files during the install process.

RPS Sync Service

This release contains the following Sync Service enhancements:

RPS CDN

This release contains the following RPS Content Delivery Network (CDN) enhancements:

- DFS-R is used for communication between Region and Site nodes.

- BITS is still used for communication between Master and Region nodes.
- Due to DFS-R mesh networking, all files are replicated to all Region and Site nodes within a domain, regardless of assignment.
- Patches will still only be installed on assigned targets

Admin UI

This release contains the following Admin UI enhancements:

- Replaced 'Patching' on the top menu bar with 'Distriburtion'
- Added dynamically created sections under distriburtion for Content Distriburtion and Software Distriburtion

Resolved Issues

Top issues addressed in 3.0:

- Fixed issue where the SMA runbook service account was denied access to the MofStore after STIGs were applied
- Fixed issue where there was a credential conflict with the Windowsfeature Net-Framework-Core resource, between the RpsSMA, and RpsSQL partial.

Known Issues

SQL Server 2012

This release contains the following SQL enhancements:

- Microsoft SQL Server 2012 has been upgraded from Service Pack 2 (SP2) to SP4

PowerShell

This release contains the following PowerShell enhancements:

- Added support to allow certificate store parameter to be passed from CMDB.

- Added support for additional certificate roles:

    - CAp7b
    - CertificationAuthorityPFX
    - CACertificateChain

    - Added Force switch for Set-RpsResourceItem, Set-RpsTargetItem, New-RpsResourceItem, New-RpsTargetItem, New-RpsResourceGroup, and New-RpsTargetGroup

- Removed module RPS-Credentials and functions:
    - Get-Credential
    - New-Credential
    - Get-ServiceAccount
        - Added data files for Users and Certificates. They can be found here 'Setup\Configuration\Data\RpsAccounts.csv' and here 'Setup\Configuration\Data\RpsCertificates.csv'. For both data files, if no password is provided in the password column, a password will be randomly generated per user/certificate.

Rps-Credential

- New-RpsCredential was updated to allow generation of a password, with or without a provided password policy.

Rps-Installer

- Import-RpsCredential was updated to allow generation of a password, with or without a provided password policy.
- MofStore location was changed from C:\Windows\Temp to C:\ContentStore\DSC

DSC

This release contains the following DSC enhancements:

- Added Certification Authority Partial to install and configure a Certification Authority node.
- Domain Admins no longer joins machines to the domain.
- The RpsDomainJoin account now joins staged computer objects, within the Computers OU, to the domain using the minimum permissions required.
- ContentDeliveryNetwork partial updated to install DFS-R for Region and Site nodes
- Updated Dsc Modules to the following versions:

MODULE	CORE VERSION
AccessControlDsc	1.3.0.0
ComputerManagementDsc	6.2.0.0
PowerStig	3.1.0
xWebAdministration	2.5.0.0
ResourceControllerDSC	2.0.1

RPS API

This release contains the following API enhancements:

- Updated the Set-RpsTargetItem and Update-RpsTargetItem cmdlets to not allow the altering of an existing Parent if it has already been set.
- Adding the following ResourceType constants:
    - CATemplate
    - OcspUriPath
    - CdpUriPath
    - AiaUriPath
    - RegistryItem
    - Crl
    - NPSPolicyMap
    - NPSClient
    - RegistryAccessEntry
    - RegistryAccessControlList
    - RegistryAccessRule
    - CertificationAuthority
- BREAKING CHANGE: TypeDefinitions are now enforced on Target Items. All required params will have to be set when creating the target item.
- Added Instance Definitions, which are pre-defined complex, default data for the purpose of quickly defining data but also codifying configuration data
- Added a User Profile context to the Rps API in order to provide and track the current user. This provides the foundation for a RBAC implementation.
- Added an optional method of export to provide the user with a plaintext XML file where protected properties are in the clear.
- Added an optional CertificateThumbprint parameter to the Enter-RpsSession, Exit-RpsSession, Export-RpsData, and Import-

RpsData cmdlets in order to encrypt/decrypt file exports and imports.

- Added Get-RpsPasswordPolicy cmdlet
- Added Set-RpsPasswordPolicy cmdlet
- Added New-RpsPassword cmdlet
- Added Update-MasterKey cmdlet
- Added Get-RpsProtectedProperty cmdlet.
- Added Instance Definition Nodes, which are pre-defined objects that can be used to create Nodes that are associated with Instance Definitions
- Added Set-RpsTargetType cmdlet.
- Added Set-RpsResourceType cmdlet.
- Added Set-RpsSubType cmdlet.
- Added Set-RpsChildType cmdlet.
- Added Set-RpsTypeProperty cmdlet.
- Added Set-RpsTypeRA cmdlet.
- Added Set-RpsTargetAction cmdlet.
- Added Set-RpsResourceGroupType cmdlet.
- Added Get-RpsCredential cmdlet.
- Added New-RpsCredential cmdlet.
- Added Get-UnixHash cmdlet

- Added Set-RpsTargetGroupType cmdlet.

  Sample:

```
    $secureResult = Get-RpsProtectedProperty -TargetItem $targetItem -Name $name
    $secureResult = Get-RpsProtectedProperty -ResourceItem $resourceItem -Name $name
    $secureResult = Get-RpsProtectedProperty -Node $node -Name $name
    $secureResult = Get-RpsProtectedProperty -TaskMapAssignment $taskMapAssignment -Name $name
    $secureResult = Get-RpsProtectedProperty -ResourceGroup $resourceGroup -Name $name
    $secureResult = Get-RpsProtectedProperty -TargetGroup $targetGroup -Name $name

    # parameter setup
    $name = "propertiy name"

    # return variable to plain text
    $plainText = ConvertFrom-SecureString $secureString
```

- Added Set-RpsProtectedProperty cmdlet.

  Sample:

```
    Set-RpsProtectedProperty -TargetItem $targetItem -Name $name -Value $securePwd
    Set-RpsProtectedProperty -ResourceItem $resourceItem -Name $name  -Value $securePwd
    Set-RpsProtectedProperty -Node $node -Name $name -Value $securePwd
    Set-RpsProtectedProperty -TaskMapAssignment $taskMapAssignment -Name $name -Value $securePwd
    Set-RpsProtectedProperty -ResourceGroup $resourceGroup -Name $name -Value $securePwd
    Set-RpsProtectedProperty -TargetGroup $targetGroup -Name $name -Value $securePwd

    # parameter setup
    $name = "propertiy name"
    $value = ConvertTo-SecureString "Value" -AsPlainText -Force
```

- Enhanced Set-RpsTargetItem and Set-RpsResourceItem cmdlets to accept protected properties from a hashtable using the SecureString type. Sample:

```
  Set-RpsTargetItem  -Name $name -Type $type -Properties @{ Protected = $secureString }
  Set-RpsResourceItem  -Name $name -Type $type -Properties @{ Protected = $secureString }
  ```
```
```

- Breaking Change: Marked New-RPSTaskMapStructure cmdlet as Obsolete.
- Modified the API to mask protected properties when they are returned to the console.
- Added New-RpsInstanceDefinition Cmdlet. Sample:

  ```
      $hs = @{
      Prop1 = "value1"
      Prop2 = "value2"
      New-RpsInstanceDefinition -Name testName -Properties $hs
  ```

- Added New-RpsInstanceDefinitionItem Cmdlet. Sample: An Instance Definition Item is a wrapper for an RPS type and associated Properties.
  `PowerShell New-RpsInstanceDefinitionItem -EntityName testEntityName -Name name2 -Properties @{Prop1 = "Value1"} -TypeDefinitionId $typedefinition.id`
- Added Invoke-RpsInstanceDefinition Cmdlet.
  `PowerShell Invoke-RpsInstanceDefItem -Settings $resourceItem -InstanceDef $instanceDefinition`
- Added Set-RpsInstanceDefinition Cmdlet. Sample:

  ```
      Set-RpsInstanceDefinition -Name $Name1
      Set-RpsInstanceDefinition -Name $Name1 -Properties @{Prop1 = "Value1"}
  ```

  Added Remove-RpsInstanceDefinitionItem Cmdlet.

  ```
      Remove-RpsInstanceDefinitionItem -Id "8825A09C-CCE3-4BB0-BCE1-03B4729AC423"
      Remove-RpsInstanceDefinitionItem -InstanceDefinitionItem $InstanceDefinitionItem
  ```

- Added Get-RpsInstanceDefinition Cmdlet. Sample:

  ```
      Get-RpsInstanceDefinition -Id "8825A09C-CCE3-4BB0-BCE1-03B4729AC423"
      Get-RpsInstanceDefinition -Name MyInstanceDef
  ```

  - Added Remove-RpsInstanceDefinition Cmdlet. Sample:

    ```
      Remove-RpsInstanceDefinition -Id "8825A09C-CCE3-4BB0-BCE1-03B4729AC423"
      Remove-RpsInstanceDefinition -InstanceDefinition $InstanceDefinition
    ```

- Added Set-RpsInstanceDefinitionItem Cmdlet. Sample:

  ```
      Set-RpsInstanceDefinitionItem -Name $Name1 -TypeDefinitionId $id -EntityName $entityName
      Set-RpsInstanceDefinitionItem -Name $Name1 -TypeDefinitionId $id -Properties @{Prop1 = "Value1"} -
  EntityName $entityName
  ```

- Added Get-RpsInstanceDefinitionReference. Sample:

  ```
  $instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
  $instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
  $reference = Get-RpsInstanceDefinitionReference -Name "name" -InstanceDefinition $instanceDef -
  InstanceDefinitionItem $instanceDefItem
  ```

- Added New-RpsInstanceDefinitionReference. Sample:

  ```
  $instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
  $instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
  $taskMapIDs = "5b8b0340-091f-4823-b2f9-de937b5b4114", "a83b5445-3cc0-433e-b5e0-0fcf70389988"
  $reference = New-RpsInstanceDefinitionReference -Name "name" -InstanceDefinition $instanceDef -
  InstanceDefinitionItem $instanceDefItem -TaskMapIDs $taskMapIDs
  ```

- Added Remove-RpsInstanceDefinitionReference. Sample:

```
$instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
$instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
Remove-RpsInstanceDefinitionReference -Name "name" -InstanceDefinition $instanceDef -
InstanceDefinitionItem $instanceDefItem
```

- Added Remove-RpsInstanceDefinitionAssociation. Sample:

```
$instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
$instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
$instanceDefItem2 = Get-RpsInstanceDefinitionItem -Name "MyItem2"
Remove-RpsInstanceDefinitionAssociation-InstanceDefinition $instanceDef -PrimaryReference
$instanceDefItem -Secondaryreference $instanceDefItem2
```

  - Added New-RpsInstanceDefinitionAssociation. Sample:

    ```
    $instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
    $instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
    $instanceDefItem2 = Get-RpsInstanceDefinitionItem -Name "MyItem2"
    New-RpsInstanceDefinitionAssociation-InstanceDefinition $instanceDef -PrimaryReference
    $instanceDefItem -Secondaryreference $instanceDefItem2
    ```

  - Added Get-RpsInstanceDefinitionAssociation. Sample:

    ```
    $instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
    $instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
    $instanceDefItem2 = Get-RpsInstanceDefinitionItem -Name "MyItem2"
    Get-RpsInstanceDefinitionAssociation-InstanceDefinition $instanceDef -PrimaryReference
    $instanceDefItem -Secondaryreference $instanceDefItem2
    ```

- Added New-InstanceDefinitionNode. Sample:

```
New-RpsInstanceDefinitionNode -EntityName testEntityName -Name name2 -Hostname hostname -IPAddress
1.1.1.1 -SyncEndpointUrl syncEndpoint -certificateThumbprint certThumbprint -pollingInterval 1
```

- Added Set-InstanceDefinitionNode. Sample:
  `PowerShell Set-RpsInstanceDefinitionNode -Name name1 -EntityName testEntityName2 -Hostname hostname2 -IPAddress 2.2.2.2 -SyncEndpointUrl syncEndpoint2 -certificateThumbprint certThumbprint2 -pollingInterval 2`
- Added Get-InstanceDefinitionNode. Sample:
  `PowerShell $instanceDefNode = Get-RpsInstanceDefinitionNode -Name name1`
- Added Remove-InstanceDefinitionNode. Sample:
  `PowerShell Remove-RpsInstanceDefinitionNode -Id "8825A09C-CCE3-4BB0-BCE1-03B4729AC423" Remove-RpsInstanceDefinitionNode -InstanceDefDefinitionNode $InstanceDefinitionNode`

- Updated demo data scaffolding to support DCA image testing.

  - Added NOP79190 node to Nodes.psd1 to support DCA image testing.
  - Added NOP79190 target item to TargetItems folder to support DCA testing.
  - Updated Assignments.psd1, ResourceGroups.psd1, and Initialize-Image.ps1 to support DCA image testing.

- Updated demo data scaffolding to support NDM image testing.

  - Updated TCN79192 demo data scaffolding to support NDM image testing.
  - Added TCN79192 node to Nodes.psd1 to support NDM testing.
  - Added TCN79192 target item to TargetItems folder to support NDM testing.
  - Updated Assignments.psd1, ResourceGroups.psd1, and Initialize-Image.ps1 for NDM image testing.

- Modified Initialize-Baseline.ps1 to support dynamic testing of images.

- Updated PartialConfigurations-cmdb.tests.ps1 to support dynamic testing of images. Now includes the DCA and NDM image.

Admin UI

- Added Generate Random Password functionality for Resource Item, Target Item, and Patch Password fields.
- Added the ability for Generate Random Password to be based on a Password Policy.
- Modified the UI to mask protected properties.
- Added password/protected property reveal functionality to the UI.
- Added the ability to supply a certificate thumbprint for encrypting/decrypting CMDB file export/import via the UI.
- Updated the Task Map Step Number and Depends On columns, so it has a consistent sort order.

Resolved Issues

Top issues addressed in 3.0:

- Fixed issue in the the RPS Install that was causing DSC to fail on Node Registration and import.
- Fixed issue with SID translation that would force manual intervention.
- Fixed issue where user rights assignment settings within the core repo were conflicting when STIGs were applied.

Known Issues

Issues Addressed in RPS release 2.4.6

- #21479 LCM Configuration Mode is not controllable per target from the CMDB
- #21445 Missing utilities folder in local content store
- #21422 RpsProvisioning cannot configure virtual drive
- #21557 Rps-encryption breaking SAN's
- #21558 Rps-Network doesnt allow existing exclusion assignments
- #21559 RpsDomainController only applies tombstone lifetime to primary dc
- #20900 2Q19 User Principal Name suffix isn't being configured on DSC VM or AD.rps.local
- #21469 Access Database partial is assigned in colorless baseline data for RVP
- #21483 Remove RVPS GUI files and install and powerstig from the release
- #21538 Prov Vlan updates from GDMS
- #21539 TaskMap Updates from GDMS
- #21566 Runbook retries implemented where network communications can be a factor
- #21536 GPO Updates to UnifiedAD
- #20788 Duplicate Import-RPSNode Functions
- #21472 OCSP website has request filters that need to be removed
- #20670 PowerSTIG Service rules fail if the expected service does not exist

Issues Addressed in RPS release 2.4.5

- #21275 Cannot add patches to a target and republish
- #21396 RVP computer account on Dev node deployment not added to correct OU
- #21371 OsCore does not create a Disk resource for targets with multiple disks
- #20952 2Q19 Update to only install McAfee agent 5.5
- #21269 Publish-DSCPatch.ps1 pathing bug breaks Patching
- #20951 2Q19 Update VMware-tools-10.3.10-12406962
- #16291 RVP - CdnPath points to C$ instead of share
- #20897 RVP missing ImagesParentPath property
- #20867 New-ProvisioningNodeConfiguration.ps1 has incorrect property name images parent folder property

- #20875 Unneeded array item causes a duplicate resource ID error during compilation
- #20674 Failing resource on DSC - [xADForestProperties]
- #20878 RpsSQL and RpsSMA resource controller has incorrect import version
- #17860 On a running Prov Laptop APP VM, when a taskmap assignment and RunOnLocalNode has been issued, the first 3 workflows fail
- #21354 Get-DSCStatus is not assigned to targets therefore patch status is not updated
- #20897 RVP missing ImagesParentPath property
- #20928 2Q19 Utilities and Certificates not present in localContent Store Path
- #20931 Created Dev Enclave/Node to reduce the complexity of deployments across teams
- #20932 Remove hardcoded data from installer. Modify Hyper-V VM creation scripts to ensure VM environments are generated as specified
- #20948 2Q19 Update Adobe Reader to 19.012.20034
- #20950 2Q19 Update Java\jre-8u212-windows-x64.exe
- #21395 OcspResponder partial is skipped on RVP due to missing property
- #21383 RPSOSCore.ps1 network profile configuration can only set one interface to Private, otherwise there are resource conflicts
- #21006 2Q19 Certificate generation creates malformed SANs during import
- #20928 2Q19 Utilities and Certificates not present in localContent Store Path

Issues Addressed in RPS release 2.4.4

- 19832 Copy-ContentStore does not log an alert when a file copy fails
- 19666 ResourceGroups.psd1 for the DSC missing items
- 20757 Adobe Acrobat version needs updated in partial
- 20559 Firewall Rules only allow traffic for specific applications
- 20603 GPO SIDs are not being translated into domain accounts when imported
- 20607 TrustElementRepository Reader/Writer sites have incorrect bindings
- 20690 Failing resource on RVP - [xPackage]McAfee Agent
- 19528 Test-DscMof does not detect resource conflicts between PowerStigConfiguration and other partials
- 20605 TrustElementRootPath gets set to wrong path
- 20611 Failing resource on RVP - [AdcsOnlineResponder]OnlineResponder
- 20776 RVP data has assigned partials that should not be assigned
- 19832 Copy-ContentStore does not log an alert when a file copy fails
- 19661 Set contentfreshness for sysvol replication on all domain controllers to 365 days maxtimeofflineindays setting
- 20604 Conflicting ComputerManagementDsc module versions
- 20821 SMA Runbook account needs to have LogonAsAService permissions
- 20607 TrustElementRepository Reader/Writer sites have incorrect bindings
- 19528 Test-DscMof does not detect resource conflicts between PowerStigConfiguration and other partials
- 20605 TrustElementRootPath gets set to wrong path
- Update to PowerSTIG 3.2.0
- STIG Rule Updates: 41023, 41024, 4102, 41026, 41407, 41021, 41022, 41027 41028, 41029, 41030, 41031, 41032, 41033, 41035, 41042 41305, 41306, 41307, 41037, V-41251, V-40950, V-69169 V-40952, V-40953, V-41016, V-41017

Issues Addressed in RPS release 2.4.3

> Released on May 15, 2019

- Removed dependency on xCAPIstore resources
- Removed unneeded service restart in domain controller resource
- Updated Install-MNRps so execution can occur without $VhdFolderPath and $VMTemplateFileName
- Separated reader and writer SSL settings for the Trusted Element Reader website
- Fixed duplicate resources being created between PowerStig and RPS partials

- The $allComputers variable in Rps-Installer module was not properly populated and resulted in unexpected deployment
- Fixed issues where DNS Zones were not loaded; the same fix addresses 'Domain Controller promotion fails due to unknown root cause; possible SMB contention issue' and 'replica DC promo fails multiple connection issue'
- Added an array for value data on IPv6 disable resource
- Fixed Master-Controller failing to resume after service or machine restart
- Addressed xWebAdministration version references mismatch
- Added a reboot for ssl binding registry update to address Registry resource failing to add registry keys for TER
- Fixed inbound Reader and Writer traffic being blocked to the RVP
- Fixed the issue where PFX files were attempting to get uploaded to the TER site
- Removal of ResourceControllerDSC module version 1.3.1 and added ResourceControllerDSC module version 2.0.0.
- Updated version for import-module calls for ResourceControllerDSC
- Added use of ResourceController for Allow Log on Locally, and Log on as a Service URAs within RpsSync partial.
- Within the RpsSecIIS partial, added three resources that leverage ResourceControllerDSC to remove .NET v.4.5, and .NET v.4.5 Classic accounts from Log on as a service, Generate security audits, and replace a process level token before they become rogue/dead sids.
- COTS Update - McAfee
  - Agent 5.5.1.462
  - ACCM 3.2.5
  - RSD 5.0.6.125
  - SIEM Collector **new**
- COTS Update - ActivClient 7.1
- COTS Update - Adobe Reader 19.10.20091.53467

Issues Addressed in Core release 2.4.2

> Released on April 17, 2019

- Added and updated tombstone parameter
- Added forest name to domain object for laptop build
- User and Group property updates
- Add MC check to ensure only one MC is running
- Add max reserved memory for SQL
- Added RSAT for DNS
- Add tombstone configuration

Issues Addressed in DSC_Images tagged 2.4.2

- Changes made to partial to reflect the most current VMWare tools software
- Integrate PowerStig 3.1
- Bug fix to allow for duplicate name
- Add a forest name property to the ADDomain object
- Add valid task map action
- Updating the NT Auth store can fail - this breaks CAC login
- Updated NPS partial to use NPS group configuration from CMDB
- Update tombstone value on domain controller
- Added property for max memory

Issues Addressed in 2.4.1

> Released on April 2, 2019

- partial update for the gpomanagementdsc module update in DSC_Images
- adding UPN Suffix to adobjects
- Update RpsDomainController.ps1
- disable ipv6
- Using Registry instead of xRegistry

What's New in 2.4

> Released on January 29, 2019

PowerShell

This release contains the following PowerShell enhancements:

- The RPS Installer was updated to support complex task map execution in order to provide the ability to create ESXi, VMware, or Hyper-V based hosts and virtual machines.
- Added support for ESXi Host and virtual machine configurations.
- Improved Installer's ability to generate representative XML for RPS Import by reducing the number of switches required during the installation/configuration.
- Reorganized RPS PowerShell Modules into:

MODULE	DESCRIPTION
Rps-Api	Core API functions
Rps-Credential	Create and access credentials in RPS CMDB
Rps-Dsc	Utility to help publish, manage and test RPS DSC Partials
Rps-Encryption	Manage certificates and encryption
Rps-Installer	RPS Configuration, Data Import and Installation helpers
Rps-IpSheet	Import networking information from an IPSheet Excel document
Rps-Network	Network Utilities
Rps-Snmp	Communicate with network switches
Rps-Types	Create and manage RPS Type Definitions
Rps-Utilities	Additional Utilities
Rps-Virtualization	Management of Virtualization

- Refactored New-HypervVirtualMachine to support additional configuration options. The new runbook is now called Set-HyperVVirtualMachine. Enhancements include support for the following:
  - Generation 1 virtual machines
  - Vhd disks
  - All virtual switch types (Internal, External, Private)
  - N number of disk/dvd drives and nics (Up to Hyper-V limitations)

- Processor configuration
- Static/Dynamic memory configuration
- Image from .iso, differencing disk, existing disk

- Virtual network adapter IP address configuration, including VLAN tagging

  In order to take advantage of all these configurable options, the data must be representative of the configuration that is desired. Below is a representation of the relationship within the Rps type definitions:

OBJECT	RPS ENTITY TYPE	RPS TYPE	RPS SUBTYPE	PARENT OBJECT	ASSIGNMENT
Host	Resource/Target	Host	HyperV	N/A	VirtualMachine
Virtual Machine	Target	VirtualMachine	N/A	N/A	Host
Virtual NIC	Target	NIC	VirtualMachine	N/A	VirtualSwitch
VHD(X)	Target	Drive	Disk	VirtualMachine	N/A
Dvd	Target	Drive	DVD	VirtualMachine	N/A
Processor	Target	Processor	N/A	VirtualMachine	N/A
Virtual Switch	Resource	VirtualSwitch	HyperV	N/A	NIC

  To see the configurable properties on each of these objects, please reference the Rps type definitions located at "ContentStore\Setup\Configuration\Import-RpsTypes.ps1".

  Sample configurations are located at "ContentStore\Demos\Set-HyperVVirtualMachine".

DSC

This release contains the following DSC enhancements:

- Added support for multi-step software installs to the Software Distribution Partial.
- Updated Runbook Guidance based on lessons learned from ESXi and SNE MVP.

- Added support for additional DHCP configuration options in the RpsDhcp partial such as:
  - Scope option definitions
  - Scope definitions
  - Exclusion ranges
  - Server bindings

- Updated Dsc Modules to the following versions:

MODULE	CORE VERSION
ComputerManagementDsc	6.0.0.0
ResourceControllerDSC	1.3.1.0
SqlServerDsc	12.1.0.0
xActiveDirectory	2.22.0.0

MODULE	CORE VERSION
xHyper-V	3.13.0.0
xWebAdministration	2.3.0.0

- Added support for PKI functionlity to support DCA image with new DSC resource MN_ActiveDirectoryCSDsc (Forked from ActiveDirectoryCSDsc 3.1.0.0). New resource include:
  - AdcsAiaExtension
  - AdcsCdpExtension
  - AdcsCertificateTemplate
  - AdcsImportCrl
  - AdcsInstallCertificate
  - AdcsOcspExtension
  - AdcsPublishCert
  - AdcsPublishCrl
- Added support for GPO Management functionlity to support NDM image with DSC resource MN_GpoManagementDsc. New resource include:
  - GpSecurityFilter

RPS API

This release contains the following API enhancements:

- Added support for structured logging during unattended RPS Installer executions.
- Updated the RPS API to optimize Target loading with several Task Map Assignments.
- Resource Items and Resource Assignments can now be retrieved by Role, which is a special property designated for tracking the purpose of a resource item or its relationship to a target item. The Role property can be placed on a Resource Item or the Resource Assignment and can hold multiple values separated by the `|` symbol. To get resource items that have a specific role or have an assignment with a specific role, use the `-MatchAssignmentRole` parameter.

  Sample:

  ```
      $clientAuthCerts = Get-RpsResourceItem -Type Certificate -Role "ClientAuth"
      $localAdmins = Get-RpsResourceItem -TargetItem $computer -Type Credential -Role "LocalAdministrator"
  -MatchAssignmentRole
  ```

  Sample:

  In this example, a Credential (Resource) is assigned to a Computer (Target). The assignment is given a Role of "LocalAdministrator". We can retrieve the designated Local Administrator credential for the computer by using the `-Role` parameter.

  ```
      # assign credential and set roles
      $computer = Get-RpsTargetItem -Type "Computer" -Name "Win137"
      $credential = Get-RpsResourceItem -Type "Credential" -Name "RpsAdministrator"
      $assignedCredential = New-RpsResourceAssignment -TargetItem $computer -ResourceItem $credential
      $assignedCredential.Role = "LocalAdministrator|RpsUser"
      $assignedCredential.Update()

      # retrieve the LocalAdmin credential for the computer
      $localAdminAssignment = Get-RpsResourceAssignment -TargetItem $computer -Role "LocalAdministrator"
  ```

- Added the -Scope parameter on the New-RpsTaskStep cmdlet.

  Sample:

```
    New-RpsTaskItem -WorkflowName "Resolve-TargetMacAddress"
    New-RpsTaskItem -WorkflowName "Wait-TargetReady"
    New-RpsTaskItem -WorkflowName "Wait-TargetReady"
    New-RpsTaskItem -WorkflowName "Copy-BaseImages"
    New-RpsTaskItem -WorkflowName "New-VMWareVirtualMachine"
    New-RpsTaskItem -WorkflowName "Resolve-TargetDhcpIPAddress"

    # parameter setup
    $baremetalConfig = @{ TargetItemType = "Computer"; Filters = @{ IsHypervisor = "False" } }
    $esxConfig = @{ TargetItemType = "Computer"; Filters = @{ IsHypervisor = "True" } }
    $vmConfig = @{ TargetItemType = "VirtualMachine"; Filters = @{ "IsAppliance" = "False" } }
    $vmApplianceConfig = @{ TargetItemType = "VirtualMachine"; Filters = @{ "IsAppliance" = "True" } }
    $rvpConfig = @{
        TargetItemType = "VirtualMachine";
        Filters = @{ "IsAppliance" = "False"; "Designation" = "RVP" }
    }

    # Task Map creation
    $map = New-RpsTaskMap -Type "ProvisionSystemDemo" -Name "ProvisionSystemDemo"
    $mapConfig = @{ TaskMap = $map; AllowMultipleTargets = $true; IsTargetRequired = $true }

    # Adding steps to task map
    $resolveMac = New-RpsTaskMapStep @mapConfig -RunbookName "Resolve-TargetMacAddress" -TargetItemType
"Switch"
    $waitBaremetal = New-RpsTaskMapStep @mapConfig -RunbookName "Wait-TargetReady" -TargetItemType
"Computer"

    $baremetalHV1 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Copy-BaseImages" -Dependencies
$waitBaremetal @esxConfig
    $baremetalVM1 = New-RpsTaskMapStep @mapConfig -RunbookName "New-VMWareVirtualMachine" -Dependencies
$baremetalHV1 @vmConfig

    #Adding step with dependecy
    $baremetalVM2 = New-RpsTaskMapStep @mapConfig -RunbookName "Resolve-TargetDhcpIPAddress" @vmConfig -
Dependencies $baremetalVM1 -Scope Self
```

Admin UI

This release contains the following Admin UI enhancements:

- Added LocalNode UI option on execution of Task Map Assignment.
- Defaulted Resource Assignment state to Ready where no approval action needs to take place.
- Calculated the file hash of an imported file on Import.
- Corrected Pending Task Individual Count.
- Removed the Pending Actions on targeting list views.
- Made the Active and Global flags display consistently throughout the UI.
- Replaced the Edit and Remove hyperlinks with command buttons on the TaskMap derail views.
- Updated the UI to optimize loading a Target Item's details when several Task Map Assignments exist.
- Combined Pending Tasks and Task Information sections into one Job section on the Target Details view.

Resolved Issues

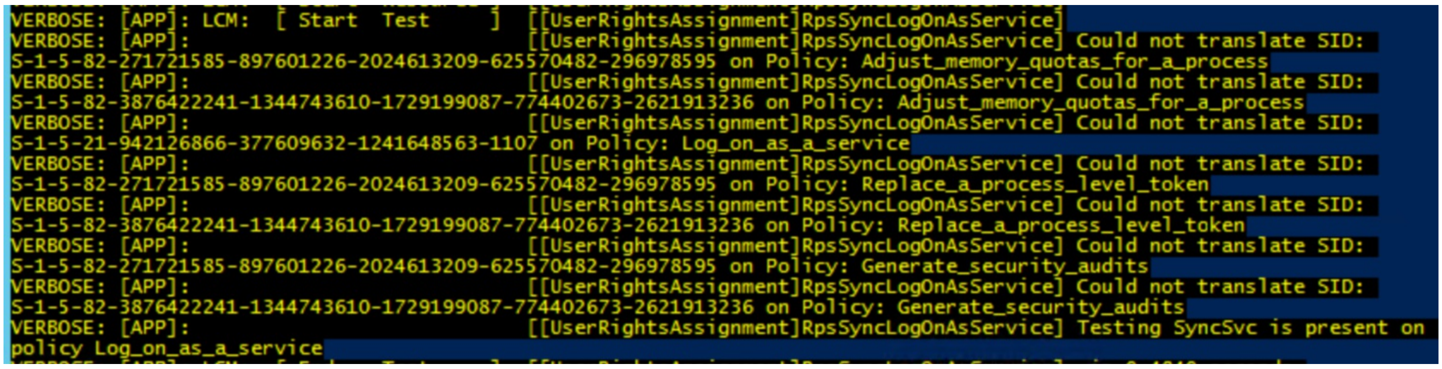
Top issues addressed in 2.4:

- Resource Groups fails to import when group references already exist.
- Provisioning Service returns a 500 error if a duplicate object is found.
- Calling Set-RpsResourceItem and/or Set-RpsTargetItem with null properties causes a null-reference exception.
- Set-RpsResourceItem does not update parent's state when adding children.

- Import TaskMap with non-default dependency when scope is ignored.
- IpSheet import fails due to missing Access Database Engine pre-requisite.
- Task Assignment History not saved while in Session.
- RPS Session failing to refresh deleted Task Assignments from Target Item.
- Pending Task Individual Count is incorrect in the Target Item Detail view.
- Modified the Wait-TargetReady runbook to support both PhysicalMachine and Computer Types.
- The Installer's -GenerateXmlOnly switch fails to generate usable file when -ConfigFilename specified.
- Added the ability for more than one process to access isolated storage at the same time.
- Exported data doesn't include the Task Map Assignment if assigned to Child Item.
- Task Map Step Dependency scope is not imported.
- Added a fix for Installer when Script fails to fully execute when not running elevated.
- Inception deployment fails with an HttpSetServiceConfiguration error.
- Import-RpsIpsheet on TestIpSheet takes too long.
- Encrypted Dsc partials fail to decrypt when a partial without a credential is applied first.

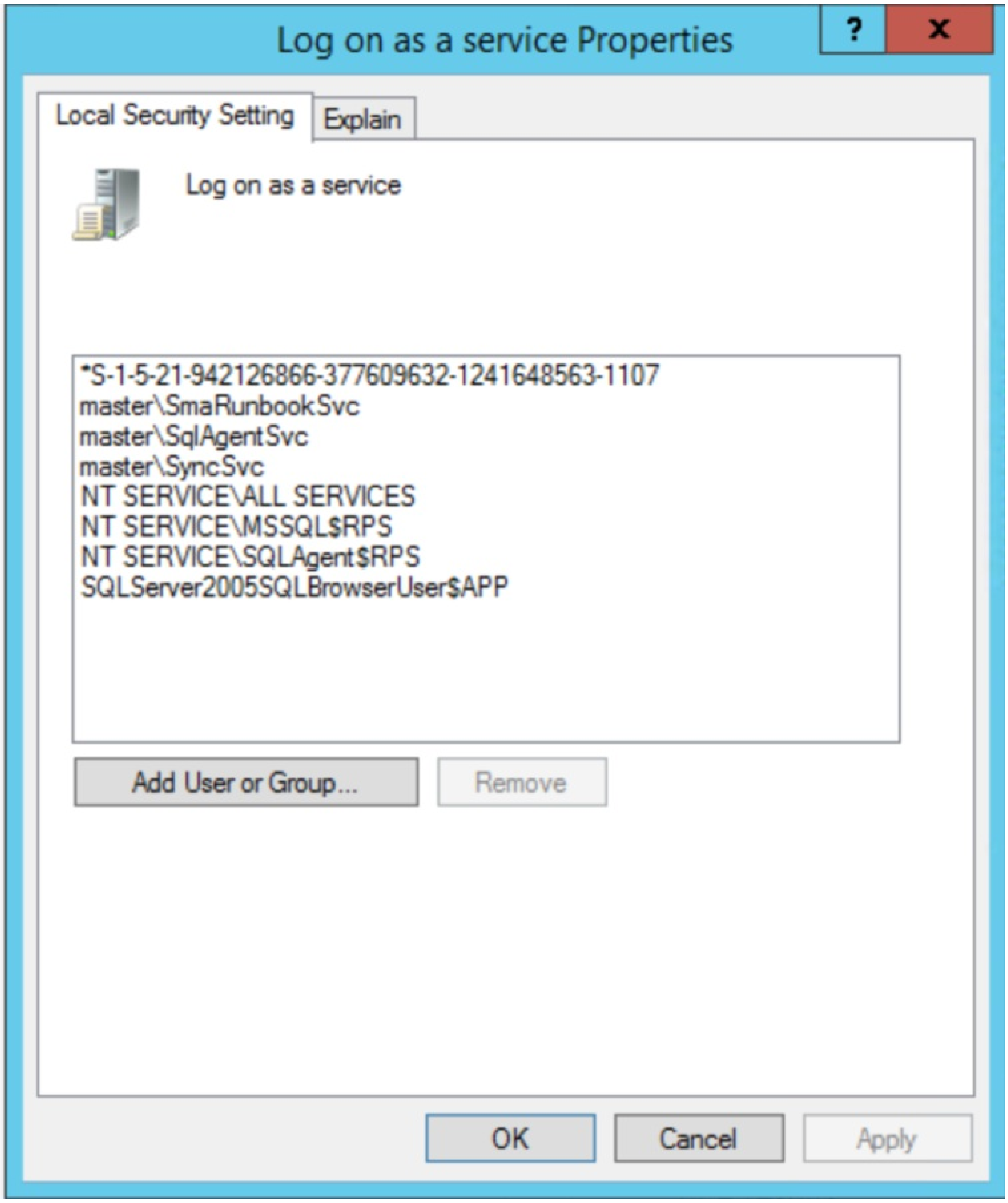
Known Issues

- RPS Install isn't exporting Host Node info, causing DSC to fail on Node Registration and import.
- UserRightsAssignment Dsc resource can sometimes fail due to failure to translate SIDs. See here for more information on the details. You can see this error exposed in Dsc:



```
VERBOSE: [APP]: LCM:  [ Start  Test     ]  [[UserRightsAssignment]RpsSyncLogOnAsService]
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-271721585-897601226-2024613209-625570482-296978595 on Policy: Adjust_memory_quotas_for_a_process
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-3876422241-1344743610-1729199087-774402673-2621913236 on Policy: Adjust_memory_quotas_for_a_process
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-21-942126866-377609632-1241648563-1107 on Policy: Log_on_as_a_service
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-271721585-897601226-2024613209-625570482-296978595 on Policy: Replace_a_process_level_token
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-3876422241-1344743610-1729199087-774402673-2621913236 on Policy: Replace_a_process_level_token
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-271721585-897601226-2024613209-625570482-296978595 on Policy: Generate_security_audits
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-3876422241-1344743610-1729199087-774402673-2621913236 on Policy: Generate_security_audits
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Testing SyncSvc is present on
policy Log_on_as_a_service
```

A workaround for this is to open Secpol.msc and remove any untranslated SID's for the targeted user right:



```
*S-1-5-21-942126866-377609632-1241648563-1107
master\SmaRunbookSvc
master\SqlAgentSvc
master\SyncSvc
NT SERVICE\ALL SERVICES
NT SERVICE\MSSQL$RPS
NT SERVICE\SQLAgent$RPS
SQLServer2005SQLBrowserUser$APP
```

What's New in 2.3

> Released on October 30, 2018

PowerShell

This release contains the following PowerShell enhancements:

- Master-Controller now has the ability to run recurring tasks and scheduled tasks.
- The Get-DscStatus runbook will now by default run every two hours.
- Virtual disk file locations will use the Hyper-V default filepath when creating a virtual machine. You can also optionally specify an alternate location to store the vhdx.

- ServerAdmin role created for all administrative functions required by Rps. It previously required the DomainAdmin role.
- Installer can dynamically generate self-signed certificates per deployment. It will use the configuration data supplied to populate their properties. Can also supply your own certificates. See the *Certificate Usage* document for details.
- Added the capability to suppress reboots for individual software installs.
- Reorganized RPS PowerShell Modules into:

MODULE	DESCRIPTION
Rps-Api	Core API functions
Rps-Credential	Create and access credentials in RPS CMDB
Rps-Dsc	Utility to help publish, manage and test RPS DSC Partials
Rps-Encryption	Manage certificates and encryption
Rps-Installer	RPS Configuration, Data Import and Installation helpers
Rps-IpSheet	Import networking information from an IPSheet Excel document
Rps-Snmp	Communicate with network switches
Rps-Types	Create and manage RPS Type Definitions
Rps-Utilities	Additional Utilities

DSC

This release contains the following DSC enhancements:

- Created/Updated DSC Partial Configurations to support cross-forest configurations, Provisioning Service, CDN Service
- Updated Dsc Modules to the following versions:

MODULE	CORE VERSION
AccessControlDsc	1.1.0.0
CertificateDsc	4.4.0.0
ComputerManagementDsc	5.2.0.0
NetworkingDsc	6.1.0.0
SecurityPolicyDsc	2.4.0.0
SqlServerDsc	11.4.0.0
xActiveDirectory	2.21.0.0
xDatabase	1.9.0.0
xDhcpServer	2.0.0.0

MODULE	CORE VERSION
xDnsServer	1.11.0.0
xPSDesiredStateConfiguration	8.3.0.0
xSmbShare	2.1.0.0
xWebAdministration	2.2.0.0
xWindowsUpdate	2.7.0.0

RPS API

This release contains the following API enhancements:

- The Task assignment restrictions were relaxed so that a Task Map can be assigned to non-root Target items. The New-RpsTaskAssignment Cmdlet previously restricted an assignment to only root-level target items. However, the restriction is no longer applicable within vehicle provisioning scenarios, where the vehicle is the root, DCEs are child items and virtual machines are grandchildren targets.
- Updated the Task Map Dependency scope to allow defining dependencies scoped to "all" (target), "self", and "parent". This provides the capability to have DCE1 tasks run parallel to DCE2 tasks, given that the DCEs are both children of a SNE parent.
- Simplified Task Map creation by creating the New-RpsTaskMapStep Cmdlet. The Cmdlet may accept a runbook name parameter instead of "TaskItem", essentially eliminating the need for using the existing Task Map structure. In addition, the New-RpsTaskMapStep Cmdlet will accept filters and dependencies inline.

- The New-TaskMapDefinition, New-TaskMapDefFilter, and New-TaskMapDefDependency Cmdlets are marked as obsolete and have been replaced by the New-RpsTaskMapStep, New-RpsTaskMapStepFilter, and New-RpsTaskMapStepDependency Cmdlets respectively.

  Sample:

```
# Create target items
$sne1 = New-RpsTargetItem -Type Vehicle -Name SNE1
$switch1 = New-RpsTargetItem -Type Switch -Name "Cisco Switch" -ParentItem $sne1
$dce1 = New-RpsTargetItem -Type DCE -Name "DCE 1" -ParentItem $sne1
$dce2 = New-RpsTargetItem -Type DCE -Name "DCE 2" -ParentItem $sne1
$dce3 = New-RpsTargetItem -Type DCE -Name "DCE 3" -ParentItem $sne1
$rvpVM = New-RpsTargetItem -Type VM -Name "RVP" -ParentItem $dce2

# Create tasks
$task1 = New-RpsTaskItem -WorkflowName "Wait-Switch"
$task2 = New-RpsTaskItem -WorkflowName "Wait-DCE"
$task3 = New-RpsTaskItem -WorkflowName "Set-DCEConfig"
$task4 = New-RpsTaskItem -WorkflowName "New-ESXIVM"
$task5 = New-RpsTaskItem -WorkflowName "Publish-Dsc"

# Create task map
$map = New-RpsTaskMap -Type "Provision-Vehicle" -Name "Provision-SNE"
$step1 = New-RpsTaskMapStep -TaskMap $map -TaskItem $task1 -TargetItemType Switch
$step2 = New-RpsTaskMapStep -TaskMap $map -TaskItem $task2 -TargetItemType DCE -Dependencies $step1
$step3 = New-RpsTaskMapStep -TaskMap $map -TaskItem $task3 -TargetItemType DCE
New-RpsTaskMapStepDependency -PreviousStep $step2 -Step $step3 -Scope Self
$step4 = New-RpsTaskMapStep -TaskMap $map -TaskItem $task4 -TargetItemType VM
New-RpsTaskMapStepDependency -PreviousStep $step3 -Step $step4 -Scope Parent
$step5 = New-RpsTaskMapStep -TaskMap $map -TaskItem $task5 -TargetItemType VM
New-RpsTaskMapStepDependency -PreviousStep $step4 -Step $step5 -Scope Self

# Assign map
New-RpsTaskAssignment -TaskMap $map -TargetItem $sne1
```

Sample: Inline filters and dependencies

```
$byFilter = New-RpsTaskMapStep -TaskMap $map -Filters @{ Type = "VirtualMachine"; IsDsc = $true }
$withDependencies = New-RpsTaskMapStep -TaskMap $map -Dependencies @( $step1, $step2 )
$byRunbookName = New-RpsTaskMapStep -TaskMap $map -RunbookName "Publish-Dsc"
```

- Added the ability to nest Resource Groups in order to enable RPS to model many complex scenarios such as AD Security groups.

  Sample:

```
# Define a new Type Definition with the IsGroupReference flag
Set-RpsResourceType -Name "ADGroup" -IsRoot -IsGroupReference

# Create AD Groups
$ADDomainUsersGroup = New-RpsResourceGroup -Type "ADGroup" -Name "All Domain Users"
$ADAdminGroup = New-RpsResourceGroup -Type "ADGroup" -Name "Domain Admins"
$ADDNSAdminGroup = New-RpsResourceGroup -Type "ADGroup" -Name "DNS Admins"
$ADDirectorsGroup = New-RpsResourceGroup -Type "ADGroup" -Name "Directors"

# Create AD Users and asign them to groups
$AdUser1 = New-RpsResourceItem -Type "AdUser" -Name "AdUser1" -ResourceGroup $ADDomainUsersGroup -IsGlobal $true
$AdUser2 = New-RpsResourceItem -Type "AdUser" -Name "AdUser2" -ResourceGroup $ADAdminGroup -IsGlobal $true
$AdUser3 = New-RpsResourceItem -Type "AdUser" -Name "AdUser3" -ResourceGroup $ADAdminGroup -IsGlobal $true
$AdUser4 = New-RpsResourceItem -Type "AdUser" -Name "AdUser4" -ResourceGroup $ADDNSAdminGroup -IsGlobal $true
$AdUser5 = New-RpsResourceItem -Type "AdUser" -Name "AdUser5" -ResourceGroup $ADDirectorsGroup -IsGlobal $true
$AdUser6 = New-RpsResourceItem -Type "AdUser" -Name "AdUser6" -ResourceGroup $ADDirectorsGroup -IsGlobal $true

# Add AdUser3 to the AD Directors Group as well
$ADDirectorsGroup.AddChildItem($AdUser3)
$ADDirectorsGroup.Update()

# Get Group references
$ADAdminGroupRef = Get-RpsResourceItem -Id $ADAdminGroup.Id
$ADDNSAdminGroupRef = Get-RpsResourceItem -Id $ADDNSAdminGroup.Id
$ADDirectorsGroupRef = Get-RpsResourceItem -Id $ADDirectorsGroup.Id

# Assign Group references to All Domain Users Group
$ADDomainUsersGroup.AddChildItem($ADAdminGroupRef)
$ADDomainUsersGroup.AddChildItem($ADDNSAdminGroupRef)
$ADDomainUsersGroup.AddChildItem($ADDirectorsGroupRef)
$ADDomainUsersGroup.Update()
```

- Find-Rps* Cmdlets have been deprecated and renamed to Get-Rps* with the same functionality. The original Find cmdlets have been retained and marked obsolete. However, they will be removed in a future release.

- Added new Set-RpsTargetItem and Set-RpsResourceItem Cmdlets. Both Target and Resource Items can be created and edited via their respective Set-RpsTargetItem and Set-RpsResourceItem Cmdlets.

  Sample: Create a new target item via Set-RpsTargetItem

  ```
  $computer = Set-RpsTargetItem -Type "Computer" -Name "Win137" -ParentItem $serverRack
  ```

  Sample: Update an existing target item via Set-RpsTargetItem

  ```
  $computer = Set-RpsTargetItem -Type "Computer" -Name "Win137" -IsActive $false
  ```

  Sample: Create a new resource item via Set-RpsResourceItem

  ```
  $resourceItem = Set-RpsResourceItem -Type "type" -Name "name"
  ```

  Sample: Update an existing resource item via Set-RpsResourceItem

  ```
  $resourceItem = Set-RpsResourceItem -Type "type" -Name "name" -IsActive $false
  ```

- Updated the Target Type Definitions to include a child type for Actions. Actions link a Target of a certain type to a TaskMap.

This allows a user to easily determine the status of an Action via the Admin UI.

- Added support for the retrieval of Target items, Target groups, Resource items, and Resource groups via wildcard property filters. The Get-RpsTargetItem, Get-RpsTargetGroup, Get-RpsResourceItem, and Get-RpsResourceGroup Cmdlets will return target\resource items and target\resource groups respectively using the properties supplied. If no properties are supplied, all items\groups will be returned. When using the -Filter Parameter, a $null value may be passed as a wildcard.

  Sample: Get target items by properties

  ```
  $foundItem = Get-RpsTargetItem -Filter @{"MAC" = "00:11:22:33:44:55"}
  $foundAllItemsWithMACProperty = Get-RpsTargetItem -Filter @{"MAC" = $null}
  ```

- Modified the API to allow for duplicate Task Map assignments to be created. RPS prevented assigning a Task Map to the same Target item more than once. This restriction was a legacy component in order to prevent Task Maps from changing after they were assigned. However, many scenarios such as the patching and provisioning processes are required to be run multiple times. Allowing Task Maps to be run multiple times enables RPS to have a cleaner user interface, cleaner logic, and overall better response times.

- Added a new Get-RpsConstants Cmdlet that will return all the defined RPS constants.

  Sample:

  ```
  $rps = Get-RpsConstants
  ```

- Added Get-RpsInstanceDefinitionItem Cmdlet. Sample:

  ```
      Get-RpsInstanceDefinitionItem -Id "8825A09C-CCE3-4BB0-BCE1-03B4729AC423"
      Get-RpsInstanceDefinitionItem -Name MyInstanceDefItem
      Get-RpsInstanceDefinitionItem -ResourceItem $resourceItem -Filter $filterHashtable
  ```

Admin UI

This release contains the following Admin UI enhancements:

- Added the TaskMap dependency scope to the TaskMap detail page.
- Added the ability to navigate between nested Resource Groups within the user interface.
- Changed the user interface's default landing page to the local Node's detail page.
- Added a section to the Node's detail page to display the status of its child Target items.
- Added the display of associated Actions to the Target Item detail page. This allows a user to easily determine the status of an assigned TaskMap, such as "SNE Provisioning", and start the TaskMap if necessary via the Admin UI.
- The Folder detail page was modified to list the files contained within the CDN folder.
- Added bread crumbs to the user interface to simplify site navigation.
- Simplified the Target details page by modifying the view to present just the high-level processes that are running and to use drill-downs view to access the more detailed information.
- Added a new detail page for Task Map Assignment.
- Added a new detail page for Resource Assignment.
- Modified the Target Group page to allow for adding and removing Target items to and from a group.
- Added the ability to select from any Task Map or Task item when assigning a new task to a Target item.

RPS Sync Service

This release contains the following Sync Service enhancements:

- Separated process of requesting changes and sending changes, so a child node will not block operations on a parent node.
- Queue received changes on all nodes, so changes won't be re-transferred on merge errors.
- Added Snapshot Isolation to transactions to avoid inconsistent data when gathering changes.
- Audit fields have been added to CMDB objects for use by Sync processes. These will be used by API in 2.4.

RPS CDN

This release contains the following RPS Content Delivery Network (CDN) enhancements:

- CDN now uses Background Intelligent Transfer Service (BITS) to transfer files from Parent to Child CDN.
- CDN uses hierarchical topology, where child requests files from parent, instead of full mesh used by DFS-R.
- CDN includes a new Indexer Service which stores File and Folder information in the CMDB to reduce duplicate transfers.

Baremetal Provisioning Service

The RPS Provisioning Service is an HTTP-based Web API hosted in IIS for use in brokering information from the RPS CMDB to a pre-execution environment such as iPXE for installation of a defined image and configuration. For instance, iPXE can be configured to "point to" the Provisioning Service which will return a boot script file for the MAC address requested.

This release contains the new Baremetal Provisioning Service with the following features:

- Return iPxe boot scripts from an http/https service based on matching devices in the CMDB.
- Host full images (such as .iso, .wim) in the service for download from iPxe.
- Host ESXi Kickstart scripts for ESXi configuration support.
- Support approval of base image through Resource Assignments in CMDB.
- Avoid boot looping through a configurable iPxe expiration period.

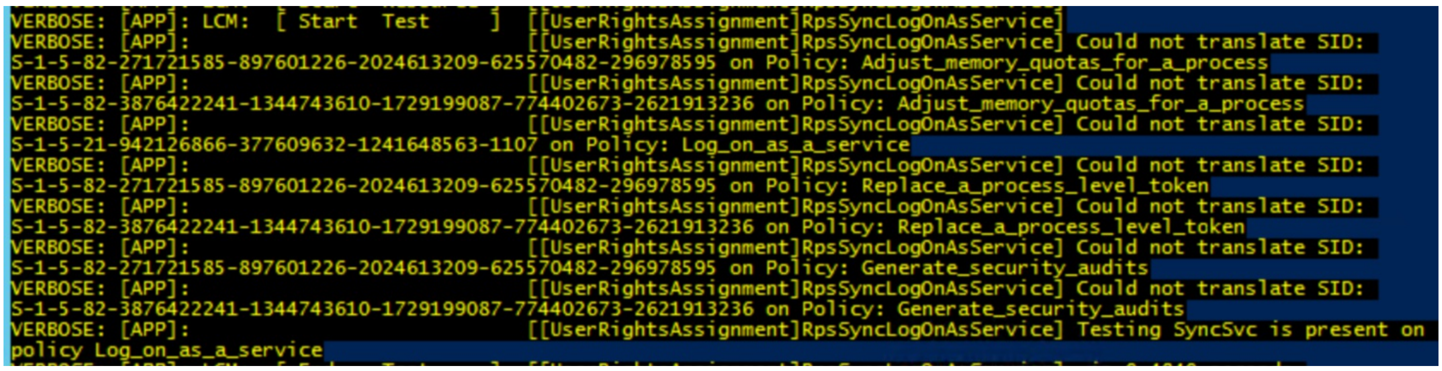
Resolved Issues

Top issues addressed in 2.3:

- RPS objects were not consistently setting dates\times to UTC dates\times.
- The DependsOn attribute was not handling all options.
- In RpsSession, the TaskAssignment Cmdlets attempted to transact with the Database.
- Internet Explorer failed to display glyph icons when using custom Cache-Control.
- The RpsGui Partial throws an error when applying SSL Certificate.
- In Server 2012, the New-HyperVVirtualMachine Runbook fails to add NICs to new VirtualMachines.
- Remove-ItemProperty fails in RpsSession.
- Test-DSCConfiguration returns false on CMDB deployment even after dacpac is deployed.
- Property Bag missing support for deleting a property.
- Target Item to GetResourceGroups and Resource to GetTargetGroups returns duplicate groups if group contains multiple members.
- Duplicate node error upon entering previously saved RpsSession.
- Calling Install-Rps for a specific node fails to create a parent node.
- Sync property replication failure.
- Sql Encryption fails to apply correctly.

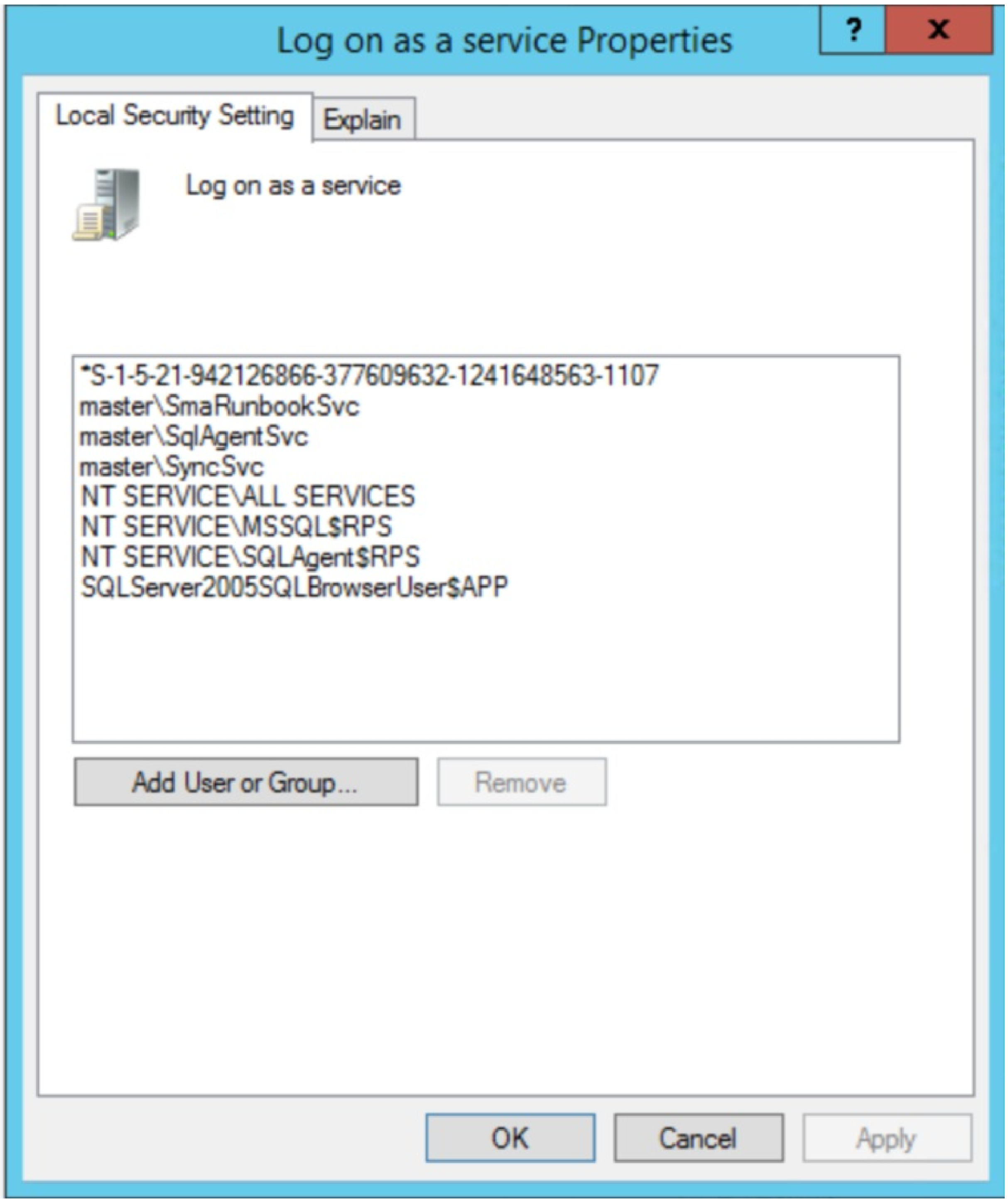
Known Issues

- UserRightsAssignment Dsc resource can sometimes fail due to failure to translate SIDs. See here for more information on the details. You can see this error exposed in Dsc:



```
VERBOSE: [APP]: LCM:  [ Start  Test     ]  [[UserRightsAssignment]RpsSyncLogOnAsService]
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-271721585-897601226-2024613209-625570482-296978595 on Policy: Adjust_memory_quotas_for_a_process
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-3876422241-1344743610-1729199087-774402673-2621913236 on Policy: Adjust_memory_quotas_for_a_process
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-21-942126866-377609632-1241648563-1107 on Policy: Log_on_as_a_service
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-271721585-897601226-2024613209-625570482-296978595 on Policy: Replace_a_process_level_token
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-3876422241-1344743610-1729199087-774402673-2621913236 on Policy: Replace_a_process_level_token
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-271721585-897601226-2024613209-625570482-296978595 on Policy: Generate_security_audits
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Could not translate SID:
S-1-5-82-3876422241-1344743610-1729199087-774402673-2621913236 on Policy: Generate_security_audits
VERBOSE: [APP]:                            [[UserRightsAssignment]RpsSyncLogOnAsService] Testing SyncSvc is present on
policy Log_on_as_a_service
```

A workaround for this is to open Secpol.msc and remove any untranslated SID's for the targeted user right:



```
*S-1-5-21-942126866-377609632-1241648563-1107
master\SmaRunbookSvc
master\SqlAgentSvc
master\SyncSvc
NT SERVICE\ALL SERVICES
NT SERVICE\MSSQL$RPS
NT SERVICE\SQLAgent$RPS
SQLServer2005SQLBrowserUser$APP
```

Using the RPS API

NOTE

Find-Rps* Cmdlets have been deprecated and renamed to Get-Rps* with the same functionality, Find verb will be removed in a feature release.

Target Items

Create Target Items

TIP

Target Items can also be created via Set-RpsTargetItem.

Create a new target item

```
$computer = New-RpsTargetItem -Type "Computer" -Name "Win137"
```

Create a new target item child

Specify the parent item when creating a child item. You can nest target items as deep as necessary.

```
$nic = New-RpsTargetItem -Type "NetworkInterface" -Name "137-NIC1" -ParentItem $computer
```

Create a new target item with Properties

Use a PowerShell Hashtable to pass properties when creating a new target item.

```
$platform = @{
    Architecture = "x64"
    OSVersion = "8.1"
    OSType = "Windows"
}
$computer = New-RpsTargetItem -Type "Computer" -Name "Win137" -Properties $platform
```

Create a target item for a different Node

By default, RPS will create new target items for the current node, but you can supply an alternative node.

```
$site2 = Get-RpsNode -Name "Site 2"
New-RpsTargetItem -Type "Computer" -Name "Win137" -Node $site2
```

Get Target Items

Use the `Get-RpsTargetItem` cmdlet to retrieve target items from RPS. Retrieve a single item using the `-Id` parameter, or retrieve multiple items using any combination of parameters.

Get target items by type

```
$allRouters = Get-RpsTargetItem -Type "Router"
```

Get target items by property value

Target Items can be filtered by a specific property or properties.

```
$deviceByMac = Get-RpsTargetItem -Filter @{ "MacAddress" = "00:11:22:33:44:55" }
```

Get target items by property

Target Items can be retrieved which have any value for a specific property. To retrieve items with a property, use `$null` for the value in the `-Filter` parameter.

```
$allDevicesWithMac = Get-RpsTargetItem -Filter @{ "MacAddress" = $null }
```

Get target items by Node

Target items can be retrieved for a specific node.

> **TIP**
>
> The Node parameter is currently limited to just selecting **root** target items within the specified node. A future update may expand the search to **all** items within the Node, including child items.

```
$site2Computers = Get-RpsTargetItem -Node $site2 -Type "Computer"
```

Update Target Items

Update target items using the `Set-RpsTargetItem` cmdlet. The `Set-Rps*` cmdlets will also create items if they don't exist.

The standard way to update an item uses the item's logical identifier, which is Type and Name. To update an existing target item, supply the `-Type` and `-Name` parameters and any additional parameters you wish to change.

Update an existing target item

```
$computer = Set-RpsTargetItem -Type "Computer" -Name "Win137" -IsActive $false
```

Resource Items

Create Resource Items

Create a new resource item

```
$resourceItem = Set-RpsResourceItem -Type "type" -Name "name"
```

Update an existing resource item

```
$resourceItem = Set-RpsResourceItem -Type "type" -Name "name" -IsActive $false
```

Get Resource Items

Get all resource items

```
$allItems = Get-RpsResourceItem
```

Get resource items by properties

```
$foundItemsWithSpecificKBNumber = Get-RpsResourceItem -Filter @{"KbNumber" = "3135782"}
$foundAllItemsWithKBNumber = Get-RpsResourceItem -Filter @{"KbNumber" = $null}
```

Get resource items by role

Resource Items can be retrieved by Role, which is a special property designated for tracking the purpose of a resource item or it's relationship to a target item. To get resource items that have a specific role or have an assignment with a specific role, use the `-MatchAssignmentRole` parameter. See the section below on Resource Assignments for more info.

```
$clientAuthCerts = Get-RpsResourceItem -Type Certificate -Role "ClientAuth"
$localAdmins = Get-RpsResourceItem -TargetItem $computer -Type Credential -Role "LocalAdministrator" -
MatchAssignmentRole
```

Target / Resource Groups

Get target groups by name and properties : Sample 1

```
$foundGroupByName = Get-RpsTargetGroup -Name "Target Group Name"
$foundGroups = Get-RpsTargetGroup -Filter @{"Color" = "Yellow"; "Food" = "Apples"; }
$foundAllGroupsWithColor = Get-RpsTargetGroup -Filter @{"Color" = $null}
```

Get resource groups by Id. : Sample 1

```
$foundGroup = Get-RpsResourceGroup -Id $resourceGroup.Id
```

Get resource groups by properties. : Sample 2

```
 $foundGroups = Get-RpsResourceGroup -Filter @{"Color" = "Yellow"}
 $foundAllGroupsWithColor = Get-RpsResourceGroup -Filter @{"Color" = $null}
```

Add resource group

```
 $resourceGroup = New-RpsResourceGroup -Type "testGroupType" -Name "GroupName"
```

Add resource group with properties

```
 $resourceGroup = New-RpsResourceGroup -Type "testGroupType" -Name "GroupName" -Properties @{ "Prop1" =
'Prop1' }
```

Add Resource Item to Resource Group

There are two methods to add a Resource Item to a Resource Group. A Resource Item can be assigned to a Resource Group during its initialization. The Resource Group must exist prior to assigning a Resource Item to a Resource Group during its initialization, as shown in the first example.

Add Resource Item in Resource Group in initialization

```
$resourceItem = New-RpsResourceItem -Type "testType" -Name "TestResourceItem" -ResourceGroup
$TestResourceGroup
```

Additionally, if the Resource Item and the Resource Group already exist, the Resource Item can be added to the Resource Group, as shown in the next example.

Add Resource Item to Resource Group via API

```
$testResourceGroup = Get-RpsResourceGroup -Name "TestResourceGroup"
$testResourceItem = Get-RpsResourceItem -Name "TestResourceItem"
$testResourceGroup.AddChildItem($testResourceItem)
$testResourceGroup.Update()
```

Add Target Item to Target Group

There are two methods to add a Target Item to a Target Group. A Target Item can be assigned to a Target Group during its initialization. The Target Group must exist prior to assigning a Target Item to a Target Group during its initialization, as shown in the first example.

Add New Target Item to Target Group

```
$targetItem = New-RpsTargetItem -Type "Computer" -Name "Server 150" -TargetGroup $servers
```

Additionally, if the Target Item and the Target Group already exist, the Target Item can be added to the Target Group, as shown in the next example.

Add Target Item to Target Group via API

```
$testTargetGroup = Get-RpsTargetGroup -Name "TestTargetGroup"
$testTargetItem = Get-RpsTargetItem -Name "TestTargetItem"
$testTargetGroup.AddChildItem($testTargetItem)
$testTargetGroup.Update()
```

Resource Assignment

A Resource Assignment is the assignment of a specific Resource Item to a specific Target Item. The assocation between the two items can contain Properties, a Status, and other useful information for automations. For example, a resource assignment can track the assignment of a Software Package (Resource) to a Computer (Target). The status may be used to indicate if that software is approved, installed, or up to date.

For convenience, you can specify a resource group in order to quickly assign multiple resources to a target. You can also specify a target group, or both!

NOTE

As of Release 2.2, duplicate assignments are not allowed. While allowing the assignment to groups, we are enforcing constraints to prevent duplicatous Assignments.

Create Resource Assignments

Assign a Resource Item to Target Item

```
$assign = New-RpsResourceAssignment -ResourceItem $software -TargetItem $computer
```

Assign a Resource Item to Resource Source

```
$assign = New-RpsResourceAssignment -ResourceItem $resourceItem -TargetItem $targetItem -ResourceState
$approved
```

Assign multiple Resource Items to Target Items using Groups

```
$assignments = New-RpsResourceAssignment -ResourceGroup $securityHotfixes -TargetGroup $allComputers
```

Get Resource Assignments

Use the `Get-RpsResourceAssignment` cmdlet to retrieve assignments.

Get Assignments for a Target Item

```
$computer = Get-RpsTargetItem -Type "Computer" -Name "Win137"
$assignedCredentials = Get-RpsResourceAssignment -TargetItem $computer -Type $rps.ResourceTypes.Credential
```

Get Assignments by Role

Role is a special property used frequently in RPS to track the purpose of the resource item's association to a target item. The **Role** property can be placed on a Resource Item or the Resource Assignment, and can hold multiple values separated by the `|` symbol.

In this example, a Credential (Resource) is assigned to a Computer (Target). The assignment is given a Role of "LocalAdministrator". We can retrieve the designated Local Administrator credential for the computer by using the `-Role` parameter.

```
# assign credential and set roles
$computer = Get-RpsTargetItem -Type "Computer" -Name "Win137"
$credential = Get-RpsResourceItem -Type "Credential" -Name "RpsAdministrator"
$assignedCredential = New-RpsResourceAssignment -TargetItem $computer -ResourceItem $credential
$assignedCredential.Role = "LocalAdministrator|RpsUser"
$assignedCredential.Update()

# retrieve the LocalAdmin credential for the computer
$localAdminAssignment = Get-RpsResourceAssignment -TargetItem $computer -Role "LocalAdministrator"```
```

Update Resource Assignments

Like Task Assignments, Resource Assignments track the history of State changes. Update a resource assignment with the Update-RpsResourceAssignment cmdlet.

Update state of the patch assignment to Denied

The system default of a Resource Assignment is **Approved**. Here, we want to be able to set whether an administrator is going to **Approve** or **Deny** the patch. To manually update the Resource State for the patch, we run the following:

```
$assignment.ResourceState = "Denied"
Update-RpsResourceAssignment -ResourceAssignment $assignment
```

Create Patch Group

An updated feature for Release 2.2 is the ability to create Patch Groups. With Patch Groups, a user can add multiple children (members) to the group. This allows for the deployment of multiple patches, simultaneously.

Here, we are creating a new RPS Resource Group, called: `$patchGroup`.

```
# Creating patch group
$patchGroup = New-RpsResourceGroup -Type Patch -Name DemoPatches

# Add children to the patch group
$patchGroup.AddChildren($hotfixJuly, $patch2, $patch3)

# Run update on the patch group

$patchGroup.Update()
```

Now that we have created our patch group and added our 3 children, we can assign our updated Patch to the Target Item.

Assign group of patches to a laptop

Assign the updated Patch Group to the Target Item, `$demoLaptop`.

```
New-RpsResourceAssignment -ResourceGroup $patchGroup -TargetItem $demoLaptop
```

Protected Properties

A Protected Property is a property in any property list that is marked as protected. Once a property is marked as protected, the data is encrypted and the property is marked as protected. To create a Protected Property use the cmdlet Set-Rps ProtectedProperty described below. To retrieve the value from a Protected Property use the cmdlet Get-RpsProtectedProperty as described below.

Get a Protected Property

Retrive a protected property from a Rps item by passing in the item and name of the property. The cmdlet returns a SecureString.

```
$name = 'Property Name'
$secureResult = Get-RpsProtectedProperty -TargetItem $targetItem -Name $name
$secureResult = Get-RpsProtectedProperty -ResourceItem $resourceItem -Name $name
$secureResult = Get-RpsProtectedProperty -Node $node -Name $name
$secureResult = Get-RpsProtectedProperty -TaskMapAssignment $taskMapAssignment -Name $name
$secureResult = Get-RpsProtectedProperty -ResourceGroup $resourceGroup -Name $name
$secureResult = Get-RpsProtectedProperty -TargetGroup $targetGroup -Name $name

# return variable to plain text
$plainText = ConvertFrom-SecureString $secureResult
```

Set a Protected Property

To add or update a protected property to any Rps properties collection by passing in the item, name of the property, and a SecureString for the value. The cmdlet returns a true if successful.

```
#setup
$securePwd = ConvertTo-SecureString "Password" -AsPlainText -force

$result = Set-RpsProtectedProperty -TargetItem $targetItem -Name "Password" -Value $securePwd
Set-RpsProtectedProperty -TargetItem $targetItem -Name $name -Value $securePwd
Set-RpsProtectedProperty -ResourceItem $resourceItem -Name $name  -Value $securePwd
Set-RpsProtectedProperty -Node $node -Name $name -Value $securePwd
Set-RpsProtectedProperty -TaskMapAssignment $taskMapAssignment -Name $name -Value $securePwd
Set-RpsProtectedProperty -ResourceGroup $resourceGroup -Name $name -Value $securePwd
Set-RpsProtectedProperty -TargetGroup $targetGroup -Name $name -Value $securePwd
```

Password Policy

A Password Policy is a set of password guidelines for a particular system or group of systems. The policy is saved in the database as a Resource Item.

- The default minimum password length is 16 characters
- The default maximum password length is 64 characters
- A WhiteList, when specified, denotes the only characters allowed in a password
- A BlackList, when specified, denotes the only characters disallowed from the standard list, which can be found by executing the code below `PowerShell $charList = [Rps.Api.Utils.PasswordUtils]::DefaultPasswordCharacters`

Set Password Policy

The following code creates a password policy with a minimum and maximum length and saves that policy to the connected data store

```
$policy = Set-RpsPasswordPolicy -Name WindowsPasswordPolicy -MinLength 8 -MaxLength 16
```

Get Password Policy

The following code retrieves the policy that was previously created

```
$retrievedPolicy = Get-RpsPasswordPolicy -Name WindowsPasswordPolicy
```

Password

A Password is a secret string of characters used to gain access to something

Generate Password

A password can be generated using default parameters or by specifying a Password Policy to use. The password can be returned as either a plaintext or secure string by using the AsSecureString switch. Passwords are not persisted to the data store by default.

```
$newPassword = New-RpsPassword -AsSecureString
```

```
$policy = Set-RpsPasswordPolicy -MinLength 20 -MinUppers 5 -MinNumbers 3
$newPassword = New-RpsPassword -PasswordPolicy $policy
```

Instance Definition

Instance Definitions are pre-defined data sets that consist of various Type Definitions, Items, and Associations between Items that can be used to create complex concrete objects.

Create Instance Definition

Creates an Instance Definition by providing a Name, Properties, and a Parent Node

```
$nodeId = "81B8272D-B49C-4350-A8F4-ABBB9CE29C68"
$hs = @{
    Prop1 = "value1"
    Prop2 = "value2"
}
New-RpsInstanceDefinition -Name testName -Properties $hs -ParentNodeId $nodeId
```

Get Instance Definition

Retrieves an Instance Definition by ID or Name

```

Get-RpsInstanceDefinition -Id "8825A09C-CCE3-4BB0-BCE1-03B4729AC423"
Get-RpsInstanceDefinition -Name testName
```

Set Instance Definition

Creates or Updates Instance Definitions, associated Properties, and associated parent Node.

```
Set-RpsInstanceDefinition -Name $Name1
Set-RpsInstanceDefinition -Name $Name1 -Properties @{Prop1 = "Value1"}
Set-RpsInstanceDefinition -Name $Name1 -Properties @{Prop1 = "Value1"} -ParentNodeId $nodeId
```

Remove Instance Definition

Removes an Instance Definition by ID or by object

```
Remove-RpsInstanceDefinition -Id "8825A09C-CCE3-4BB0-BCE1-03B4729AC423"
Remove-RpsInstanceDefinition -InstanceDefDefinition $InstanceDefinition
```

Invoke an Instance Definition

Creates an instance from the definition using settings stored in a resource item.

```
Invoke-RpsInstanceDefItem -Settings $resourceItem -InstanceDef $instanceDefinition
```

Get Instance Definition Reference

Retrieves an Instance Definition Reference by providing an Instance Definition and an Instance Definition Item.

```
    $instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
    $instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
    $reference = Get-RpsInstanceDefinitionReference -Name "name" -InstanceDefinition $instanceDef -
InstanceDefinitionItem $instanceDefItem
```

New Instance Definition Reference

Creates a new Instance Definition Reference.

```
    $instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
    $instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
    $taskMapIDs = "5b8b0340-091f-4823-b2f9-de937b5b4114", "a83b5445-3cc0-433e-b5e0-0fcf70389988"
    $reference = New-RpsInstanceDefinitionReference -Name "name" -InstanceDefinition $instanceDef -
InstanceDefinitionItem $instanceDefItem -TaskMapIDs $taskMapIDs
```

Remove Instance Definition Reference

Removes an Instance Definition Reference by Instance Definition and Instance Definition Item.

```
    $instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
    $instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
    Remove-RpsInstanceDefinitionReference -Name "name" -InstanceDefinition $instanceDef -
InstanceDefinitionItem $instanceDefItem
```

Instance Definition Item

An Instance Definition Item is a part of an Instance Definition that can be used to create concrete items such as Target Items, Resource Items, etc.

Get Instance Definition Item

Retrieves an Instance Definition Item by ID, by Name, or by Resource Item.

```
Get-RpsInstanceDefinitionItem -Id "8825A09C-CCE3-4BB0-BCE1-03B4729AC423"
Get-RpsInstanceDefinitionItem -Name MyInstanceDefItem
Get-RpsInstanceDefinitionItem -ResourceItem $resourceItem -Filter $filterHashtable
```

Create Instance Definition Item

Creates a new Instance Definition Item by providing an Entity Name, Name, Properties, and Type Definition ID

```
New-RpsInstanceDefinitionItem -EntityName testEntityName -Name name2 -Properties @{Prop1 = "Value1"} -
TypeDefinitionId $typedefinition.id
```

Set Instance Definition Item

Creates or Updates Instance Definition Items and associated Properties.

```
Set-RpsInstanceDefinitionItem -Name $Name1 -TypeDefinitionId $id -EntityName $entityName
Set-RpsInstanceDefinitionItem -Name $Name1 -TypeDefinitionId $id -Properties @{Prop1 = "Value1"} -EntityName
$entityName
```

Remove Instance Definition Item

Removes an instance definition item by either its Id or the instance definition item

```
Remove-RpsInstanceDefinitionItem -Id "8825A09C-CCE3-4BB0-BCE1-03B4729AC423"
Remove-RpsInstanceDefinitionItem -InstanceDefinitionItem $InstanceDefinitionItem
```

Instance Definition Node

Create an Instance Definition Node

An Instance Definition Node is a wrapper for an RPS type and associated Properties.

```
New-RpsInstanceDefinitionNode -EntityName testEntityName -Name name2 -Hostname hostname -IPAddress 1.1.1.1 -
SyncEndpointUrl syncEndpoint -certificateThumbprint certThumbprint -pollingInterval 1
```

Set Instance Definition Node

Creates or updates an Instance Definition Node.

```
Set-RpsInstanceDefinitionNode -Name name1 -EntityName testEntityName2 -Hostname hostname2 -IPAddress 2.2.2.2
-SyncEndpointUrl syncEndpoint2 -certificateThumbprint certThumbprint2 -pollingInterval 2
```

Get Instance Definition Node

Get an Instance Definition Node by name.

```
Get-RpsInstanceDefinitionNode -Name name1
```

Remove Instance Definition Node

Removes an Instance Definition Node by ID or by object

```
Remove-RpsInstanceDefinitionNode -Id "8825A09C-CCE3-4BB0-BCE1-03B4729AC423"
Remove-RpsInstanceDefinitionNode -InstanceDefDefinitionNode $InstanceDefinitionNode
```

Remove Instance Definition Association

Removes an Instance Definition Association by ID or by object

```
$instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
$instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
$instanceDefItem2 = Get-RpsInstanceDefinitionItem -Name "MyItem2"
Remove-RpsInstanceDefinitionAssociation-InstanceDefinition $instanceDef -PrimaryReference $instanceDefItem -
Secondaryreference $instanceDefItem2
```

New Instance Definition Association

Creates an Instance Definition Association by ID or by object

```
$instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
$instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
$instanceDefItem2 = Get-RpsInstanceDefinitionItem -Name "MyItem2"
New-RpsInstanceDefinitionAssociation-InstanceDefinition $instanceDef -PrimaryReference $instanceDefItem -
Secondaryreference $instanceDefItem2
```

Get Instance Definition Association

Creates an Instance Definition Association by ID or by object

```
$instanceDef = Get-RpsInstanceDefinition -Name "MyDefinition"
$instanceDefItem = Get-RpsInstanceDefinitionItem -Name "MyItem"
$instanceDefItem2 = Get-RpsInstanceDefinitionItem -Name "MyItem2"
Get-RpsInstanceDefinitionAssociation-InstanceDefinition $instanceDef -PrimaryReference $instanceDefItem -
Secondaryreference $instanceDefItem2
```

Types

Set Target Type

To Create or Update a RPS Target Type.

```
Set-RpsTargetType -Name Computer -IsRoot
```

Set Resource Type

To Create or Update a RPS Resource Type.

```
Set-RpsResourceType -Name Host -IsRoot -EnableSubType
```

Set Sub Type

To Create or Update a RPS Sub Type.

```
Set-RpsSubType -Parent $patchDef -SubType 'CAB'
```

Set Child Type

To Create or Update a RPS Child Type.

```
 Set-RpsChildType -Parent $def -ChildType $Rps.TargetTypes.NIC
    -DisplayName 'Network Adapters' -IsRequired -AllowMultiples
```

Set Type Property

To Create or Update a RPS Type Propery template.

```
New-RpsTypeProperty -Parent $template -Name VmType -PropertyType Text -IsRequired -DefaultValue 2012R2
```

Set Type Resource Assignment

Creates or updates a Resource Assignment template.

```
Set-RpsTypeRA -Parent $template -ResourceType 'Host' -DisplayName 'Hypervisor Host' -IsRequired
```

Set Target Action

Associates an action (TaskMap Type) with a Target Type definition.

```
 Set-RpsTargetAction -Parent $template -TaskMapType 'Provision-Vehicle' -Description 'Provision SNE'
```

Set Resource Group Type

Creates or updates a Resource Group Type

```
 Set-RpsResourceGroupType -Name MyResourceGroupType -IsGroupReference
```

Set Target Group Type

Creates or updates a Target Group Type

```
 Set-RpsTargetGroupType -Name MyTargetGroupType -IsGroupReference
```

RPS Sample Scenario - Tourism

The Rapid Provisioning System is designed to provide a mobile, modular, and extensible automations framework that allows coded automations activities to be executed across a wide area network environment with limited connectivity and bandwidth. The breadth of customization options can sometimes be overwhelming, so this Sample Scenario has been designed as an example of what RPS can do even on a small scale. A working knowledge of RPS entities and architecture is recommended prior to reading this guide. It is recommended that this sample data NOT be imported into a production environment.

Sample Dataset

Tourism Sample Dataset

In this example, RPS is used to support the technical operations of a Grand Canyon Touring company, GC Tour Guides. GC Tour Guides must rely on satellite and line-of-sight networking solutions for guides on tour, due to the extreme geology of the terrain and lack of reliable network connectivity on tour routes. Guides are equipped with a handheld device that contains all the necessary information and applications to get through their tour routes, including maps, a GPS tracker, communications services such as email and instant messaging, and inventory management. Additionally, guides take a minimum of two pack animals – mostly donkeys – with them on the trails. Each donkey has a tracker attached with sensors that detect and report on many environmental and biological events or changes, such as GPS coordinates, current heading, humidity, and even health statistics of the donkey, such as skin temperature, heart rate, number of steps and so on.

GC Tour Guides has deployed RPS to automate configuration and data management between the guides' handhelds, the pack animals' trackers, and headquarters. Additionally, RPS is used to transfer patches and updated map files to these devices. Finally, GC Tour Guides defines and deploys the configuration of their devices through PowerShell Desired State Configuration.

The donkeys are grouped together in a Target Group (TG) called "Donkeys" with the type "Tracker". Each Donkey is defined as a Target Item (TI) within the Donkeys TG. Target Items can have several custom properties assigned, and in this example, we have defined Manufacturer and Model as custom TI properties. Another TG is created for the guides, with a guide TI named "Clarence". Handheld TIs are grouped in a TG called Handhelds.

Before a guide leaves on a tour, a TG is created with all the entities that are scheduled to go on the tour. In our example, the guide Clarence, two donkeys, and a handheld are scheduled for June 2017 tour within a TG called "Guide Tour June 2017". By grouping the entities within a TG, inventory can be tracked and alerts can be set to go off if a Donkey TI's GPS Coordinates drift too far from the Handheld TI's GPS Coordinates, for example.

Continuing, we group two Patches as Resource Items (RI) within a Resource Group (RG) called "TrackerPatches". These are patches that will apply to just Tracker TIs. Finally, the Patch RG is associated with the Donkeys TG using a Resource Assignment (RA), which creates a link between each patch RI and applicable donkey TI. Figure 1 illustrates the example dataset.

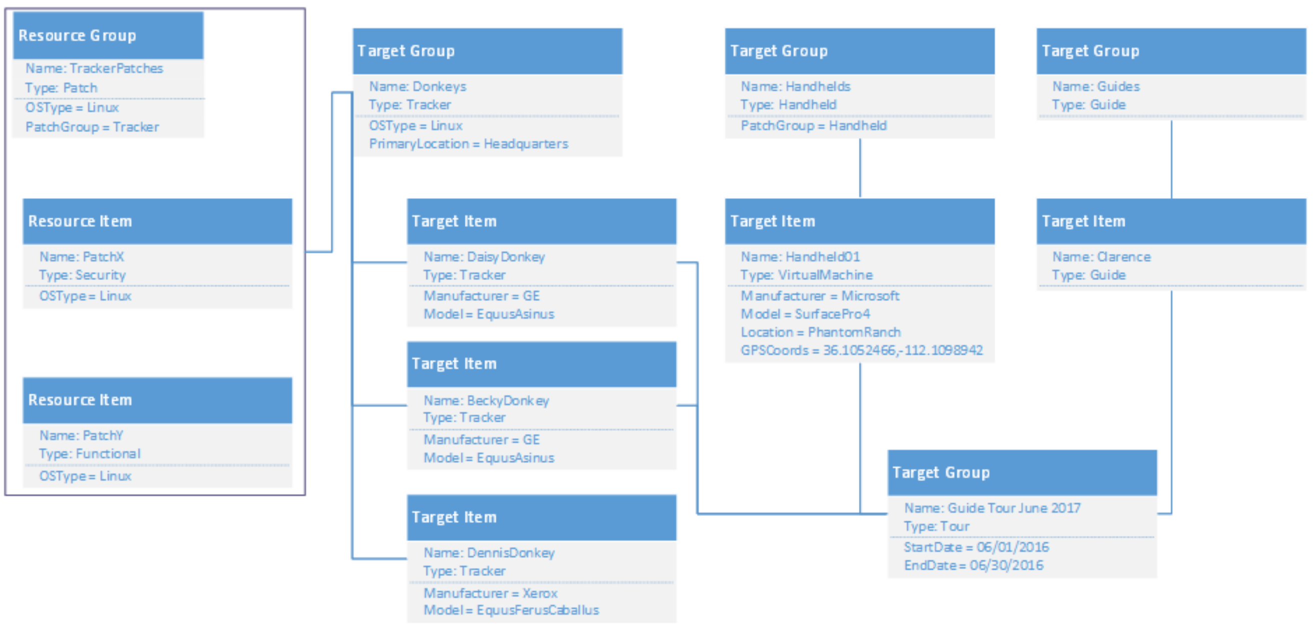
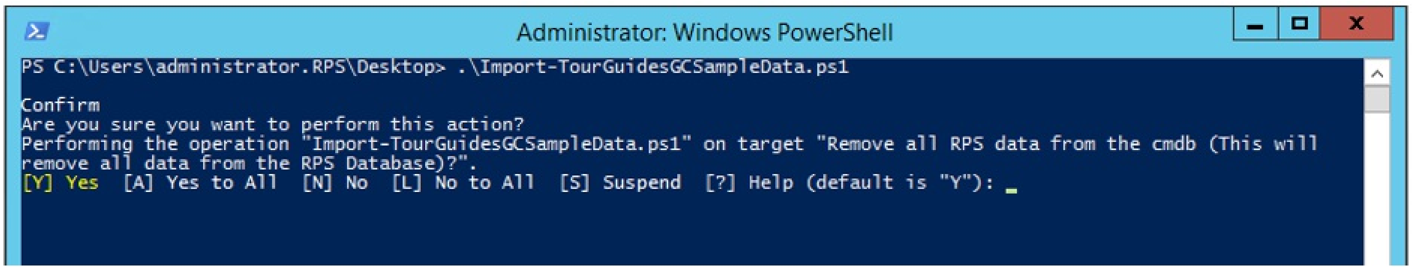


Figure 1 GC Tour Guides Example Dataset

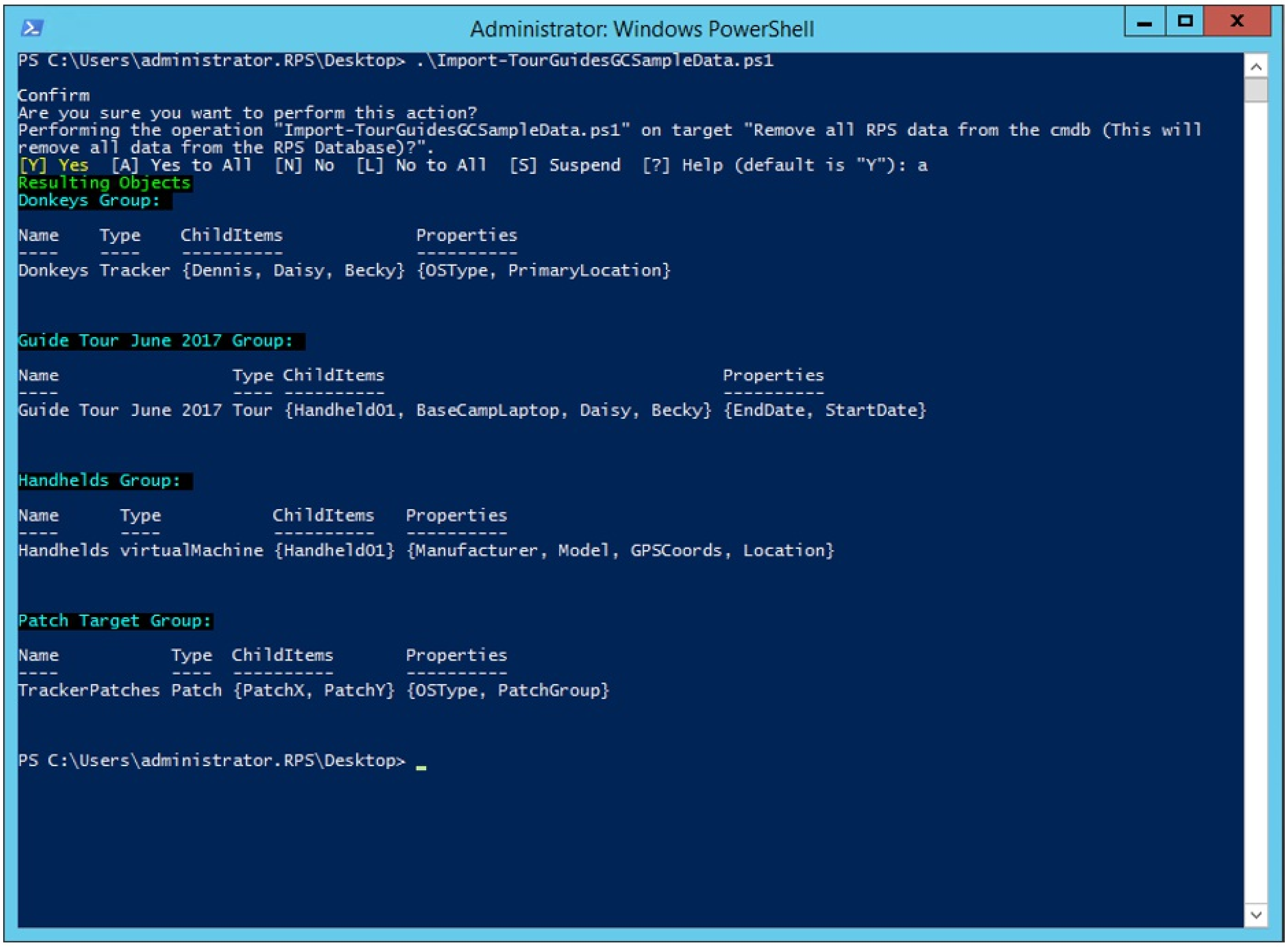
Importing Sample Data

A fully-functioning RPS Node is required to import the sample data from the script Import-TourGuidesGCSampleData.ps1 (located at $\Documents\Samples). To import the sample data, simply run the script from any RPS machine from a PowerShell console as shown in Figure 2. You will initially be prompted to clean the RPS database of any existing RPS Entities.



```
PS C:\Users\administrator.RPS\Desktop> .\Import-TourGuidesGCSampleData.ps1

Confirm
Are you sure you want to perform this action?
Performing the operation "Import-TourGuidesGCSampleData.ps1" on target "Remove all RPS data from the cmdb (This will
remove all data from the RPS Database)?".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): _
```

Proceed as necessary, and the objects described in this document will be created in the RPS CMDB. Once the data is imported (expected output is shown in Figure 3), you can begin manipulating the data as necessary to familiarize yourself with the RPS system.



```
PS C:\Users\administrator.RPS\Desktop> .\Import-TourGuidesGCSampleData.ps1

Confirm
Are you sure you want to perform this action?
Performing the operation "Import-TourGuidesGCSampleData.ps1" on target "Remove all RPS data from the cmdb (This will
remove all data from the RPS Database)?".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): a
Resulting Objects
Donkeys Group:

Name    Type    ChildItems             Properties
----    ----    ----------             ----------
Donkeys Tracker {Dennis, Daisy, Becky} {OSType, PrimaryLocation}


Guide Tour June 2017 Group:

Name                 Type ChildItems                                Properties
----                 ---- ----------                                ----------
Guide Tour June 2017 Tour {Handheld01, BaseCampLaptop, Daisy, Becky} {EndDate, StartDate}


Handhelds Group:

Name      Type           ChildItems   Properties
----      ----           ----------   ----------
Handhelds virtualMachine {Handheld01} {Manufacturer, Model, GPSCoords, Location}


Patch Target Group:

Name           Type  ChildItems       Properties
----           ----  ----------       ----------
TrackerPatches Patch {PatchX, PatchY} {OSType, PatchGroup}


PS C:\Users\administrator.RPS\Desktop> _
```

Figure 3 Expected output of the sample data import script

Desired State Configuration

GC Tour Guides would like to configure Clarence's handheld with DSC to ensure that it is in the correct state during expeditions.

For this scenario, GC Tour Guides has decided that one of RPS's existing DSC partial configurations will work for their needs. If this was not the case, they could author their own DSC partial configuration and add it to the system.

Before the configuration can be applied we need to add some information to the database. Figure 4 below displays a partial section of the entity graph from Figure 1 with supplemental information specific to this DSC example. Two properties – ComputerName and ConfigurationName – have been added to the Handheld01 Target Item. Additionally, a Resource Group has been added that defines the configuration applied to the handheld. For more detailed information on how these properties are connected and what other DSC features are supported by RPS, reference the DSC specific documentation listed in the Supporting Documentation section of this document.

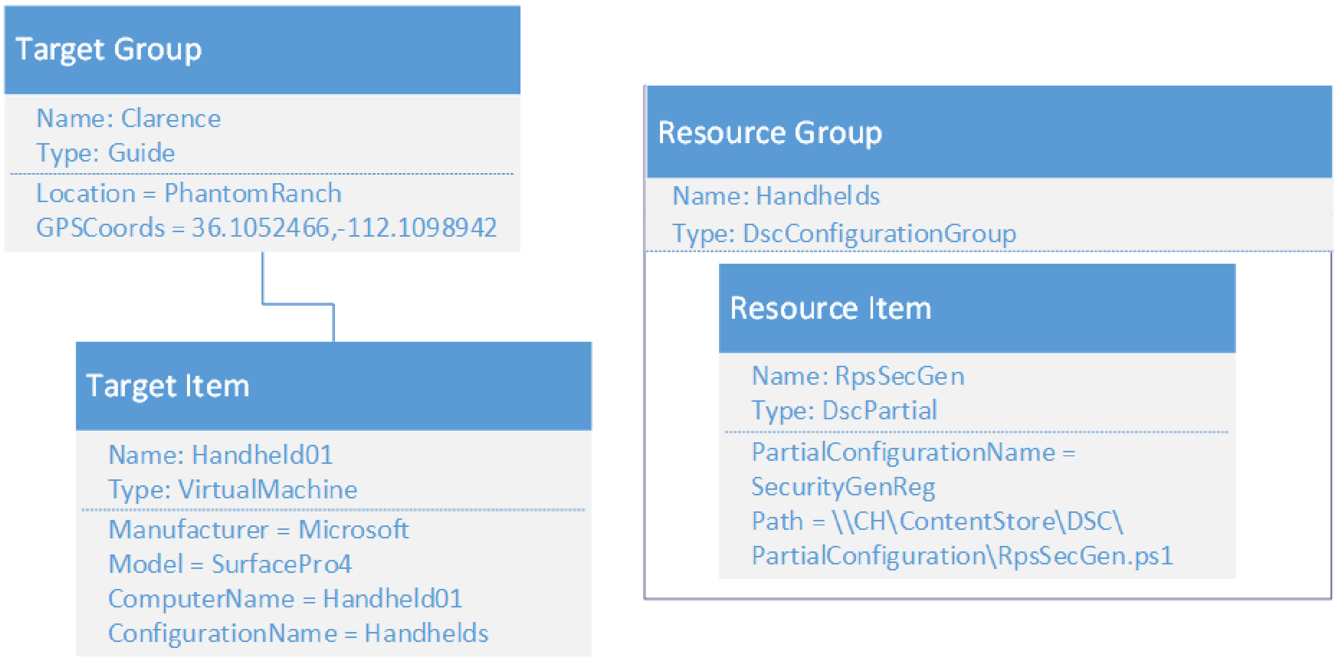


Figure 4 DSC Resource Group and Resource Item

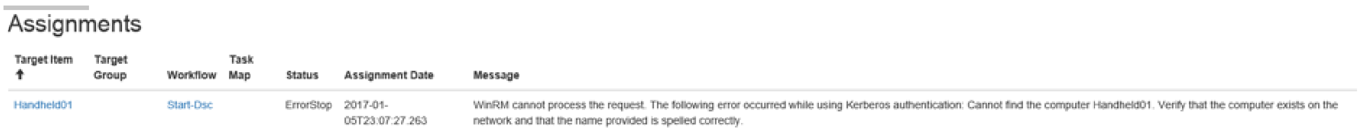
Executing the Configuration

With the additional supporting DSC elements in place, a Task Item can be generated to execute the RPS Start-Dsc Runbook (which will apply the DSC configuration) and a Task Assignment to initialize the Runbook.

Executing the Import-TourGuidesGCSampleData.ps1 script with the DscDemo switch will implement the DSC configurations with the sample dataset.

NOTE

After running the Import-TourGuidesGCSampleData.ps1 script you will be able to view the process of the configuration within the RPS website. Since the hardware does not exist, it is unreachable, so the Runbook fails and displays the error though the UI.



Assignments

Target Item ↑	Target Group	Workflow	Task Map	Status	Assignment Date	Message
Handheld01		Start-Dsc		ErrorStop	2017-01-05T23:07:27.263	WinRM cannot process the request. The following error occurred while using Kerberos authentication: Cannot find the computer Handheld01. Verify that the computer exists on the network and that the name provided is spelled correctly.

Figure 5 Task Assignment view in the UI shows runbook could not contact the Target Item.

Content Delivery Network

On Patch Tuesday, the GC guides are leading a group through the Grand Canyon. The GC IT team would like to patch the donkey's trackers with security patch X. First, however, they need to use the RPS CDN to distribute the patch from the parent RPS server stack to the Child RPS server.

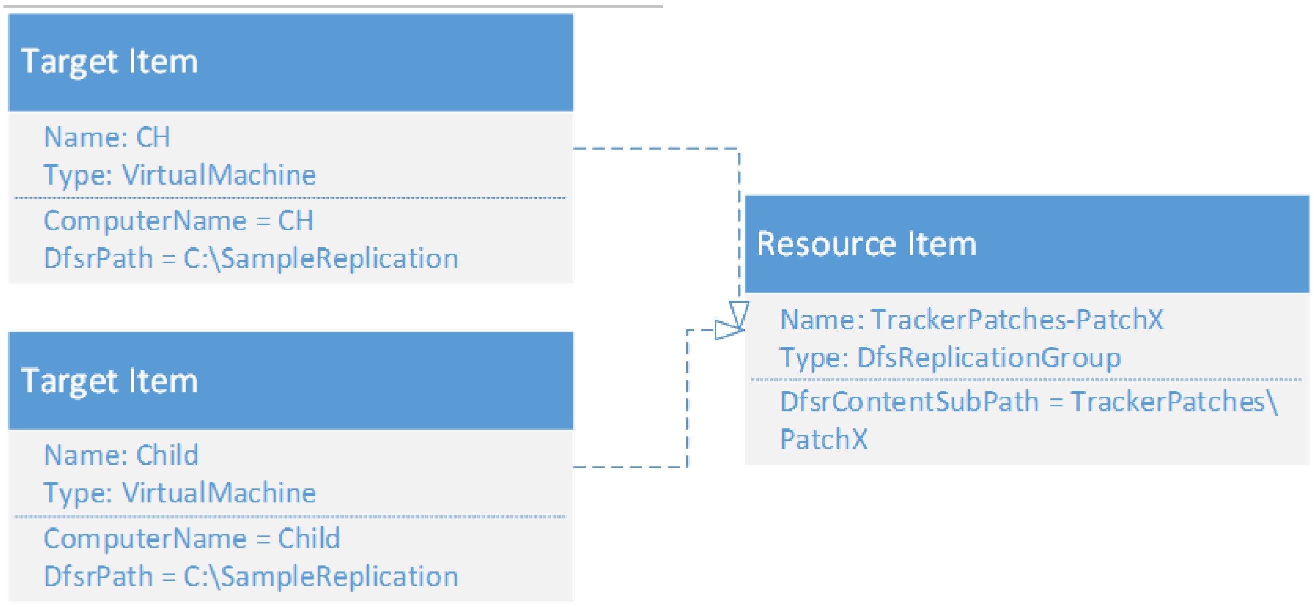
To accomplish this goal, they will need to update their database to configure DFS-R to replicate the patch files. For this example, our script Import-TourGuidesGCSampleData.ps1 script with the CdnDemo switch will handle making the changes for us.

First, the script will create the below directory structure on our DFS-R endpoint in the parent node (the CH server).

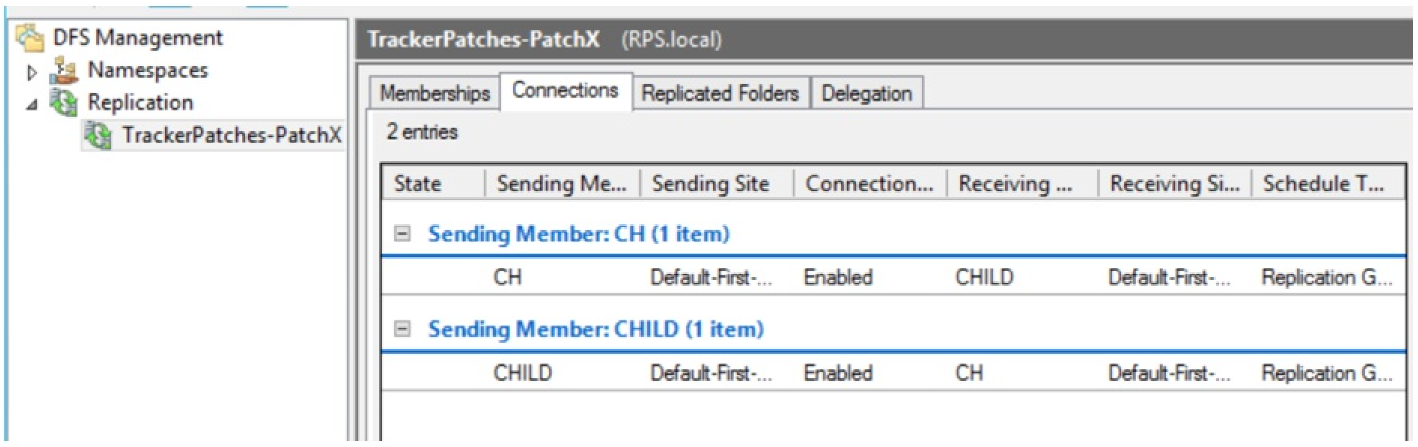
```
| C:\SampleReplication
|        TrackerPatches
|                PatchX
|                        PatchX.smpl
```

After the directory structure is created the CMDB is used to create the resource item TrackerPatches-PatchX as well as a replication group for the directory. Once this is completed the CH and Child target items are created or updated with the DfsrPath properties. Since DFS-R can replicate information to different relevant paths on each endpoint, each target items sets its DfsrPath that is concatenated with the resource item's property DfsrContentSubPath. This allows each endpoint to specify its own location, but supports a reasonable usage of relevant paths during automation. An assignment is created between the CH and Child target items with the DfsReplicationGroup.

In the end the RPS CMDB matches the below.


Target Item
Name: CH
Type: VirtualMachine
ComputerName = CH
DfsrPath = C:\SampleReplication
Target Item
Name: Child
Type: VirtualMachine
ComputerName = Child
DfsrPath = C:\SampleReplication
Resource Item
Name: TrackerPatches-PatchX
Type: DfsReplicationGroup
DfsrContentSubPath = TrackerPatches\PatchX

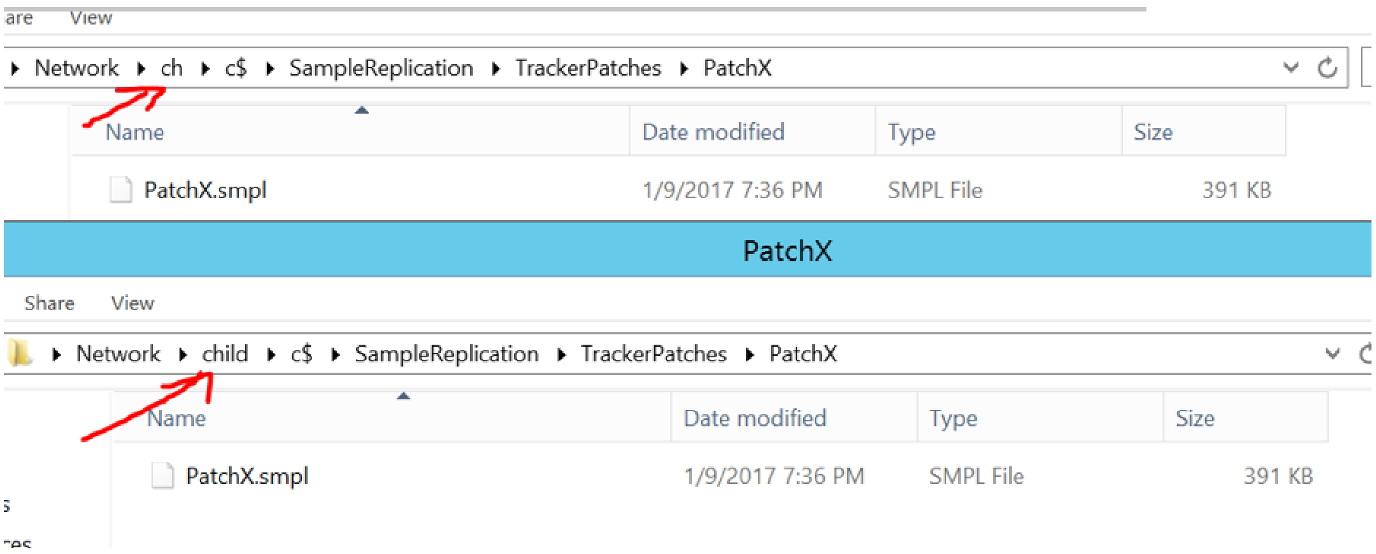
The DFS management console shows a replication group with two members, CH and Child.



2 entries

State	Sending Me...	Sending Site	Connection...	Receiving ...	Receiving Si...	Schedule T...
Sending Member: CH (1 item)						
	CH	Default-First-...	Enabled	CHILD	Default-First-...	Replication G...
Sending Member: CHILD (1 item)						
	CHILD	Default-First-...	Enabled	CH	Default-First-...	Replication G...

After waiting a few minutes the CH and Child server should have matching directories.


are View
Network ▸ ch ▸ c$ ▸ SampleReplication ▸ TrackerPatches ▸ PatchX
PatchX
Share View
Network ▸ child ▸ c$ ▸ SampleReplication ▸ TrackerPatches ▸ PatchX

Name	Date modified	Type	Size
PatchX.smpl	1/9/2017 7:36 PM	SMPL File	391 KB

Name	Date modified	Type	Size
PatchX.smpl	1/9/2017 7:36 PM	SMPL File	391 KB

Sync Service

The GC Tour Guides company would like to administer and monitor patching and weather update tasks from their main office (HQ). They will use a mobile laptop acting as a full RPS Node to sync with HQ. The RPS Sync Service runs in the background and synchronizes Target Items, Resources and Tasking data down to a child node and telemetry data back up to the parent node.

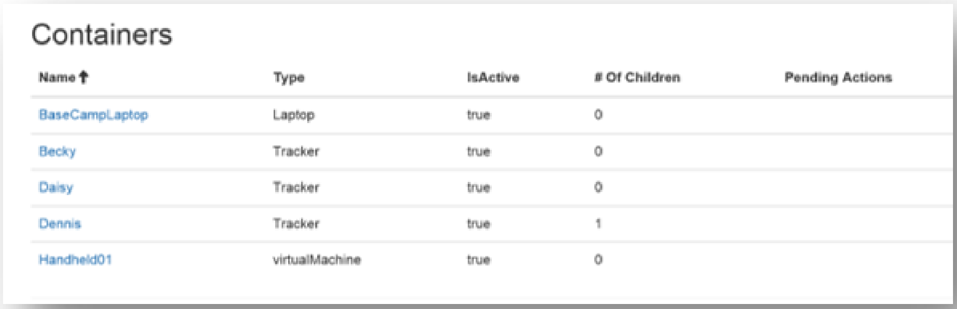
In our demo, the RpsChild VM will act as the mobile laptop. We'll assign Target Items which are part of a guided tour to the Child Node, and the RPS Sync Service will automatically synchronize the data and related tasks to that node when a connection is available. We'll create a TaskMap to simulate update of our weather app and update to the latest weather data.

Run the Demo

Run the Import-TourGuidesGCSampleData.ps1 script on the CH.RPS.local VM with the -SyncDemo switch to demonstrate how the Sync Service will synchronize automations to a child node and return telemetry data.

Verify Target Item and Assignments

From the SMA.RPS.local VM, open RPS Web at http://localhost:8080. You should see containers imported from the sample script. Although it isn't visible here, the BaseCampLaptop Target Item (TI) is assigned to the Child.RPS.local Node, and all automations will run there.



Containers

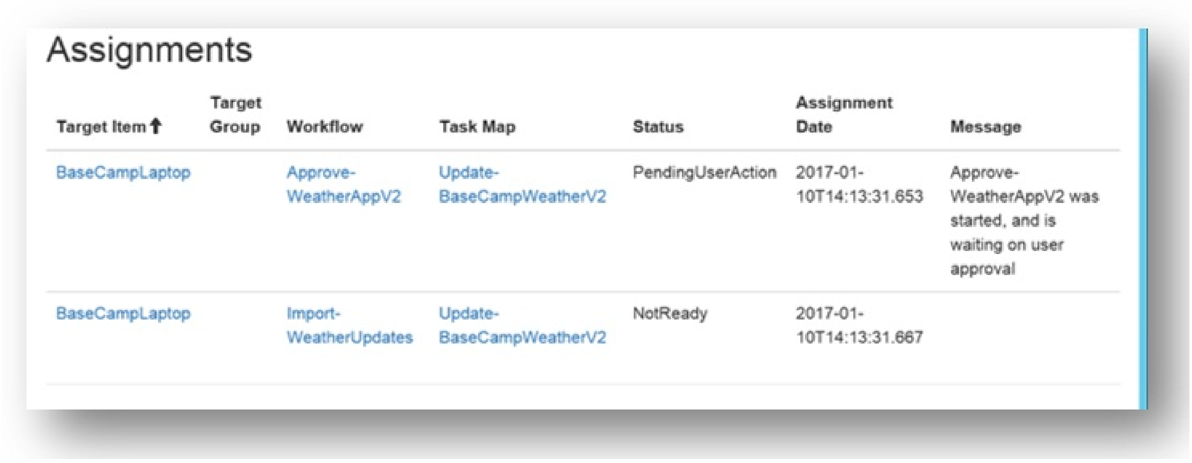
Name ↑	Type	IsActive	# Of Children	Pending Actions
BaseCampLaptop	Laptop	true	0	
Becky	Tracker	true	0	
Daisy	Tracker	true	0	
Dennis	Tracker	true	1	
Handheld01	virtualMachine	true	0	

The BaseCampLaptop TI is assigned to a Task Map with two sequential steps. Step 1 is an approval step for updating the Weather App, and Step 2 imports the latest weather data. Click on the BaseCampLaptop link or navigate to Tasking, Assignments to see the two tasks. Initially, the Status should appear to be NotReady, but after a few minutes, Step 1 will run on the child node.

NOTE

The two steps are sample runbooks which are imported into the c:/source/runbooks/ folder on the child VM.

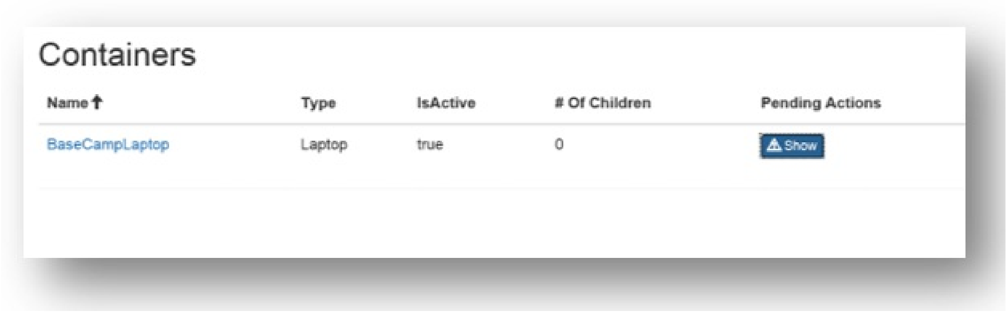
DSC is used to import any runbooks in that folder into SMA, so we must wait for that process to complete.



Assignments

Target Item	Target Group	Workflow	Task Map	Status	Assignment Date	Message
BaseCampLaptop		Approve-WeatherAppV2	Update-BaseCampWeatherV2	PendingUserAction	2017-01-10T14:13:31.653	Approve-WeatherAppV2 was started, and is waiting on user approval
BaseCampLaptop		Import-WeatherUpdates	Update-BaseCampWeatherV2	NotReady	2017-01-10T14:13:31.667	

Open RPS Web on the child VM. Notice that the BaseCampLaptop TI has been synchronized from the parent node, but not the other items. All the related Tasking and Target data has been synchronized as well.



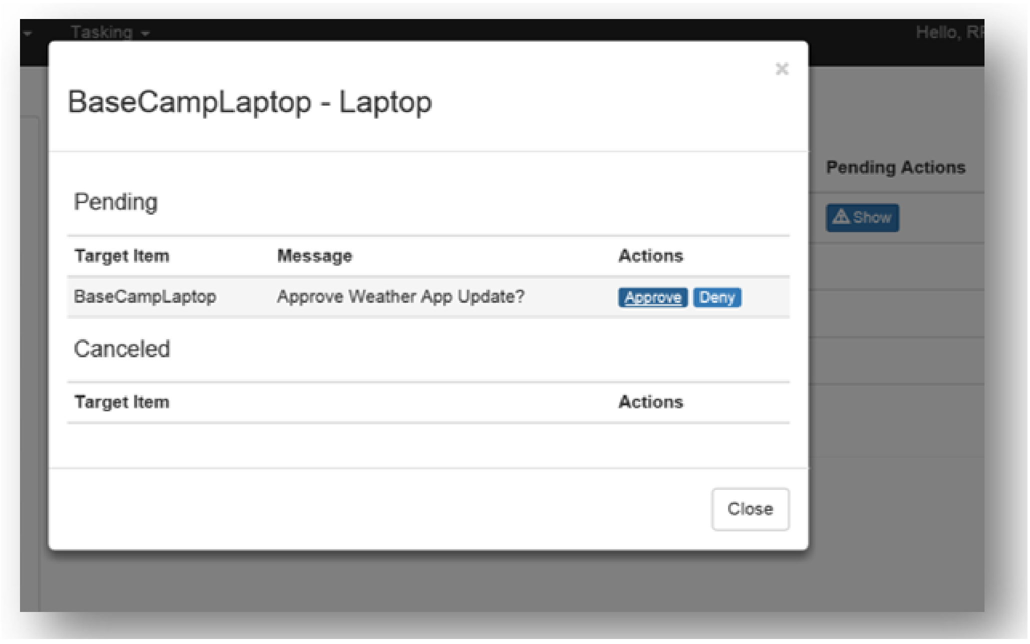
Containers

Name	Type	IsActive	# Of Children	Pending Actions
BaseCampLaptop	Laptop	true	0	Show

Step 1 - Approve Weather App Update

The first assignment in our Task Map requires user approval. This approval can be done via RPS Web at the parent (SMA.RPS.local) or child (child.RPS.local) node.

The first Task Assignment should be in the PendingUserAction status, and you will see a button to show Pending Actions. Click the Show button to open the approval.



BaseCampLaptop - Laptop

Pending

Target Item	Message	Actions
BaseCampLaptop	Approve Weather App Update?	Approve Deny

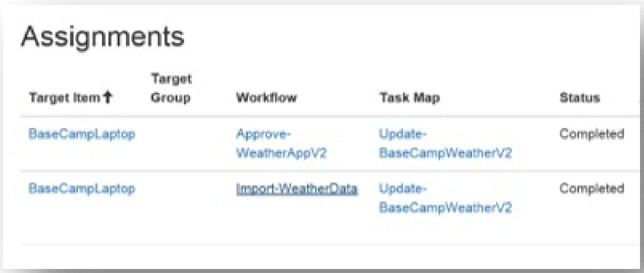
Canceled

Target Item	Actions

After approval, the Task Assignment status will be completed, and the status will synchronize between child and parent nodes.

Step 2 – Import Weather Data

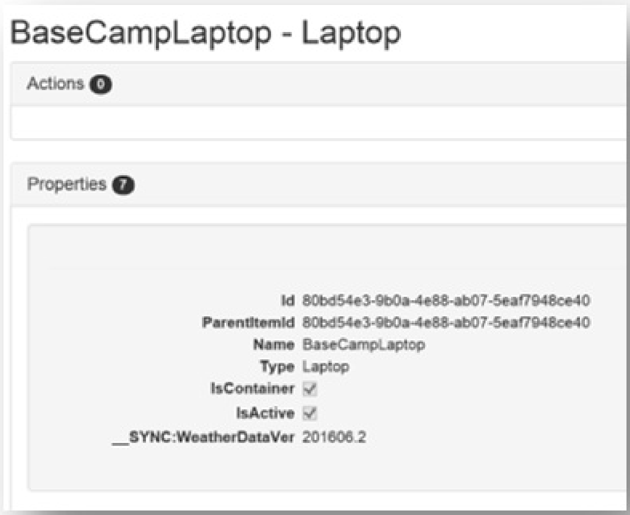
Step 2 will update to Ready status once Step 1 is complete. Then, the runbook to update the weather data will begin and the status will change to Running and finally Completed. The sample runbook, Import-WeatherData, will simulate the update by adding a special property to the BaseCampLaptop TI.



Assignments

Target Item	Target Group	Workflow	Task Map	Status
BaseCampLaptop		Approve-WeatherAppV2	Update-BaseCampWeatherV2	Completed
BaseCampLaptop		Import-WeatherData	Update-BaseCampWeatherV2	Completed

Open RPS Web on SMA.RPS.local to verify the Task Assignments are all synchronized. Click the BaseCampLaptop link to view the properties, and observe the property denoting the weather data version. This property is prefixed with "__SYNC:", which tells the Sync Service to synchronize the value back to the parent node.



Id	80bd54e3-9b0a-4e88-ab07-5eaf7948ce40
ParentItemId	80bd54e3-9b0a-4e88-ab07-5eaf7948ce40
Name	BaseCampLaptop
Type	Laptop
IsContainer	✓
IsActive	✓
__SYNC:WeatherDataVer	201606.2

Additional Scenarios

This sample just uses a simple Target Item and Task Map to illustrate how the Sync Service operates and how automations can be viewed and controlled remotely. In a more complex scenario, we'd see more Nodes and Containers with many child items representing physical or virtual devices, and complex task maps that administer patches, collect data, etc. In the GC Tour Guides example, we could imagine that each guided group has a Laptop which acts as an RPS Node. The physical things, including donkeys, people, handheld devices and communication equipment may be represented as Target Items in one container. Headquarters, which acts as the parent RPS Node, would monitor all of the various groups in RPS Web.

Conclusion

RPS is a very customizable solution, which can sometimes be overwhelming. This sample scenario describes a simple implementation of how RPS can be used to accomplish automation tasks with minimal overhead.

More Resources

- RPS Software Design

RPS Install Guide

Purpose

The purpose of this document is to provide an overview of the Rapid Provisioning System (RPS) Install process and detailed instructions for setting up a virtualized RPS Environment using Hyper-V.

Audience

This document is intended for anyone evaluating or testing RPS, such as IT Staff or Developers. Users should have some familiarity with core RPS Concepts as well as basic PowerShell and Windows comprehension.

System Requirements

1. Windows 10/Server 2016
2. PowerShell/WMF 5.1
3. 16GB RAM
4. 100GB HDD free-space
5. Hyper-V PowerShell Module & Management Tools

NOTE

RPS Authored content is signed, but 3rd party code may not be. RPS Installer was tested with PowerShell Execution Policy set to RemoteSigned.

Ports and Protocols

Ports and Protocols information can be found in the Ports and Protocols Guide

Service Accounts

The following accounts are used by RPS for setup and maintenance of Nodes.

ACCOUNT (ROLE)	DESCRIPTION	PERMISSIONS
DomainAdmin	Create/promote a DC and manage AD Computers, Users, Groups and OUs	AD DomainAdmin*
ServerAdmin	Push certificates and settings, manage DSC configuration, pull files from content store	AD Account w/ Local Admin
LocalAdmin	Manage machine settings for non-domain joined computer	Local Admin

NOTE

DomainAdmin membership is required to create a new Domain Controller. After initial creation, the account should be removed from DomainAdmins, but should still retain permissions to manage AD Users, Computers, Groups and OUs.

RPS Installation (Default)

Installing RPS requires the latest RPS release, the install media for SQL, SMA and windows features, and a Hyper-V image for the Windows 2012R2 VMs that will be created. The instructions below can be used to build a default Master RPS Node, which includes a Domain Controller (AD.Master.Rps) and an application server (App.Master.Rps). You can also choose to create a Region Node, Site Node, and an Inception Node, based on available resources.

Download Content

Before installing RPS in Hyper-V, you must download the RPS Release, static Install Media and a base image for RPS VMs.

1. Download the RPS Release with media (i.e. RpsWithMedia_v2_3.zip) from the RPS Release website.
2. Extract **RpsWithMedia_v2_3.zip** to a location such as **c:\RPS**

   NOTE

   This location will be referred to as **Install Root**.

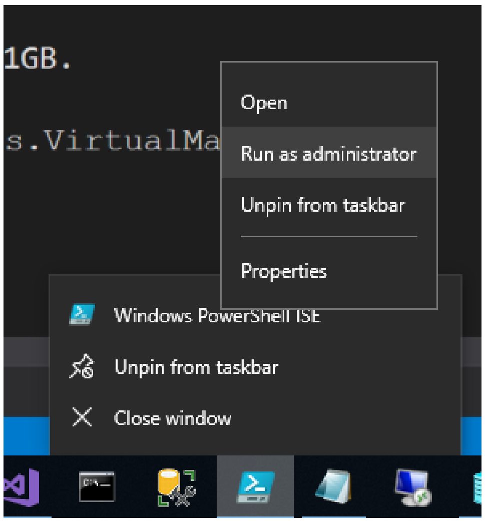
3. Download the Windows Server 2012R2 Hyper-V image from Windows_Server_2012_R2_VL-dev.zip.
4. Extract **Windows_Server_2012_R2_VL-dev.zip** to a location which is **NOT** in **Install Root**. > [!NOTE] > Do not add the .vhdx file into the **Install Root**.

The **Install Root** folder should contain the following sub-directories.

- \ContentStore
  - \Certificates
  - \CMDB
  - \Demos
  - \Documents
  - \DSC
  - \Export
  - \Images
  - \iPxeDistro
  - \Management
  - \Modules
  - \RpsBitsDownloadService
  - \RpsGui
  - \RpsProvisioning
  - \RpsSync
  - \Runbooks
  - \Setup
  - \SqlSecurity
  - \SqlServer2012
  - \SystemCenter2016
  - \Utilities
  - \Windowserver2012

Install RPS

1. Open PowerShell as Administrator. Right-click the PowerShell Icon from Start Menu or Task Bar and select **Run as administrator**



2. Set location to **Install Root**\ContentStore\ by executing:

```
Set-Location c:\RPS\ContentStore\
```

3. Install RPS and supply the location of the VM Template vhdx, the NodeType and if needed, specify specific configuration using -NodeConfigurationName. See examples below:

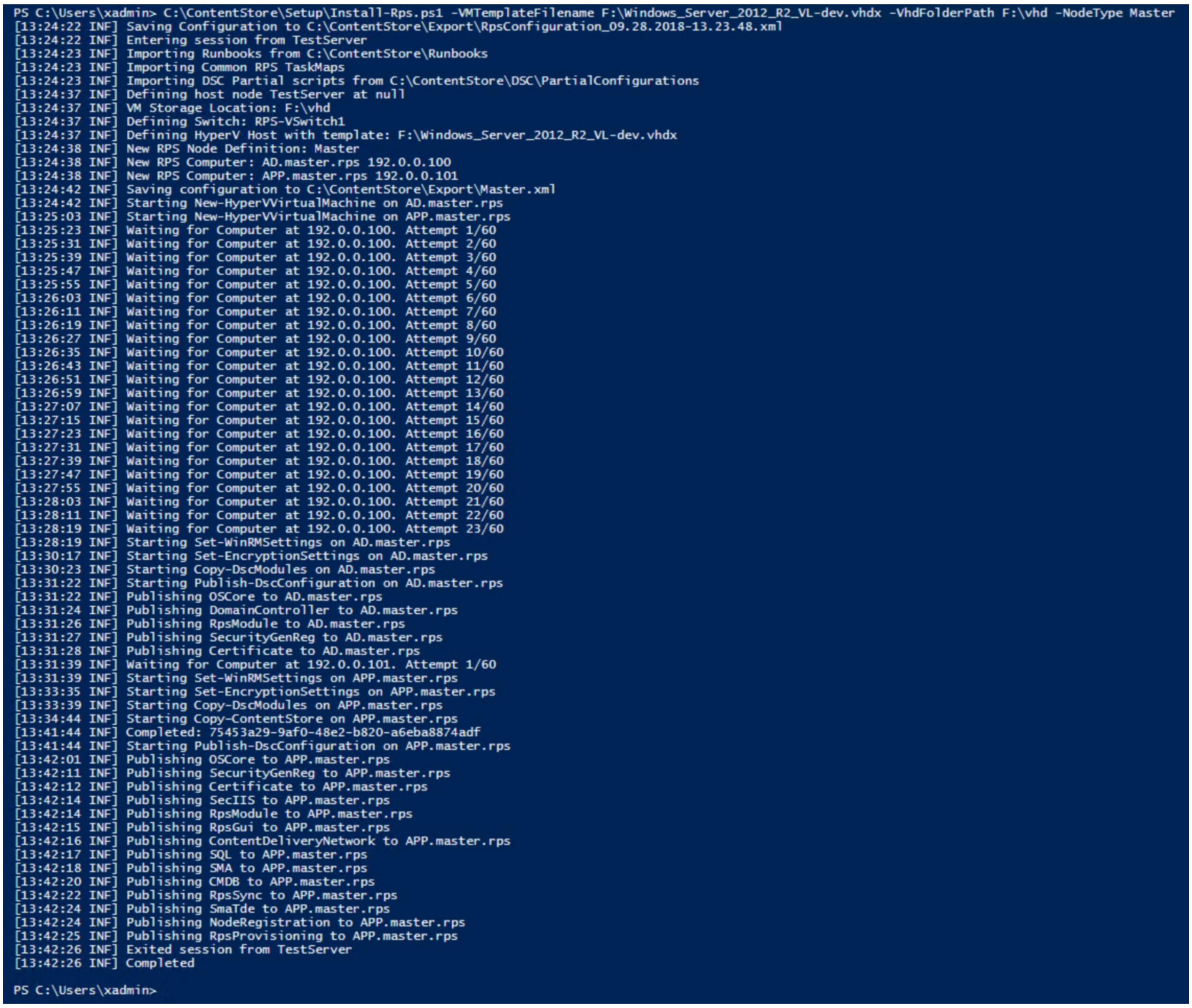
```
.\Setup\Install-Rps.ps1" -VMTemplateFilename D:\Common\Windows_Server_2012_R2_VL-dev.vhdx -NodeType
'master'
```

```
.\Setup\Install-Rps.ps1 -VMTemplateFilename D:\Common\Windows_Server_2012_R2_VL-dev.vhdx -
NodeConfigurationName MN -Enclave SIPR -NodeType Provisioning
```

NOTE

When using "MN" for the NodeConfigurationName, -Enclave must be specified. For further options, refer to the **Install-Rps Parameter Definition** table below.

Output similar to the following should appear:



```
PS C:\Users\xadmin> C:\ContentStore\Setup\Install-Rps.ps1 -VMTemplateFilename F:\Windows_Server_2012_R2_VL-dev.vhdx -VhdFolderPath F:\vhd -NodeType Master
[13:24:22 INF] Saving Configuration to C:\ContentStore\Export\RpsConfiguration_09.28.2018-13.23.48.xml
[13:24:22 INF] Entering session from TestServer
[13:24:23 INF] Importing Runbooks from C:\ContentStore\Runbooks
[13:24:23 INF] Importing Common RPS TaskMaps
[13:24:23 INF] Importing DSC Partial scripts from C:\ContentStore\DSC\PartialConfigurations
[13:24:37 INF] Defining host node TestServer at null
[13:24:37 INF] VM Storage Location: F:\vhd
[13:24:37 INF] Defining Switch: RPS-VSwitch1
[13:24:37 INF] Defining HyperV Host with template: F:\Windows_Server_2012_R2_VL-dev.vhdx
[13:24:38 INF] New RPS Node Definition: Master
[13:24:38 INF] New RPS Computer: AD.master.rps 192.0.0.100
[13:24:38 INF] New RPS Computer: APP.master.rps 192.0.0.101
[13:24:42 INF] Saving configuration to C:\ContentStore\Export\Master.xml
[13:24:42 INF] Starting New-HyperVVirtualMachine on AD.master.rps
[13:25:03 INF] Starting New-HyperVVirtualMachine on APP.master.rps
[13:25:23 INF] Waiting for Computer at 192.0.0.100. Attempt 1/60
[13:25:31 INF] Waiting for Computer at 192.0.0.100. Attempt 2/60
[13:25:39 INF] Waiting for Computer at 192.0.0.100. Attempt 3/60
[13:25:47 INF] Waiting for Computer at 192.0.0.100. Attempt 4/60
[13:25:55 INF] Waiting for Computer at 192.0.0.100. Attempt 5/60
[13:26:03 INF] Waiting for Computer at 192.0.0.100. Attempt 6/60
[13:26:11 INF] Waiting for Computer at 192.0.0.100. Attempt 7/60
[13:26:19 INF] Waiting for Computer at 192.0.0.100. Attempt 8/60
[13:26:27 INF] Waiting for Computer at 192.0.0.100. Attempt 9/60
[13:26:35 INF] Waiting for Computer at 192.0.0.100. Attempt 10/60
[13:26:43 INF] Waiting for Computer at 192.0.0.100. Attempt 11/60
[13:26:51 INF] Waiting for Computer at 192.0.0.100. Attempt 12/60
[13:26:59 INF] Waiting for Computer at 192.0.0.100. Attempt 13/60
[13:27:07 INF] Waiting for Computer at 192.0.0.100. Attempt 14/60
[13:27:15 INF] Waiting for Computer at 192.0.0.100. Attempt 15/60
[13:27:23 INF] Waiting for Computer at 192.0.0.100. Attempt 16/60
[13:27:31 INF] Waiting for Computer at 192.0.0.100. Attempt 17/60
[13:27:39 INF] Waiting for Computer at 192.0.0.100. Attempt 18/60
[13:27:47 INF] Waiting for Computer at 192.0.0.100. Attempt 19/60
[13:27:55 INF] Waiting for Computer at 192.0.0.100. Attempt 20/60
[13:28:03 INF] Waiting for Computer at 192.0.0.100. Attempt 21/60
[13:28:11 INF] Waiting for Computer at 192.0.0.100. Attempt 22/60
[13:28:19 INF] Waiting for Computer at 192.0.0.100. Attempt 23/60
[13:28:19 INF] Starting Set-WinRMSettings on AD.master.rps
[13:30:17 INF] Starting Set-EncryptionSettings on AD.master.rps
[13:30:23 INF] Starting Copy-DscModules on AD.master.rps
[13:31:22 INF] Starting Publish-DscConfiguration on AD.master.rps
[13:31:22 INF] Publishing OSCore to AD.master.rps
[13:31:24 INF] Publishing DomainController to AD.master.rps
[13:31:26 INF] Publishing RpsModule to AD.master.rps
[13:31:27 INF] Publishing SecurityGenReg to AD.master.rps
[13:31:28 INF] Publishing Certificate to AD.master.rps
[13:31:39 INF] Waiting for Computer at 192.0.0.101. Attempt 1/60
[13:31:39 INF] Starting Set-WinRMSettings on APP.master.rps
[13:33:35 INF] Starting Set-EncryptionSettings on APP.master.rps
[13:33:39 INF] Starting Copy-DscModules on APP.master.rps
[13:34:44 INF] Starting Copy-ContentStore on APP.master.rps
[13:41:44 INF] Completed: 75453a29-9af0-48e2-b820-a6eba8874adf
[13:41:44 INF] Starting Publish-DscConfiguration on APP.master.rps
[13:42:01 INF] Publishing OSCore to APP.master.rps
[13:42:11 INF] Publishing SecurityGenReg to APP.master.rps
[13:42:12 INF] Publishing Certificate to APP.master.rps
[13:42:14 INF] Publishing SecIIS to APP.master.rps
[13:42:14 INF] Publishing RpsModule to APP.master.rps
[13:42:15 INF] Publishing RpsGui to APP.master.rps
[13:42:16 INF] Publishing ContentDeliveryNetwork to APP.master.rps
[13:42:17 INF] Publishing SQL to APP.master.rps
[13:42:18 INF] Publishing SMA to APP.master.rps
[13:42:20 INF] Publishing CMDB to APP.master.rps
[13:42:22 INF] Publishing RpsSync to APP.master.rps
[13:42:24 INF] Publishing SmaTde to APP.master.rps
[13:42:24 INF] Publishing NodeRegistration to APP.master.rps
[13:42:25 INF] Publishing RpsProvisioning to APP.master.rps
[13:42:26 INF] Exited session from TestServer
[13:42:26 INF] Completed

PS C:\Users\xadmin>
```

Install-Rps Parameter Definitions

PARAMETERS	DESCRIPTIONS
VMTemplateFilename (Required)	Path to the '.vhdx' file which will serve as the Hyper-V VHDX template.
VhdFolderPath	Path to a folder to store the .vhdx files used to create VMs. If not specified .vhdx files will be stored in same directory path as specified with **VMTemplateFileName**.
ConfigFileName	Name of file containing the RPS Configuration data. If one is not specified, it will be created in Export directory.
NodeType (Required)	The desired node you want to install. ***(i.e. Master)***, if you want to install all nodes then choose 'All'
SkipVMCreation	Switch to use Virtual Machines already created and configured with proper networking.
SkipCopyContent	Switch to not copy **InstallRoot** to the CDN server. ***(i.e. CH)***
SkipDSCPrep	Switch to not setup DSC MOF Encryption, WinRM HTTPS Listener, and not copy DSC Resources.
ComputerName	Array of the Virtual Machines you want to install from the Node. ***(i.e.*** `@("AD.Master.Rps", "App.Master.Rps")` ***)***

PARAMETERS	DESCRIPTIONS
SkipDSCPublish	Switch to skip publishing DSC.
CopyCDN	Switch to copy patches that were imported into the configuration.
DeleteVMs	Switch to delete the VirtualMachines if they exist during a new installation.
SkipDscModuleCopy	Switch to not copy **Dsc Required Modules** to the target's PowerShell module path.
Esxi	Switch to indicate whether the hypervisor is Esxi (default is Hyper-V).
GenerateXmlOnly	Switch to make usability easier to toggle all -Skip* switches.
NodeConfigurationName	The name of the configuration to use for install. Choose either Rps, Lab, or MN currently defaulted to Rps. If MN is selected, the dynamic Enclave parameter must be set as well.
TaskMapName	Name of the desired system task map that will be assigned to the target items. Default value of Install-Rps.
Enclave (Dynamic)	Required if **NodeConfigurationName** is "MN". Valid options are **"SIPR"**, **"Colorless"**, and **"Lab"**.

Manual Node Installation/Repair

1. Install-RPSNode will install the Node you specify.

PARAMETERS	DESCRIPTIONS
NodeType (Required)	The desired node you want to install. ***(i.e. Master)***, if you want to install all nodes then choose 'All'
SkipVMCreation	Switch to use Virtual Machines already created and configured with proper networking
SkipCopyContent	Switch to not copy **InstallRoot** to the CDN server. ***(i.e. CH)***
SkipDSCPrep	Switch to not setup DSC MOF Encryption, WinRM HTTPS Listener, and not copy DSC Resources
ComputerName	Array of the Virtual Machines you want to install from the Node. ***(i.e.*** `@("AD.Master.Rps", "App.Master.Rps")` ***)***
SkipDSCPublish	Switch to skip publishing DSC.
CopyCDN	Switch to copy patches that were imported into the configuration.
DeleteVMs	Switch to delete the VirtualMachines if they exist during a new installation.
SkipDscModuleCopy	Switch to not copy **Dsc Required Modules** to the target's PowerShell module path.
ConfigFilename	Path to an existing configuration export file. ***(i.e. D:\Exports\Master.xml)***
ContentStorePath	Path to the content store.
TaskMapName	Name of the desired system task map that will be assigned to the target items. Default value of Install-Rps.

```
Enter-RpsSession -Path "D:\Exports\Master.xml"
Install-RpsNode -NodeType "Master"
```

Note about SMA, Orchestrator process, and Master-Controller starting manually

RPS Security

The information below has been verified on the RPS 3.1, which shipped from Microsoft in Spring of 2020.

Ports and Protocols

RPS uses various ports and protocols for operation. Some ports are configurable as part of the RPS deployment and configuration, and some are outside the management of RPS and/or not configurable. The table below shows these RPS components (where * Indicates port is configurable via RPS).

Table 1: RPS Ports and Protocols

COMPONENT	DESCRIPTION	PORTS	PROTOCOLS
RPS API	Direct management of configuration data in SQL Server	1433	TCP (TLS 1.2)
RPS Sync Service	Synchronize Data and Static Files between RPS Nodes and managed RPS Targets	777*	HTTPS
DFSR	Transfers files between nodes within a domain	445, 135	RPC, TCP
BITS	Transfers files between nodes on different domains	80, 443	HTTP/S
RPS Web	Administrative Website for RPS	8080*	HTTPS
RPS Provisioning Service	Baremetal/iPXE Service via the specific DNS name **rpsprovisioning**	443*	HTTPS
SMA	Service Management Automation	9090	HTTPS
TER Reader	Trust Element Repository – Reader (DCA)	3443	TCP
TER Writer	Trust Element Repository – Writer (DCA)	5443	TCP
WinRM	Windows Remote Management	5985/5986	HTTP/HTTPS
SMB	File Sharing	445	SMB/HTTPS
ICMP	Device Availability		ICMP
DHCP	Dynamic Host Configuration Protocol	67-69	UDP
DNS	Domain Name Server	53	UDP/TCP

Host-Based Security System (HBSS) uses some common ports (e.g., ports 80, 443, 1433, etc.) though requires additional ports be used for full operation. Please see HBSS documentation, at https://kc.mcafee.com/corporate/index?page=content&id=KB66797.

Service Accounts

The following accounts are used by RPS for setup and maintenance of Nodes.

RPS Account Roles: Domain Accounts

Table 2: Domain Accounts

ACCOUNT (ROLE)	DESCRIPTION	PERMISSIONS
DomainAdmin	Has full control of the domain. Administrator rights on all domain controllers and member servers.	**AD:** DomainAdmin[1]
DomainJoin	Used to join computers to the domain. Rights are scoped specifically for that purpose.	**AD:** Force change password Read/Write Computers
ProvSvc	Provisioning Website App Pool Identity	**SQL:** Service Permissions[2]
GuiSvc	RPS Website App Pool Identity	**RPS:** Master Key Encryption **SQL:** Service Permissions[2]
SmaRunbookSvc	SMA Runbook	**RPS:** Master Key Encryption **SQL:** Service Permissions[2]
SmaWebSvc	SMA IIS App Pool	**SQL:** Service Permissions[2]
SqlAgentSvc	SQL Server Agent	**AD:** Log on as a Service
SqlSvc	SQL Server Service	**AD:** Log on as a Service **SQL:** DBOwner
SyncSvc	Sync Service Account	**AD:** Log on as a Service **SQL:** Service Permissions[2]

RPS Account Roles: Server Accounts

Table 3: Server Accounts

ACCOUNT (ROLE)	DESCRIPTION	PERMISSIONS
ServerAdmin	Push certificates and settings, manage DSC configuration, pull files from content store	AD Account w/ Local Admin
LocalAdmin	Manage machine settings for non-domain joined computer	Local Admin

RPS Account Roles: Other

Table 4: Other Accounts

ACCOUNT (ROLE)	DESCRIPTION	PERMISSIONS
SA	Account used to setup SQL. This account is disabled per STIG after setup is complete.	SQL

[1] Domain Administrator membership is required to create a new Domain Controller. After initial creation, the account should be removed from this group, but should still retain permissions to manage AD Users, Computers, Groups and OUs.

[2] Service SQL permissions are scoped to the RpsDb only and include Execute, Select, Insert, Update, Delete, and SyncHistory change tracking view permissions.

Security

Partial Configurations

All RPS Partial Configs must define the following parameters:

- **IPAddress** - Accessible IP Address of the computer we'll publish DSC Configuration to.
- **DSCEncryptionCertificate** - Information about the certificate used to encrypt the MOF (configuration). The LCM is set to use this certificate and any partials that are not secured will not run on a target.
- **OutputPath** - Location to temporarily store the MOF file once its compiled.

Runbooks

Many Runbooks will need to connect to the Target (Computer) to perform their duty. To connect, you must get the appropriate credential and then establish a secure connection. Runbooks use the `GetRpsCredential` cmdlet to load the right credential for the target; then use `New-SecureSession` from Rps-Api to make the connection.

Patching

Patch Management in RPS requires communication via HTTPS. The certificate authority (CA) that signed the webserver's certificate must be trusted by the Linux client or patches will not be downloaded. This is done by installing the public certificate of the CA.

Certificates

The RPS Solution uses certificates for a variety of functions, including:

- Web Site SSL binding for HTTPS encrypted transport between server (e.g., RPS Website) and client.
- RPS Sync Service for client\server authentication between **subscriber** (e.g., RPS Sync Service on Region) and **distributor** (e.g., RPS Sync Service on Master) nodes. The certificate thumbprints for all trusted nodes are whitelisted in the RPS CMDB.
- RPS Sync Service for HTTPS encrypted transport between server and client.
- DSC MOF file credentials encryption (By default DSC encrypts the entire MOF file).
- WinRM for HTTPS encrypted transport between server and client.
- SQL for HTTPS encrypted transport between server and client.
- Provisioning SSL binding for HTTPS encrypted transport between server (e.g., RPS Provisioning) and client.
- Encryption of secrets in the database (protected properties).
- Encryption of XML configurations.

Each certificate must be derived from a trusted root certificate that resides in the Trusted Root Certification Authorities store.

Table 5: Certificates

ROLE	DISTRIBUTION	KEY USAGES	PURPOSE
DscEncryption	Per VM	Key Encipherment, Data Encipherment (30)	MOF credential encryption
ProvisioningSSL	APP Master	Key Encipherment, Data Encipherment	HTTPS support for Provisioning Website
ClientCdn	Per VM		
FeedSSL			
GuiSSL	Per APP VM	Digital Signature, Non-Repudiation, Key Encipherment (e0)	HTTPS support for RPS GUI Website
iPxeSSL	Per APP VM	Digital Signature, Non-Repudiation, Key Encipherment (e0)	HTTPS support for iPXE Website
MasterKeyEncryption	Per Node		Protecting the Master Key

ROLE	DISTRIBUTION	KEY USAGES	PURPOSE
NodeEncryption	Per APP VM		Encrypting node configuration
Root	Per VM	Certificate Signing, Off-line CRL Signing, CRL Signing (06)	Deriving other certificates
Sync	Per APP VM		Allowing Sync to occur between nodes
SyncSSL	Per APP VM	Key Encipherment, Digital Signature, Non-Repudiation	Data-in-transit encryption for node sync
SqlSSL	Per APP VM		Data-in-transit encryption for SQL data
WinRM	Per VM		

Master Key

The Master Key (MK) is used to protect secrets in the database (i.e., credential/certificate passwords). Since the MK is high-value, it's encrypted using the public key of a certificate. Appropriate users are given access to the private key of the Master Key Encryption Certificate (MKEC) so that they may access the MK and decrypt protected properties in the database.

The same MK should be used for all nodes that will share secrets. The default boundary for secrets is an Active Directory domain since domain accounts will likely need access to all domain computers. This implementation is fungible, however any changes to the default implementation made by the customer/integrator may risk customer data.

Accounts which are preconfigured with the MasterKeyEncryption role during setup will have permissions to manipulate protected properties in the target environment. In order to give this permission to new users once RPS is installed, the role should be added to the appropriate account in the CMDB and DSC should be republished (at minimum, the RpsCertificate partial).

When a protected property is retrieved or set, access is determined by retrieving the MasterKeyCertThumbprint property on the node. If the user has access to the corresponding certificate private key in the LocalMachine\Personal store, they are granted access to the MK. If the user does not have rights to the MKEC, access to protected properties will be denied.

RPS Software Design

Introduction

The Rapid Provisioning System (RPS) is designed to provide a mobile, modular, and extensible framework that allows coded automation activities to be executed across a wide area network environment with limited connectivity and bandwidth. This solution was designed to survive the inherent constraints and challenges that have been observed within a mobile tactical network. RPS is designed to be a flexible solution for programs with similar constraints and automation requirements. This document will outline the technology, approach, and functionality provided by this solution.

Basic RPS Definitions

This section provides the definition for terms that are used throughout this document. Refer to this section for how these terms are being used within the RPS system.

TERM	DEFINITION
RPS Node	A running instance of the RPS system
Configuration Item (CI)	A vehicle that contains entity definitions
DSC	Microsoft's PowerShell Desired State Configuration software package
Cmdlet	A lightweight command that is used in the Windows PowerShell environment

Constraints and Challenges of a Tactical Environment

There are many constraints and challenges inherent to a mobile tactical network.

- Network capability is limited to modern satellite and line of site
  - It is common for vehicles not to have network connectivity for extended periods of time. This period can range from a few days to several weeks.
  - When a satellite connection is available, vehicles experience extremely high latency combined with extremely low throughput.
  - Line of site provides an improvement in the network capabilities of the vehicle, but has limited accessibility when vehicles are on the move (OTM).
- Immediate response capabilities for combat situations
  - RPS is designed as a combat-ready system to support the warfighter in the field.
  - Service downtime must be schedulable and interruptible.
- Long lead times to fielded changes
  - Time for changes to hardware and software takes years to progress from adoption to operating in a fielded solution.
  - Fielded solutions often have a long lifespan due to costs associated with high cost to upgrade and difficulty implementing changes to the field.
  - The RPS solution will be designed to use the latest commercial off the shelf (COTS) solutions to provide for the longest possible lifespans.

Design Goals

The RPS toolset is designed to provide the following functionality to meet the requirements of the tactical Environment:

- Targeted automations across various device types (routers, switches, radios, servers, etc.).

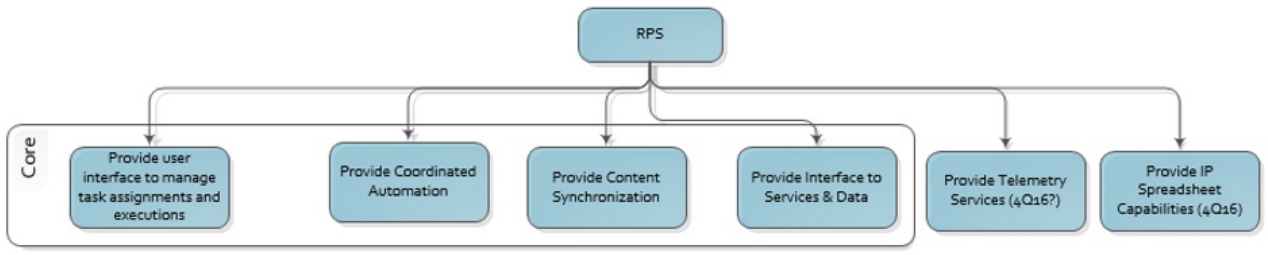
- Codependency on automations tasks (To accommodate cross-device requirements and prerequisites).
- A componentized automations model to simplify development of automations tasks.
- An extensible toolset to meet automations needs to accommodate environmental variances.
- Offline automations capabilities, specifically the ability to discretely operate with limited or no connectivity to upper-tier systems.
- Low-compute capabilities for executing on low resource systems.

Architectural Overview

The RPS system is designed to provide the infrastructure upon which various PORs can extend to automate tasks specific to their missions and deployments.

RPS Functionality

The common set of functionality provided by RPS is shown below.



RPS
Core
Provide user interface to manage task assignments and executions
Provide Coordinated Automation
Provide Content Synchronization
Provide Interface to Services & Data
Provide Telemetry Services (4Q16?)
Provide IP Spreadsheet Capabilities (4Q16)

Figure 1 - Functional Hierarchy
(source: Document > Architecture > SW Design Diagrams.vsd > 'Overall Func Arch' sheet)

1. **User interface to manage task assignments and executions** – Provides the ability for a user to manage the provisioning process at a granular level. The user can start, halt, and return feedback on provisioning processes for a specific target item.
2. **Coordinated Automation** – Provides the ability to automate and coordinate a variety of tasks.
3. **Content Synchronization** – Provides the ability to synchronize content across RPS nodes to support the tasks being automated.
4. **Interface to Services & Data** – Provides a standard interface to access RPS services through a well-documented API.
5. **Telemetry Services** – Provides the ability to collect heuristic information from a system.
6. **IP Spreadsheet Capabilities** – Provides the ability to read in the standard TNACC/TNIC provided IP spreadsheet and deliver expected results.

RPS Components

COMPONENT	CORE TECHNOLOGY	DESCRIPTION	FUNCTIONALITY MAPPING
Data Persistence	MS SQL Server	Underlying data storage that maintains the metadata needed by the system to manage the task automations. Commonly referred to as RPS CMDB.	Coordinated Automation
Automation Framework	MS SMA	Underlying automations framework used by RPS to provide RPS automation of tasks.	Coordinated Automation
Configuration Management	DSC	Underlying configuration management framework to specify, setup and maintain a desired machine configuration.	Coordinated Automation Content Synchronization Interface to Services and Data UI
Content Delivery	BITS	Underlying content delivery framework to replicate and deliver content across the network.	Content Synchronization

COMPONENT	CORE TECHNOLOGY	DESCRIPTION	FUNCTIONALITY MAPPING
Messaging Service	C#	Underlying messaging framework to provide command and control communication capability between RPS nodes.	Content Synchronization
Core Modules	C#, PowerShell	RPS specific implementation to provide a generic capability to perform task automation. This includes all the RPS runbooks, modules and capabilities that are exposed via the RPS API.	Coordinated Automation, Content Synchronization , Interface to Services and Data , Telemetry Services, IP Spreadsheet Capabilities
Application Server	IIS	Provides the capability to host a web application.	UI
RPS Web Application	.ASP.NET MVC, IIS, JQuery, Knockout, C#	RPS specific user interface to allow a user to initiate automated tasks on target systems.	UI

NOTE

Refer to each component's individual documentation for more details on the internals of each component.

Data Persistence (CMDB)

The detailed software design documentation for this component is available at:

RPS > Documents > Operations > **RPS Data Persistence (CMDB) Design.docx**

Automation Framework

The detailed software design documentation for this component is available at:

RPS > Documents > Operations > **RPS Automations Package Guidelines.docx**

Configuration Management

The detailed software design documentation for this component is available at:

RPS > Documents > Operations > **RPS Configuration Management (DSC) Design.docx**

Content Delivery

The detailed software design documentation for this component is available at:

RPS > Documents > Operations > **RPS Content Management.docx**

Messaging Service

The detailed software design documentation for this component is available at:

RPS > Documents > Development > **RPS Sync Services.docx**

Application Server

The application server provides the ability to host a web application. RPS uses Microsoft's IIS web server to provide this capability.

RPS Web Application

The detailed software design documentation for this component is available at:

RPS > Documents > Software > **TODO**

Software Development Kit (SDK)

The RPS SDK provides the developer-oriented documentation for the RPS system. It will include:

```
* RPS binary code
* Sample PowerShell source code runbooks
* RPS > Documents > Samples
* API and cmdlet documentation
* RPS > Documents > Development > **RPS API Documentation.PDF**
```

RPS Server Architecture

The RPS server architecture is intended to provide scalability for most automations requirements. To accommodate this, each RPS node will be able to either operate on its own or receive instructions from another RPS node. Those instructions will provide operations information to the RPS solution to request executions of automations code, or to manage its own environment.

Software Requirements

The RPS infrastructure requires the following software:

- A Microsoft Windows operating system (Windows Server 2012 R2 or greater)
  - The core operating system the solution will run on
- RPS API
  - Provides the .NET and PowerShell modules necessary to access, manage and maintain a RPS Node
- Microsoft SQL Server 2012
  - Provides the underlying SQL support for the custom Configuration Management Database (CMDB)
  - Provides the core SQL services to SMA
- Microsoft Service Management Automation (SMA)
  - Automation engine
- .NET Framework 5
  - Provides enhanced PowerShell frameworks and support for advanced automations
- Windows Management Framework (WMF) 5.1
  - Extends PowerShell support to enhance Desired State Configuration functionality
- Microsoft Distributed File System Replication services (DFSR)
  - Software for maintaining content distribution over slow, unstable WAN links
- RPS Sync Service
  - Provides Windows service to synchronize RPS nodes

Server Role Minimum Requirements

ROLE	CPU CORES	HARD DISK (GB)	RAM (GB)
SQL	4	20	4
SMA	2	60	4
GUI	2	50	4
CDN	2	100 (Depends on content)	4

Server Architecture

It is important to note that the server layout of the RPS components is designed to be fluid based on requirements. This section will outline the software components of the RPS architecture and how they interact, rather than the server architecture of a specific implementation.

The software components of a given node can be placed on any server layout provided network communications and software prerequisites are available. Each collection of RPS components is identified as a "RPS Node". These nodes are intended to operate both individually, or in a distributed workload capacity. Node layouts will vary based upon the requirements of a given implementation. For example:

- Single server RPS node (all software on one system)
- Single server RPS node, multiple SMA runbook workers on multiple servers
- Multi-server RPS node (each component on its own instance of Windows across multiple servers)
- Multi-server RPS node, multi-SMA runbook workers (separate component installs, multiple SMA runbook workers)

Sample RPS Architecture

A sample RPS deployment is shown in Figure 3. This architecture places systemically similar components on separate compute entities to maximize efficiency.

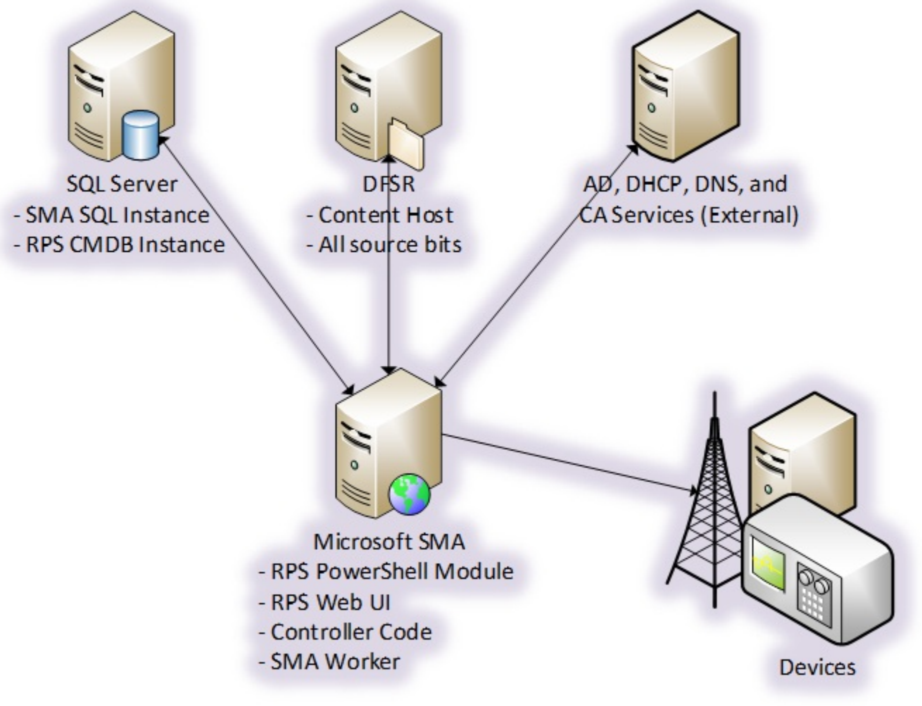


Figure 3 Sample RPS Architecture

Figure 4 illustrates a multi-node RPS hierarchical architecture. In this example, the Master Node may exist in the cloud; however, this is not a requirement.

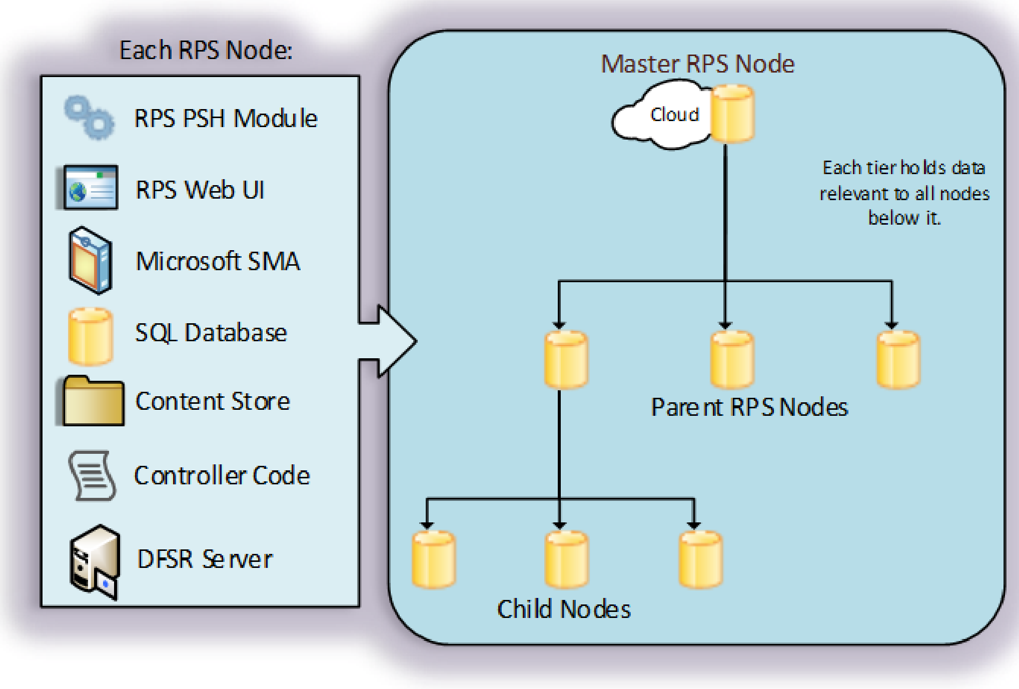


Figure 4 Multiple RPS Nodes in Hierarchy

RPS API - PowerShell Module and Data Access

The RPS API is a single library to be used in conjunction with the RPS CMDB. The API provides .NET and PowerShell interfaces for managing and manipulating RPS data and actions.

Powershell and API Functionality

Table 2 outlines the available actions the RPS API and PowerShell module offers to support and maintain the RPS environment.

HIGH LEVEL ACTIONS	PURPOSE
Managing Tasks	Provides the commands necessary to manage and register tasks.
Managing Target Items	Manage Target Items, properties, their relationship to other items, and other item related metadata.
Managing Resource Items	Manage resources, their properties, and metadata.
Managing Task Maps	Manage Task Maps and their definitions, what tasks to run, what their dependencies are, filtering criteria, etc.
Managing Resource Items	Manage Assignments between Resource Items and Target Items.
Managing Task Assignments	Manage Assignments between Tasks/Task Maps and Target Items.

HIGH LEVEL ACTIONS	PURPOSE
Misc Toolset Functionality	Actions such as resets, initiating Task Map evaluations, setting requests for user-interaction, or other functions that may not necessarily directly relate to CMDB data.
Managing RPS Infrastructure Data	Actions such as registering and managing RPS nodes and their relationships to each other.

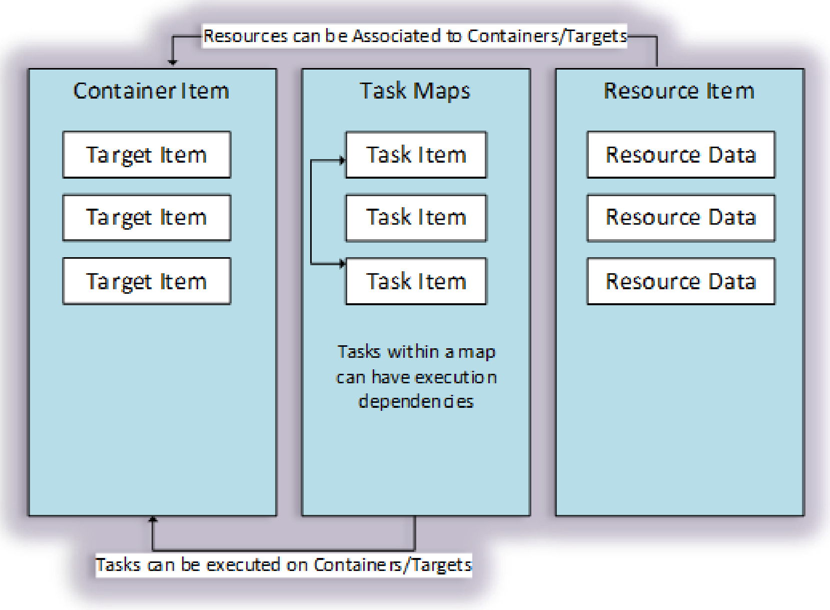
More Resources

- RPS Data Persistence (CMDB) Software Design
- RPS Master Controller Design
- RPS Configuration Management (DSC) Software Design

RPS Data Persistence (CMDB) Software Design

The data persistence feature of the Rapid Provisioning System (RPS) provides the underlying data storage needs of the system. The RPS Content Management Database (CMDB) maintains the metadata that is needed by RPS to targetable containers of items, the resources to be applied to those items, tasks and task maps for orchestrating task execution on those items, and the node hierarchy in which these items reside.

The figure below shows a high-level relationship of how these meta-data types interact for the purposes of automations.


Resources can be Associated to Containers/Targets
Container Item
Target Item
Target Item
Target Item
Task Maps
Task Item
Task Item
Task Item
Tasks within a map can have execution dependencies
Resource Item
Resource Data
Resource Data
Resource Data
Tasks can be executed on Containers/Targets

RPS CMDB – ER Database Diagram

The detailed entity relationship diagram documentation for the RPS CMDB is available at: RPS > Trunk > Documents > Architecture > **RPSEntityRelationshipDiagram.vsd**

NOTE

This document is intended to outline the physical database layout.

TERM	DEFINITION
RPS	Rapid Provisioning System.
Node	A running instance of the RPS system.
CMDB	Configuration Management Database (custom for RPS).
DSC	Microsoft PowerShell Desired State Configuration software package.
SMA	Microsoft Systems Management Automation - a platform for automating tasks via PowerShell workflow.
Device	Any computer, router or other networked device which can be targeted by an automated risk.
Automation	A single automated task that is executed by the RPS system.

TERM	DEFINITION
Target	Any container or item on a container that can be directly targeted for task execution by SMA.
Resource	Any item of resource that can be applied to a target through a running task.
Task	Any workflow/runbook that can be executed on a target by SMA.

Metadata Storage – RPS Tables

This section will describe the tables needed to persist data for the RPS API. This document will further describe how each table is related and leveraged by the RPS API.

RPS coordinates automations by leveraging the CMDB for target, task, resource, and node configuration metadata. This data represents the containers/items to target, resources to assign, tasks to assign, orchestrate & automate, and hierarchal nodes to locate and synchronize. This data is all stored in a very generic format to offer maximum flexibility and extensibility for any user of the solution. Breaking down data into a generic data model allows for any type of target, resource, and task information to be stored and utilized within the RPS Database for coordinated automation and synchronization.

The table below briefly describes the current tables represented within the RPS CMDB:

DATA TYPER	FUNCTION
Node	A Node is metadata about an identifiable location that is a running instance of RPS. Nodes can have parent/child relationships. Changed data is synchronized between related nodes.
TargetItem	A TargetItem is metadata about a device item or container. Target items can have parent/child relationships. A top-level target item is called a 'Container' since it always contains a set of targetable items. Target items can have tasks assigned to be executed on them via automation. Target items can have resources assigned to be applied to them. An example of a target item is a router or a switch.
Container	A Container is metadata about a TargetItem that is a top-level parent of a set of target Items. Task maps are only assigned to target items that are containers.
TargetGroup	A TargetGroup is metadata about a logical grouping of target items. A TargetGroup has a set of TargetItem members to simplify task/resource assignments.
ResourceItem	A ResourceItem is metadata about a resource item that is not an executable task. A resource item is either a local or global item of resource that is applied to a targetable item by a task workflow/runbook. An example of a resource item is an operating system patch or a configuration template.
ResourceGroup	A ResourceGroup is metadata about a logical grouping of resource items. A ResourceGroup has a set of ResourceItem members to simplify resource management and assignements.
ResourceAssignmentStatus	A ResourceAssignmentStatus is metadata about a resource assignment to a targetable item and its current state. Resource assignments are defined and managed by individual tasks (workflows).
TaskItem	A Task is metadata about a task workflow. A task is a workflow that is executed by SMA. Tasks are assigned to target items directly or through task maps for coordinating task execution on target items.
TaskMap	A TaskMap is metadata about a task map that is assigned to target item containers for automation by SMA. A task map contains a set of definitions that define which task items are assigned to which target items, and in what order based on dependencies and filters.

DATA TYPER	FUNCTION
TaskMapDefinition	A TaskMapDefinition is metadata about a task map definition. A task map definition defines which task item is assigned to which target item. An optional property bag filter can be associated.
TaskMapDefDependency	A TaskMapDefDependency is metadata about a task map definition dependency. Each instance defines the parent/child dependency between two task map definitions. These dependencies are used by SMA to control task execution order during automation.
TaskMapDefFilter	A TaskMapDefFilter is metadata about a task map definition filter. A filter is used to further define which target items a task item is being assigned to by specifying which property bag key/value pairs are present on a target item.
TaskMapAssignment	A TaskMapAssignment is metadata about a task map assignment to a target. A task map assignment contains information about the assignment of a task map to a target item or group of target items.
TaskAssignment	A TaskAssignment is metadata about a task assignment to a targetable item and its current state. Task assignments are sometimes direct assignments of a task to a target item. Task assignments are also generated through assignments of a task map to a container. SMA uses task assignment state to control automation.
TaskAssignmentDependency	A TaskAssignmentDependency is metadata about a task assignment dependency. Each instance defines the parent/child dependency between two task assignments. These dependencies are generated from task map definition dependencies during a task map assignment to a target. These dependencies are used by SMA to control task execution order during automation.
TaskAssignmentUserAction	A TaskAssignmentUserAction is metadata about a user action associated with a task assignment when task assignment state is 'PendingUserAction'.
ItemProperty	An ItemProperty is a key/value property bag item that contains all custom metadata for an RPS item allowing for extensibility. The ItemProperty table is used by TargetItem, TargetGroup, ResourceItem, ResourceGroup, Node, TaskItem, TaskAssignment, and ResourceAssignmentStatus.
TaskState	A TaskState is an enumeration of task states used for task assignments by the SMA controller.
LocalConfig	A LocalConfig is metadata about a local RPS instance. An example is the identification of the local RPS node.

RPS CMDB Relationships

The detailed design diagram for database entity relationships (ER) on how database entities are related to each other can be found here: RPS > Trunk > Documents > Architecture > **RPSEntityRelationshipDiagram.vsd**

The CMDB database tables have the following cardinality in relation to other tables:

- A **Node** can have zero-or-more child Nodes, and zero-or-one parent Node.
- A **Node** can contain zero-or-more TargetItems.
- A **Node** can contain zero-or-more local ResourceItems.
- A **Node** can have zero-or-more ItemProperties.
- A **LocalConfig** can reference one local Node.
- A **TargetItem** can have zero-or-more child TargetItems, and zero-to-one parent TargetItems.
- A **TargetItem** can have zero-or-more ItemProperties.
- A **TargetItem** can belong to one Node.
- A **TargetItem** can belong to zero-or-more TargetGroups.
- A **TargetItem** can belong to zero-or-more ResourceAssignments.

- A **TargetItem** can belong to zero-or-more TaskAssignments.
- A **TargetGroup** can have zero-or-more TargetItems.
- A **TargetGroup** can have zero-or-more ItemProperties.
- A **ResourceItem** can have zero-or-more child ResourceItems, and zero-or-one parent ResourceItem.
- A **ResourceItem** can belong to one local Node or be global.
- A **ResourceItem** can have zero-or-more ItemProperties.
- A **ResourceItem** can belong to zero-or-more ResourceGroups.
- A **ResourceItem** can belong to zero-or-more ResourceAssignments.
- A **ResourceGroup** can have zero-or-more ResourceItems.
- A **ResourceGroup** can have zero-or-more ItemProperties.
- A **ResourceAssignmentStatus** can reference one ResourceItem.
- A **ResourceAssignmentStatus** can reference one TargetItem.
- A **ResourceAssignmentStatus** can have zero-or-more ItemProperties.
- A **TaskItem** can belong to zero-or-more TaskMapDefinitions.
- A **TaskItem** can belong to zero-or-more TaskAssignments.
- A **TaskItem** can have zero-or-more ItemProperties.
- A **TaskMap** can have zero-or-more TaskMapDefinitions.
- A **TaskMap** can have zero-or-more TaskMapAssignments.
- A **TaskMapDefinition** can belong to one TaskMap.
- A **TaskMapDefinition** can be associated with one TaskItem.
- A **TaskMapDefinition** can have zero-or-more dependent child TaskMapDefinitions, and zero-or-more dependent parent TaskMapDefinitions.
- A **TaskMapDefinition** can have zero-or-more TaskMapDefFilters.
- A **TaskMapDefFilter** can be associated with one TaskMapDefinition.
- A **TaskMapAssignment** can reference one TaskMap.
- A **TaskMapAssignment** can have zero-or-more TaskAssignments.
- A **TaskAssignment** can reference one TaskMapAssignment.
- A **TaskAssignment** can reference one TaskItem.
- A **TaskAssignment** can reference one TargetItem.
- A **TaskAssignment** can have zero-or-more dependent child TaskAssignments, and zero-or-more dependent parent TaskAssignments.
- A **TaskAssignment** can have zero-or-more ItemProperties.
- A **TaskAssignmentUserAction** can reference one TaskAssignmentStatus

RPS CMDB Detail Descriptions

Nodes

Nodes provide a physical location for a set of containers, target items and local resource items that is globally identifiable by the synchronization of data between nodes. Custom node properties are stored in an item property bag.

For example, the defined hierarchy of nodes are assigned containers, then also task and resource assignments. A parent node can automatically provide a child node through synchronization data describing a patch or other release, and the controller can begin to execute the release as soon as the data is available.

Containers – Automations Scope

Containers provide a scoping mechanism to properly 'group' items together for automations purposes. It is defined as the top-most parent in a TargetItem's hierarchy. When a parent/child hierarchy of TargetItems is established, the highest-level item is automatically derived as the "Container". Custom Container properties are stored in an item property bag.

When executing TaskMaps that require more complex execution chains, the container provides the scope that is representative of how to coordinate those tasks across devices. This enables the capability to define many similar sets of devices with different scopes for automation purposes.

Consider a series of systems that are interdependent, such as a set of servers on which different pieces of software reside. Software or devices may require that certain components are configured before others. The container offers a scoping mechanism for how those dependencies relate.

Other examples include:

- A datacenter containing servers. Representing the datacenter as a TargetItem, then adding child TargetItems.
- A vehicle containing equipment. Representing the vehicle as an TargetItem, then adding computer devices as child TargetItems.

Both of these offer the scope necessary to automate their TargetItems as related to each other.

⚠ **IMPORTANT**

A TargetItem may only belong to one container. Containers cannot belong to other containers, as they are derived from the top-most parent item in a parent TargetItem / child TargetItem hierarchy.

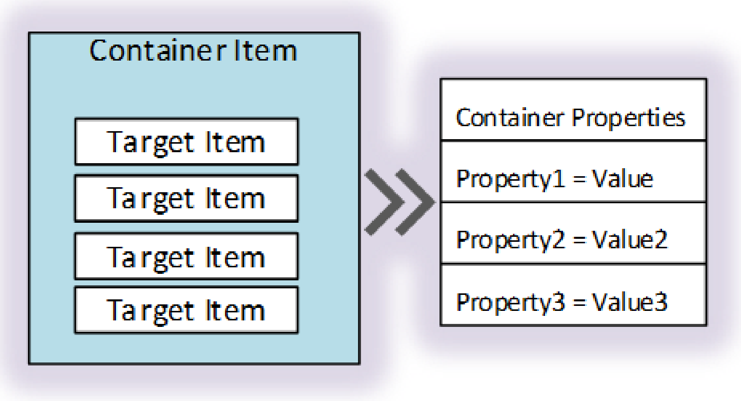


Figure 1 - Container: contains items, has properties

TargetItems – Target Device Data

TargetItems represent the data about an individual item and its properties. A target item is simply a data representation of the device that requires automation. This could be physical or logical. A target item belongs to a container for the purposes of scope and executions. Custom TargetItem properties are stored in an item property bag.

Examples include:

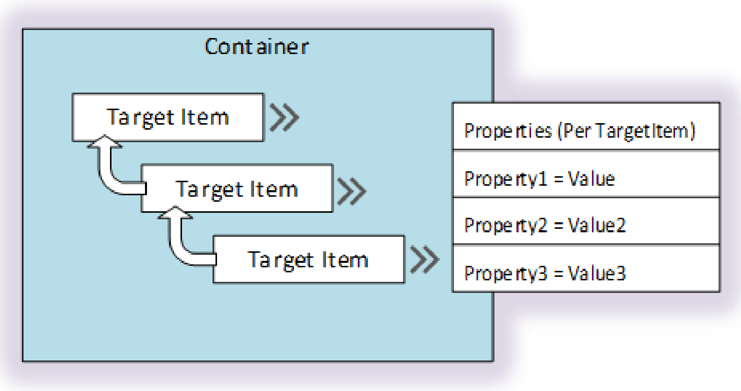
- A computer or server
- A router or switch
- A 'vehicle', logically vehicle may be physical, it is not a 'device' that we automate against, however there may be a model in which that is the case.

NOTE

TargetItems also maintain a parent-child relationship to each other, leaving the highest level parent always automatically defined as the container. This offers the ability to structure item relationships around their relationships to other items while maintaining a scope automatically. As this scoping mechanism is logical in nature, it can be defined using whatever logic best suits the business case. (E.g., a "Vehicle" containing vehicles, or a "Datacenter" containing servers, or a "Computer" containing software).

The figure below shows three sample target items, as related to each other. Each has its own separate list of properties, but all exist within the context of the Container, while the container itself is the top-most parent defined in the hierarchy of items.

Figure 2 - Target items with Properties, as related to eachother and their Container.



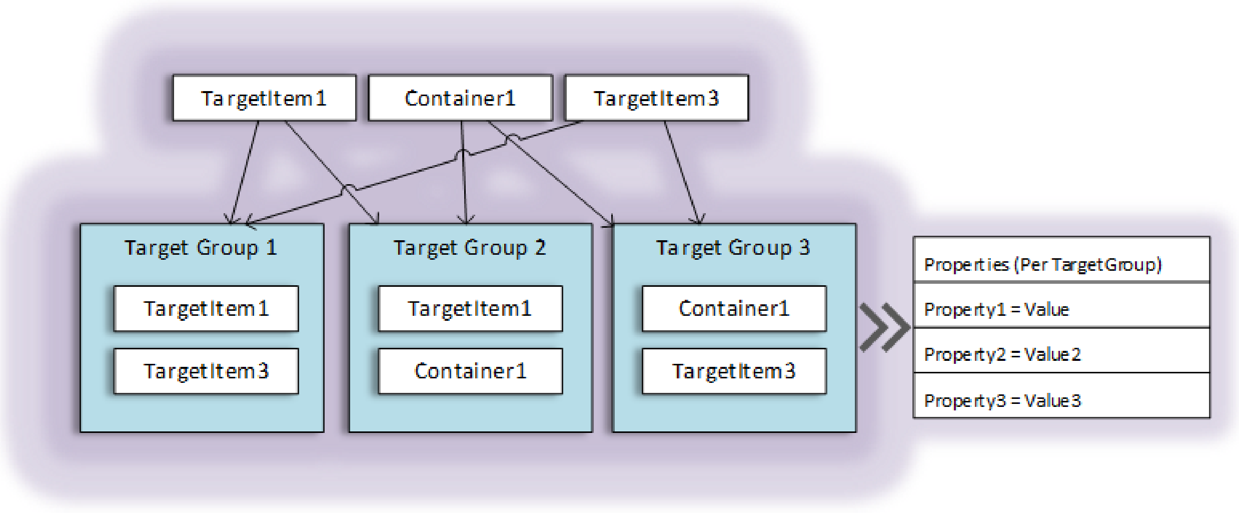
TargetGroups

TargetGroups offer the ability to relate TargetItems into a bucket for administrative purposes. TargetGroups are used to group together all items of a certain type, or all items based on a specific criterion. This provides administrative capabilities to assign tasks to a large number of TargetItems with whatever common criteria are required by the business case. (E.g., "All Servers" or "All Dell devices", etc.) Custom TargetGroup properties are stored in an item property bag.

⚠ IMPORTANT

That TargetGroups do NOT offer the same scoping functionality that Containers do for the purposes of coordinating Tasks. TargetGroups provide an administrative way to manage a bundle of TargetItems. Each is treated individually and iteratively whenever an action is taken against the group, such as assigning tasks.

Figure 3 - Containers and TargetItems within a TargetGroup, which also has properties.



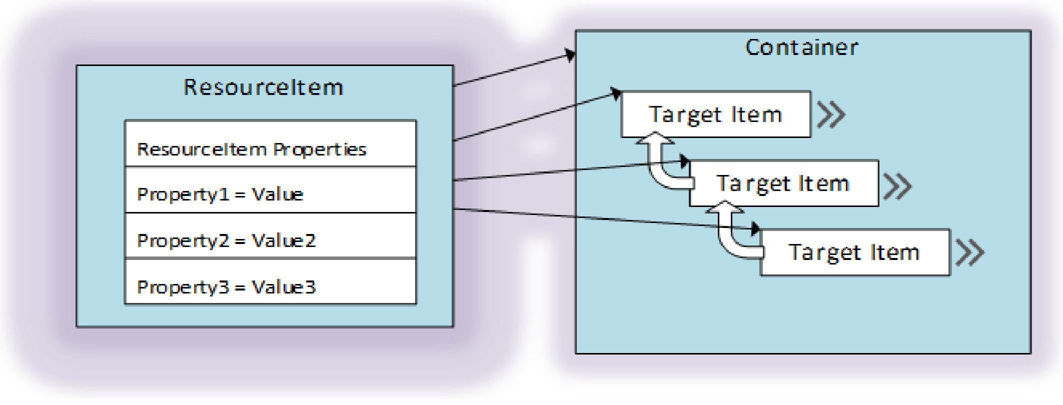
ResourceItems - Associative Resource Data

ResourceItems represent a reusable, associative piece of data for broader consumption than an individual item or container. For example, storing a list of all patches as required to be applied to Windows Servers, then associating that data to all Windows Servers in our TargetItem data. Custom ResourceItem properties are stored in an item property bag.

Resources provide this mechanism by representing the patch data as a ResourceItem (what patch, where the files are located, etc.). These details are the properties of the ResourceItem. This resource can then be associated to all of the relevant devices via a ResourceAssignment. Providing a one-to-many relationship of that single Resource to as many TargetItems as deemed applicable.

The figure below shows a simple ResourceItem and an example of the resource being 'associated' to various TargetItems or Containers.

Figure 4 - Resource item with properties, with an example of association to items/containers

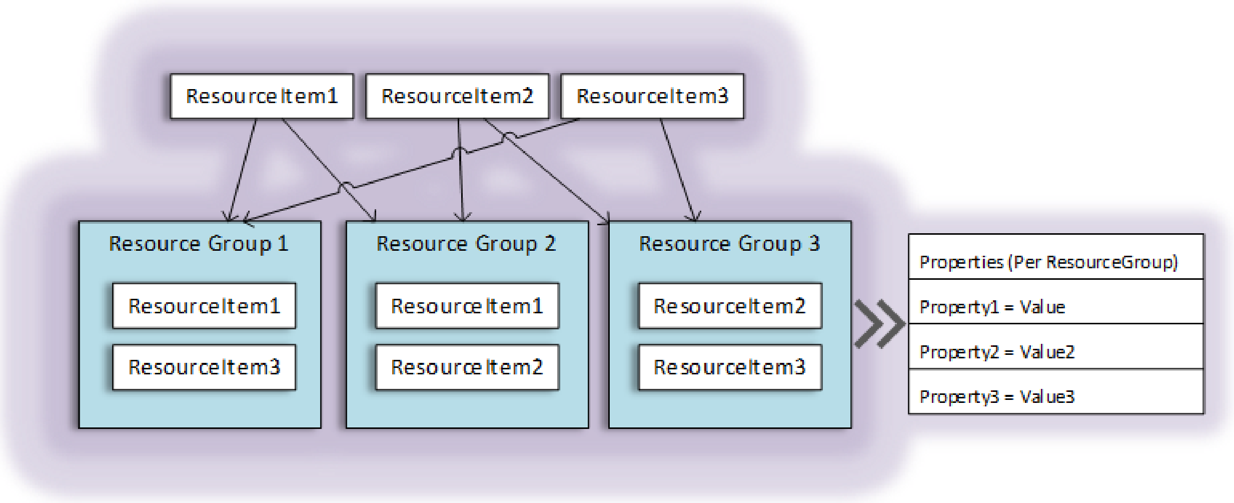


ResourceGroups

ResourceGroups offer identical functionality to TargetGroups in regards to ResourceItems. They provide a logical grouping mechanism for grouping ResourceItems together for use and consumption. As with TargetGroups, the ResourceGroup does not offer any scoping functionalities and is merely an administrative function. Custom ResourceGroup properties are stored in an item property bag.

One benefit to ResourceGroups however, given their flexibility, is that they can be used to piece together configurations from smaller sets of ResourceItems. Configurations can be stored as Resources, then added to various ResourceGroups to create different configuration sets for different purposes.

Figure 5 - ResourceGroups containing Resources, Each group also has properties



ResourceAssignment

ResourceAssignments offer a correlation mechanism between ResourceItems and TargetItems and subsequently store a status about that relationship. For example, defining a patch resource, and assigning it to a device allows for both the existence of that correlation as well as its status. Custom ResourceAssignment properties are stored in an item property bag.

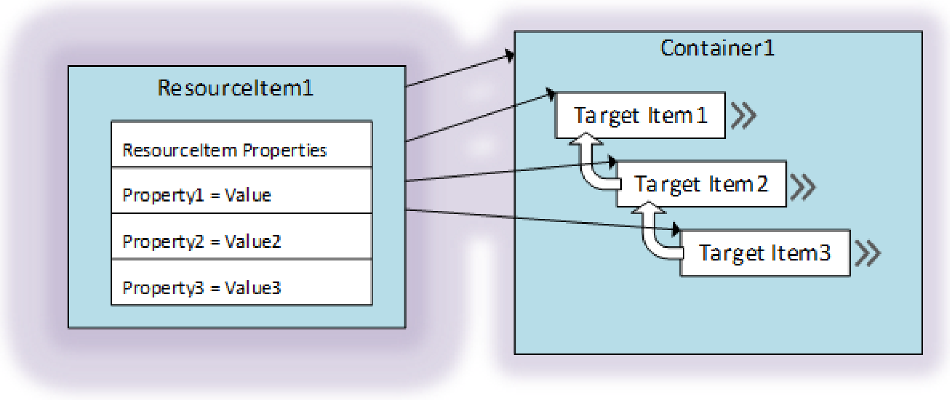
It is important to note however, the action of associating ResourceItems to TargetItems has no outward effect on automation. No execution request is performed and no action is taken simply by the action of creating this assignment. The assignment instead creates the reference between that TargetItem so it may be leveraged in code. This provides both management of what ResourceItems are assigned to what TargetItems as well as an optional AssignmentStatus for this relationship.

Using this mechanism is optional, as there may be cases where a ResourceItem does not have or require a status of the Assignment. As a result, use of ResourceAssignments are to be leveraged as needed. While one may exist, it may contain null or useless data.

Additionally, a ResourceAssignment is not required to make use of a ResourceItem in code. Storing 'global' information in a ResourceItem without assigning it to anything is an expected use-case.

The diagram below and provides context to the resulting list of ResourceAssignments that would be generated if the displayed associations were created.

Figure 6 - ResourceItem associated to multiple Targets and a container

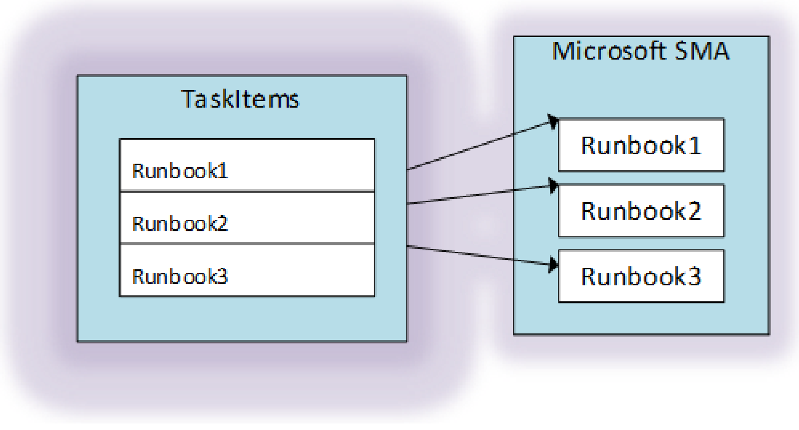


RESOURCEITEM	TARGET	STATUS
ResourceItem1	TargetItem1	Complete
ResourceItem1	TargetItem2	Incomplete
ResourceItem1	Container1	NULL
ResourceItem 1	TargetItem3	MiscStatus

TaskItems

A TaskItem is an instance of a runbook registered to the RPS database. This represents a runbook that has been imported into SMA for execution and is being added to RPS to be made available for coordinated execution by RPS. Custom TaskItem properties are stored in an item property bag.

Runbooks can exist inside of SMA without being registered to RPS for control, coordination, and execution by the automations solution. This distinction registers an item for coordinated execution by RPS.



Task Maps - Coordinating Pieces of Automations

Automations requirements vary by need from installing software to configuring or installing firmware on devices that don't necessarily run common operating systems. This makes automations tasks in general quite difficult to author due to the complexity of dependencies that exist when analyzing any specific goal. Coding these dependencies in begins to create a complex web of code that can be tedious to manage as well as a difficult to update or change.

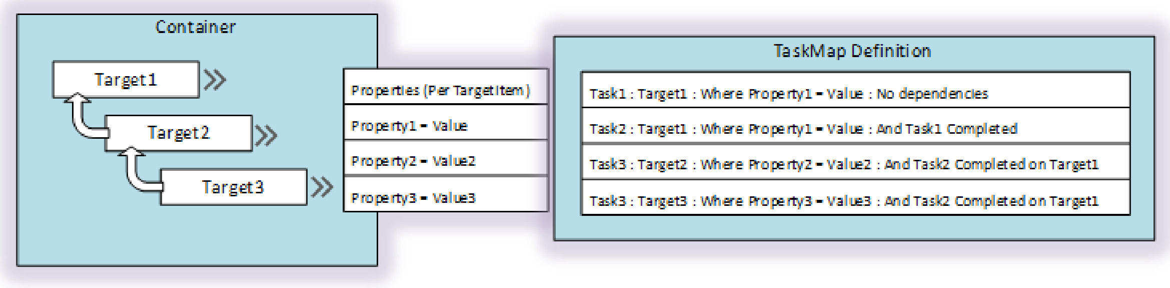
By coordinating componentized bits of code, each task can be isolated, re-used where able, and coordinated in the appropriate order or executed individually if needed. This removes the concept of authoring automations into monolithic scripts by coordinating the smaller pieces.

Each automation action's state is represented by an assignment record (see TaskAssignments) containing information about the TaskItem and TargetItem in the assignment. The status of that action/item relationship controls the flow of execution.

The RPS toolset provides this capability by managing state data about these executions and their intended target device. This allows the code to be implemented and updated quickly, while maintaining this complex web of requirements. Additions and changes become simply managing the order in which these activities are executed, or making updates to the pieces that require an update, without fear of breakage elsewhere.

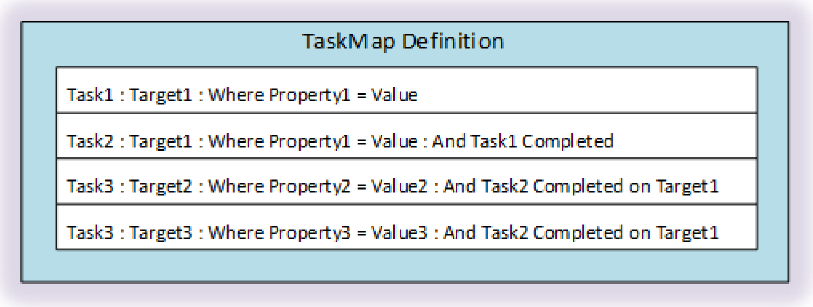
Finally, it is important to note that when executing a TaskMap a scoping mechanism was designed to allow automations that require cross-device dependencies to function. This mechanism is the "Container" as defined in the Device Metadata section of this document. Containers offer the "highest level" relationship between a set of intended targets. When developing TaskMaps which have these types of cross-device dependencies, those dependency checks are always performed within the scope of their parent Container.

Figure 7 - A visual representation of a TaskMap and how the logic is interpreted


Container
Target1
Target2
Target3
Properties (Per TargetItem)
Property1 = Value
Property2 = Value2
Property3 = Value3
TaskMap Definition
Task1 : Target1 : Where Property1 = Value : No dependencies
Task2 : Target1 : Where Property1 = Value : And Task1 Completed
Task3 : Target2 : Where Property2 = Value2 : And Task2 Completed on Target1
Task3 : Target3 : Where Property3 = Value3 : And Task2 Completed on Target1

TaskMap Definitions – Filtering on Criteria

When defining tasks for execution in a TaskMap, an optional filtering mechanism has been added to support choosing between items of similar types.


TaskMap Definition
Task1 : Target1 : Where Property1 = Value
Task2 : Target1 : Where Property1 = Value : And Task1 Completed
Task3 : Target2 : Where Property2 = Value2 : And Task2 Completed on Target1
Task3 : Target3 : Where Property3 = Value3 : And Task2 Completed on Target1

As shown in the figure above, when defining a TaskItem to be run within a TaskMap, the definition may contain a Target 'type' of device as well as provide criteria from that TargetItem's Properties to isolate specific devices. For example, if multiple "Target1" items existed in a Container, Property filters could be used to choose a specific instance of that item that meets only those criteria.

For example, a Vehicle may contain multiple computing units, but a specific computing unit may be required as a target. This method of target definition allows for control of which TargetItems receive assigned tasks.

Task and Taskmap Assignments

When a TaskItem or TaskMap is assigned to a TargetItem or Container for execution, an assignment record is generated in the RPS Database. This assignment record represents that execution request and persists the status of that request so the TaskMap can be coordinated as it is defined.

Tracking/reacting to this status data provides the core mechanism for how task maps operate as well as basic reporting information on whether or not a task was successful.

TaskMapAssignment records are persists a copy of the originating TaskMap and TaskMapDefinition along with the dependencies between definitions for automation sequencing. The TaskAssignment records are used to identify TaskItem and TargetItem execution status. Custom TaskAssignment properties are stored in an item property bag.

References

RPS Task State Diagram

The detailed design diagram for task states and how they are processed can be found here:

RPS > Trunk > Documents > Operations > **TaskStateDiagram v1.10.vsd**

> **NOTE**
>
> This document is intended to outline how task states are processed by the controller.

RPS CMDB – ERD Database Diagram

The detailed design diagram for database entity relationships (ER) can be found here:

RPS > Trunk > Documents > Architecture > **RPSEntityRelationshipDiagram.vsd**

> **NOTE**
>
> This document is intended to outline how a database entities are related to each other.

More Resources

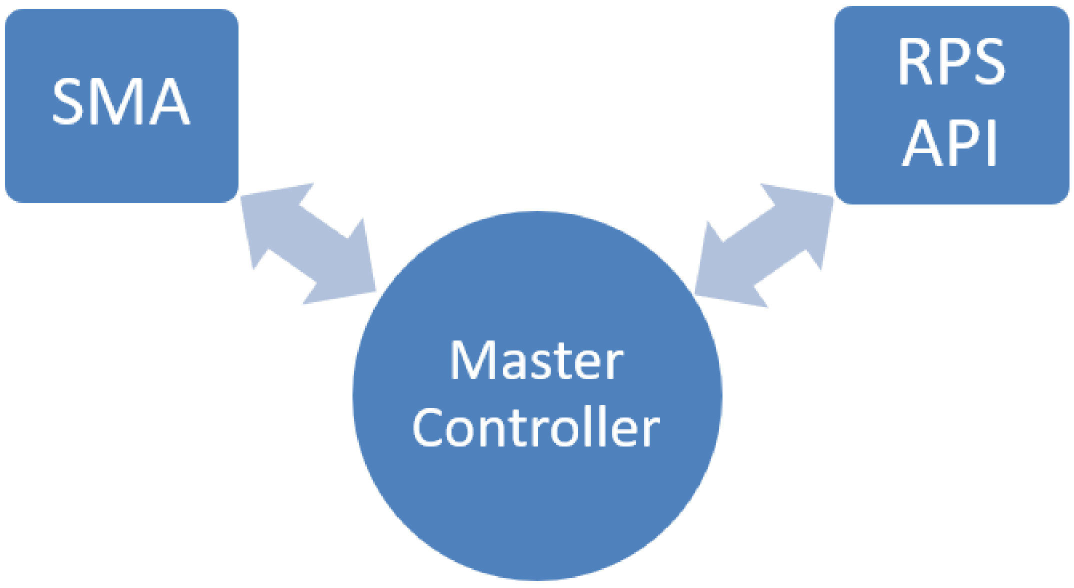
- RPS Software Design
- RPS Master Controller Design
- RPS Configuration Management (DSC) Software Design

RPS Master Controller Design

RPS Master Controller Design Overview

One of the core technologies leveraged by the Rapid Provisioning System (RPS) is Microsoft Service Management Automation (SMA). SMA allows RPS to use runbooks to automate workflow tasks and access global assets via PowerShell. While SMA is a powerful tool for automating tasks, it falls short in handling jobs that fail to execute due to network connectivity issues. One of the core requirements for RPS is to ensure it can operate within environments with low-bandwidth and intermittent connectivity.

To overcome this obstacle, the RPS team developed a Master Controller which ensures SMA jobs that fail to start or fail while executing are restarted and completed properly. The Master Controller enhances built-in SMA job execution by tracking custom RPS job states and dependencies to execute jobs in advanced sequences. It also evaluates all Task Assignments belonging to Task Maps via RPS APIs and progresses them when dependencies are met.



NOTE

This document provides an overview of the Master Controller and references technical designs, which provide additional detail to some of the components. Instead of embedding these technical design documents in this document, references to them are included to ensure that the most recent versions are always used.

TERM	DEFINITION
RPS Node	A running instance of the RPS system.
Configuration Item (CI)	A managed component, device, or appliance.
DSC	Microsoft’s PowerShell Desired State Configuration software package.
SMA	Microsoft Service Management Automation software package.
Cmdlet	A lightweight command that is used in the Windows PowerShell environment.

Task Assignments – How They are Generated from a Task Map

The referenced documents below will show how a Task Map, when assigned, will generate a set of Task Assignments. These are generated through their definitions and filters. The documents provide visual representation of how Task Assignments are generated when assigned to a container. The reference documents are available at:

```
$\Documents\Operations\**RPS TaskAssignmentDiagram.vsd**
$\Documents\Operations\**RPS Tasking Guide.docx**
```

Task States – How the Master Controller Handles Flow of Execution

The referenced document below will show how the Task Assignments resulting from a Task Map being assigned will be executed, and a visual representation of the flow the Master Controller takes to complete a Task Map's execution. These flows in general are controlled via Task States. These states indicate the next step a Task Map will make. The detailed diagram for this action is available at:

```
$\Documents\Operations\**RPS TaskAssignmentDiagram.vsd**
```

CONTROLLER "REGION"	DESCRIPTION
Invoke-Evaluation	This cmdlet triggers the algorithm that determines, based on TaskState, whether a TaskMap will proceed to the next step.
Gather Active Assignments	The cmdlet "Get-RpsActiveTaskAssignments" is used to gather all Task Assignments currently assigned to the executing node, that are both active and Ready to be launched.
Process Active Assignments in Parallel	Once all active Task Assignments are gathered for execution, they are processed in parallel threads to trigger the steps that follow (health check, execution).
Perform Health Checks as Needed	When a task is ready for execution, it will be evaluated to determine if it was previously started, and if so, if it is still in a "healthy" state and should be skipped or re-initiated.
Launch Tasks	Once health checks and states have been evaluated, if the Task in question requires an execution, it will be launched within SMA.
Correlating Tasks to Targets	Each runbook authored in support of Task Mapping/Task Item formats will have a default parameter of the TaskAssignmentId. This object correlation gives a relationship back to the Target Item, Container, and other object relationships that provide all context necessary to navigate configuration data.

Executing the Master Controller

The Master Controller is launched through a scheduled job within SMA. This schedule is configured as part of the installation process of RPS leveraging DSC. This DSC configuration maintains the SMA schedule, which launches a "health check" runbook called Check-ControllerStatus.

When started, the Check-ControllerStatus runbook determines if the Master Controller is operational, stuck, or otherwise not executing. If it is found to be unhealthy, or not executing in some way, any existing "stuck" jobs will be shut down and a new copy will be started.

NOTE

It is important that only one copy of the Master Controller runs at any given time on a given node. Multiple running Master Controller jobs can cause job or job state corruption.

Once started, the Master Controller will run on a constant loop, repeating with a default 60 second delay between completions of the entire loop. This prevents the controller from running too quickly, while remaining constantly in action.

Triggering Task Map Progression

The "Invoke-RpsEvaluateTaskAssignmentStatus" cmdlet is used to evaluate, and advance Task Maps based on their definitions and dependencies. Task State evaluation is executed using the logic shown in the document found at:

$\Documents\Operations\ *RPS TaskAssignmentDiagram.vsd*

This document outlines how Task Assignments are processed and advanced, based on the value of the Task State property.

Gathering Active Task Assignments

The "Get-RpsActiveTaskAssignments" cmdlet is used to gather all active Task Assignments. Assignments are considered active if they meet the following two conditions.

1. Task Assignment is assigned to a Target Item on the current RPS Node.
2. Task Assignment is considered "Active". See the below for more detail.

Active Definition

For a Task Assignment to be considered "Active" by RPS, all related entities must be explicitly active, meaning their IsActive flags are set to true. This includes:

1. Task Item
2. Target Item
3. Container
4. Task Map (if used for assignment)
5. Task Definition (if used for assignment)
6. Target Group (if used for assignment)

Parallel Processing of Active Assignments

Once active assignments are gathered, they are processed in parallel by the Master Controller using .NET Workflow API commands. This provides a great improvement in performance, as many jobs will be analyzed and reviewed for execution simultaneously.

Health Checks Performed on individual Jobs

Health-checks performed by the Master Controller ensure that executed jobs within SMA are actively executing and not hung-up or otherwise failed. The health-checks and resulting actions performed are listed as follows:

- Is the job supposed to be running, and is it running? – If it is not, restart it.
- Is it not running, but has been started and failed? – Job is restarted.
- Has it been started, but is queued or otherwise pending execution? – Considered healthy.
- Is it actively running/executing? Considered healthy.

Each execution is checked each time the Master Controller is run to ensure each job is healthy, or restarted in the case it is failing. This allows for controlled failures, automated recoveries, and assumed outage scenarios.

Launch Tasks

Once a Task Assignment has passed both Active checks, and Health Checks, it will be launched within SMA. This is done by calling the "Start-SmaJob" command with the Task Item as the reference runbook. At this point, SMA takes over the execution of the task, while the Master Controller and the RPS system monitor its state.

Correlating Tasks to Targets – TaskAssignmentID

Each runbook authored in support of RPS requires a single parameter – the TaskAssignmentId. This parameter enables correlation of the runbook to its TargetItem, Container, Group, TaskMap objects, and other related entities. By providing the

TaskAssignmentID, these entities can be retrieved from the RPS database to provide configuration data on a per-execution basis, as needed and at scale.

User Interactions using PendingUserActions

"New-RpsPendingUserAction" is a cmdlet designed to flag a Task Assignment for user interaction. This functionality provides a common way to initiate user-interactive requests on as-needed basis. When this command is executed, it will require a TaskAssignmentId, Button labels for Positive/Negative user replies, and a Message to display. This data can be used by the UI or API to request feedback. If the negative action is chosen, the TaskAssignmentId requesting an action will be set to "cancelled," and any executions that depend on its completion will not be executed. If the positive action is chosen, the TaskAssignmentId requesting an action will be set to Completed, and any steps depending on this task will consider it complete and proceed to any following steps.

More Resources

- RPS Software Design
- RPS Data Persistence (CMDB) Software Design
- RPS Configuration Management (DSC) Software Design
- RPS Task Assignment Diagram
- RPS Tasking Guide

RPS Configuration Management (DSC) Software Design

This guide shows the underlying RPS framework to specify, setup and maintain a desired machine configuration for the Windows computers, including RPS itself, through the RPS API and Desired State Configuration (DSC).

Configuration Management using DSC

PowerShell Desired State Configuration (DSC) is a built-in capability of most Windows devices used to manage discrete configuration of a device or system. Additionally, it has direct support for Linux systems and non-OS devices by proxy of a Windows Device (e.g. Routers).

RPS facilitates a process in which DSC configurations can be correlated to devices. This allows for the management of configurable devices in a consistent manner.

This section will outline the suggested structure for authoring DSC configurations.

TERM	DEFINITION
RPS Node	A running instance of the RPS system.
DSC	Microsoft PowerShell Desired State Configuration.

DSC Partial Configurations

DSC allows for configurations to be structured as Partial Configurations which represent small configurations for individual pieces of configuration data. By authoring DSC configurations for delivery in RPS into smaller Partial configurations, they can be pieced together and rearranged in new ways for delivery in multiple configuration sets. This modular approach greatly improves the reusability of individual configurations.

The following link provides additional information on DSC Partial Configurations: https://msdn.microsoft.com/en-us/powershell/dsc/partialconfigs

DSC Resources

If a DSC partial configuration contains the information of what the configuration will be, a DSC resource contains the information of how the configuration is set. RPS uses DSC resources from the open-source community and the RPS team itself.

NOTE

The RPS documentation includes information on authoring or adding new DSC resources to the solution. This documentation can be found at $/Documents/Operations/Authoring DSC Resources.docx.

The RPS DSC Structure

The RPS API structure used by DSC is shown below in a distilled format. This structure allows us to develop generic DSC partial configurations that can be combined in many ways using resource groups and applied to any number of target items.

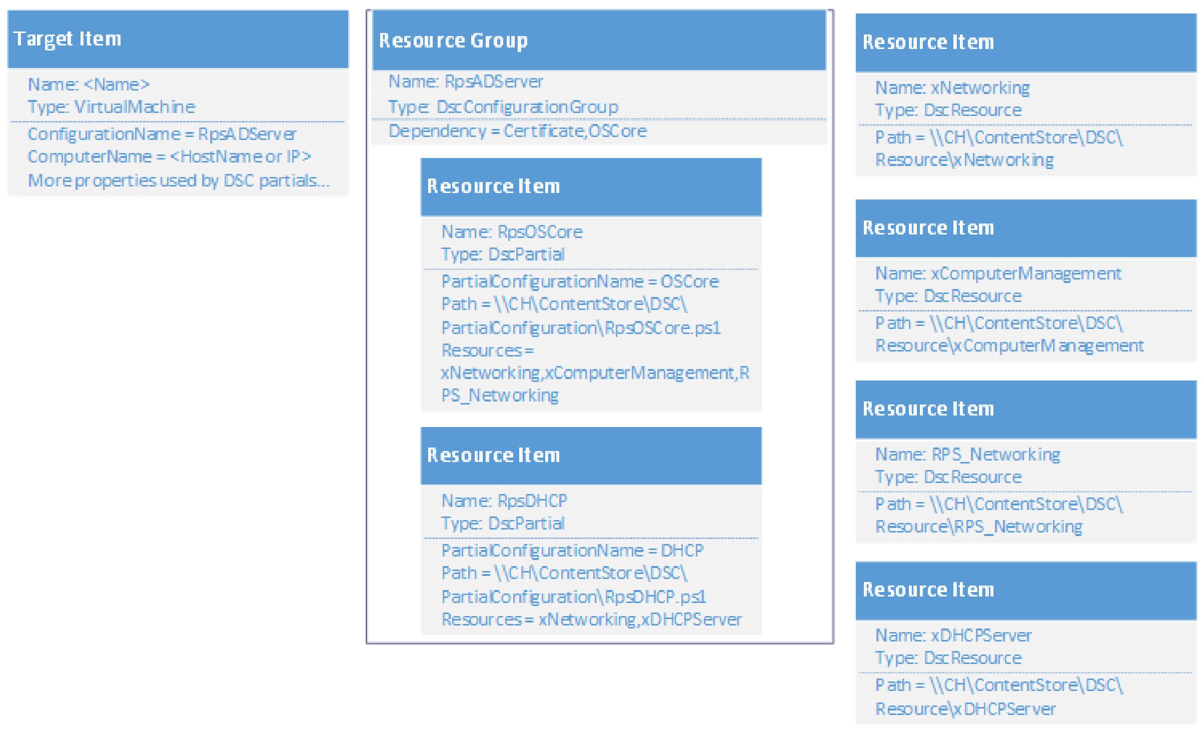

Target Item
Name: <Name>
Type: VirtualMachine
ConfigurationName = RpsADServer
ComputerName = <HostName or IP>
More properties used by DSC partials...
Resource Group
Name: RpsADServer
Type: DscConfigurationGroup
Dependency = Certificate,OSCore
Resource Item
Name: RpsOSCore
Type: DscPartial
PartialConfigurationName = OSCore
Path = \\CH\ContentStore\DSC\PartialConfiguration\RpsOSCore.ps1
Resources = xNetworking,xComputerManagement,RPS_Networking
Resource Item
Name: RpsDHCP
Type: DscPartial
PartialConfigurationName = DHCP
Path = \\CH\ContentStore\DSC\PartialConfiguration\RpsDHCP.ps1
Resources = xNetworking,xDHCPServer
Resource Item
Name: xNetworking
Type: DscResource
Path = \\CH\ContentStore\DSC\Resource\xNetworking
Resource Item
Name: xComputerManagement
Type: DscResource
Path = \\CH\ContentStore\DSC\Resource\xComputerManagement
Resource Item
Name: RPS_Networking
Type: DscResource
Path = \\CH\ContentStore\DSC\Resource\RPS_Networking
Resource Item
Name: xDHCPServer
Type: DscResource
Path = \\CH\ContentStore\DSC\Resource\xDHCPServer

Figure 1: Example RPS items for DSC use.

Each target item configured by DSC is required to be of **type** VirtualMachine. The property ConfigurationName is used to associate the target item with a resource group by the same name. Resource items within the group are the DSC partial configurations applied to the target item when Start-Dsc or Start-DscOffline is called. The property ComputerName is used by the DSC push process described below to connect to the target machine.

The resource group of type DscConfigurationGroup is used to group multiple partial configurations into groups for application to target items. This group type has a single optional property named Dependency. This is a semi-colon separated set of comma separated pairs naming the dependencies between DSC partial configurations.

For example, if the dependency property is “Certificate,OSCore;SQL,OSCore” a dependency for both the Certificate and SQL partial would be set on the OSCore partial.

The resource items of **type** DscPartial represent individual DSC partial configurations. Two of the properties, PartialConfigurationName and Path, are required with Resources being optional. Any resources used by the DSC partial listed as a comma separated list within the Resources property will be copied to the target item computer by the Start-Dsc Runbook or the Start-DscOffline script.

Finally, the resource item of **type** DscResource represents the DSC resources used by the partial configurations. Only the Path property is required.

Nearly all the DSC partial configurations have specific required parameters. Each of these expected parameters should be a property on all the target items using the partial configuration.

Start-Dsc and Start-DscOffline will use the properties on the target item as the parameters for call the DSC partial configurations. If the target item is missing a required property, the scripts will check higher memory scopes (This allows for temporary development environments with modified settings). If no property or variable is found matching all required parameters an exception will be thrown.

Compiling and Pushing Partial Configurations

When utilizing the DSC solution described in this document, there is a difference between how these DSC configurations can be used in an offline (RPS is not yet installed) scenario versus an “Online” (RPS is present) scenario.

The reason for this is, the same process used by RPS to push and configure DSC routines is the same process used to install RPS.

By building the system this way it creates a circular flow that is self-supporting. This section will describe the differences between the online and offline implementations of RPS pushing DSC configurations. Simply put, the differences in functionality reside simply in where configuration data is sourced from.

Online

The “Online” version of pushing DSC with RPS consists of a series of Powershell runbooks (workflows) authored as Task Items in RPS. These runbooks are:

- Start-Dsc
- Test-Dsc

When Start-DSC is assigned to a TargetItem through the normal processes of RPS, the configurations are pulled and compiled as described previously in this document. By creating those groups of partials, the system can pair up TargetItem property data with required parameters from the database directly and execute the DSC push. The location in which the push is targeted is defined by the TargetItem referenced.

Currently, each targeted item used in this process must have a ComputerName property with appropriate data to supply connectivity information (such as an IP address, or host name).

In this mode, the data is sourced straight from the RPS Node CMDB database that is pushing the Start-DSC Task as an executed item.

The Test-Dsc Runbook is used to provide information on the current state of the DSC configuration on the target computer.

Offline

The “Offline” version of pushing DSC with RPS consists of a series of scripts that execute in the same manner described in the Online mode, however they source their data from local files. The scripts in this use case are:

- Start-DSCOffline
- Invoke-RpsInitialization

The XML files supplied to this process are serialized using the built in RPS Commands for serialization. These XML files are created using another existing RPS installation or using the Rps-ApiMock module.

A series of XML files must be exported to support an Offline DSC operation. They must include all required data by the RPS DSC Process including:

- ResourceItem data
- ResourceGroup data
- TargetItem data

By exporting this data into XML files, Start-DSCOffline can parse them and utilize the information to execute the same way the Online use case is executed.

Invoke-RpsInitialization provides a wrapper mechanism to ensure proper files, supporting content, and metadata is present at appropriate locations before DSC Scripts are applied and begin executing the configuration process.

Building the processes this way enables RPS to self-replicate new copies to new servers, which can also maintain their own health, self-recovery and report the state of configuration.

The offline initialization process supports XML files encrypted with the Rps-Encryption module as well as plain text files.

More Resources

- RPS Software Design
- RPS Data Persistence (CMDB) Software Design
- RPS Master Controller Design

Authoring RPS DSC Partial Configurations

DSC Partial Configurations (Partial Configs) are used to organize and apply a set of configuration settings to a target computer. Multiple partial configs can be applied and combined to form the full DSC configuration applied to a target computer. RPS-enabled Partial Configs are ones that adhere to specific guidelines which allow them to be used by RPS to configure and publish configurations to computers on an RPS Node.

This document contains the guidelines to create RPS-enabled Partial Configurations.

Outline

RPS-Enabled Partial Config

What is "RPS-Enabled"?

"RPS-Enabled" indicates that a Partial Config is capable of being automatically published by RPS. In order to do this, a partial must adhere to certain rules:

1. Partial must contain baseline parameters used for secure publishing.
2. Partial must describe its dependencies and required inputs.
3. Partial and DSC Modules must be imported into RPS.

Sample Partial Config

This sample partial configuration shows the common elements of an RPS-enabled partial.

```
using namespace Rps.Api.PowerShell
using namespace Rps.Api.Constants

[DependsOn(DscPartialName = 'OSCore')]
Param
(
    [Parameter(Mandatory = $true)]
    [ValidateScript({[ipaddress]::Parse($_)})]
    [string]
    $IPAddress,

    [Parameter(Mandatory = $true)]
    [ResourceItemMapping(EntityType = [ResourceTypes]::Certificate, Role = 'DSCEncryption')]
    [Hashtable]
    $DSCEncryptionCertificate,

    [Parameter(Mandatory = $true)]
    [string]
    $OutputPath
)

Configuration SampleConfiguration
{
    Import-DscResource -ModuleName 'PSDesiredStateConfiguration' -ModuleVersion 1.1
    Import-DscResource -ModuleName 'SampleDscModule' -ModuleVersion 1.0

    Node $IPAddress
    {
        File SampleFile
        {
            Ensure = 'Present'
            SourcePath = "sourcepath"
            DestinationPath = "destinationPath"
            Force = $true
        }

        SampleDscResource SampleResourceName
        {
            Ensure = 'Present'
            Properties = Values
        }
    }
}

$certificatesPath = (Get-Item "$PSScriptRoot\..\..\Certificates").FullName

$ConfigData = @{
    AllNodes = @(
        @{
            # DSC Encryption common data
            CertificateFile = ConvertTo-RootedPath -Filename $DSCEncryptionCertificate.PublicKeyPath -
FolderPath $certificatesPath
            Thumbprint = $DSCEncryptionCertificate.Thumbprint

            NodeName = $IPAddress
            PSDscAllowDomainUser = $true
        }
    )
}

$null = SampleConfiguration -ConfigurationData $ConfigData -OutputPath $OutputPath
```

Common Parameters

All RPS Partial Configs must define the following parameters:

1. **IPAddress** - Accessible IPAddress of the computer we'll publish DSC Configuration to.
2. **DSCEncryptionCertificate** - Information about the certificate used to encrypt the mof (configuration). The LCM is set to use this certificate and any partials that are not secured will not run on a target.
3. **OutputPath** - Location to temporarily store the mof file once its compiled.

RPS-Mapped Parameters

The main advantage of using RPS-enabled partials is that they can be dynamically built from data within RPS' CMDB. When published through RPS, RPS will supply a partial config with values for all parameters from the CMDB. This system of mapping is based on a few simple conventions and parameter attributes.

For more information on RPS-Mapped Parameters see How to configure Rps-Mapped Parameters

RPS Native Parameters

RPS supports another parameter type, `[Rps.Api.TargetItem]`. If you supply a parameter of this type, RPS will return the native **TargetItem** you are publishing to. From there, you are free to use the Rps-Api module and objects to access any data.

Partial Config Dependencies

Some Partial Configs rely on others, and to optimize publishing, they need to be compiled and published in the right order.

To indicate to RPS a dependency, use the `[DependsOn]` attribute above the param block:

```
[DependsOn(DscPartialName = 'OSCore', Mandatory = $true)]
```

- DscPartialName is the name of the partial which is depended on.
- Mandatory indicates that the dependent partial is required on this target. If `$false`, this dependency is ignored if the dependent partial isn't assigned to the target.

Importing Partial Configs into RPS

Partial Configs must be imported into RPS in order to be published via RPS. When a partial is imported, RPS parses the file and stores the metadata about the partial, its parameters and its dependencies as ResourceItems. Once imported, the partial is able to be assigned to a computer, represented as a TargetItem.

Importing a Partial

- Save Partial configs in the **\DSC\PartialConfigurations\** folder.
- Save any referenced DSC Resources to the **\DSC\Modules\** folder.
- Import the Partial via the `Import-DscPartial` cmdlet, found in the Rps-Dsc module.
- Specify a folder or file for `-Path`

Example: Import all DSC Partials

```
Set-Location $ContentStore
Import-Module .\Modules\Rps-Api
Import-Module .\Modules\Rps-Dsc

Enter-RpsSession

Import-DscPartial -Path .\DSC\PartialConfigurations
```

Testing a Partial

You can test the RPS supplied configuration data for a partial by using the `Get-DscPartialParam` cmdlet. This is useful for understanding how configuration data from Targets and Resources is passed to the DSC Partial.

The cmdlet returns a `Hashtable` that can be splatted for use directly with the Partial.

Example: Get Config Data for OSCore

```
Set-Location $ContentStore
Import-Module .\Modules\Rps-Api
Import-Module .\Modules\Rps-Dsc

Enter-RpsSession

# import partial for OSCore
$partialPath = ".\dsc\PartialConfigurations"
Import-DscPartial -Path $partialPath
$osCorePartial = Set-RpsResourceItem -Type $Rps.ResourceTypes.DscPartial -Name OSCore

# create target item
$computer = Set-RpsTargetItem -Type $Rps.TargetTypes.Computer -Name "TestComputer1" `
    -Properties @{ ComputerName = "DEMO"; IPAddress = "10.0.0.1"; JoinDomain = "true" }

# set node properties
$computer.ContainerNode.SystemTimeZone = "UTC"
$computer.ContainerNode.OutputPath = "c:\temp"
$computer.ContainerNode.DomainName = "rps.master"
$computer.ContainerNode.Update()

# assign certificate
$dscCert = Set-RpsResourceItem -Type $Rps.ResourceTypes.Certificate -Name "DSCCert" `
    -Properties @{ Role = "DscEncryption"; Path = "test"; Password = "something" }
$null = $dscCert.AssignTo($computer)

# assign partial
$assignment = $osCorePartial.AssignTo($computer)

# get DSC Params
Get-DscPartialParam -PartialAssignment $assignment
```

Output

```
PS > Get-DscPartialParam -PartialAssignment $assignment
[12:55:30 INF] No credential resolved on 4ca80061-7951-4ecc-bf40-b7afcd297119: DomainAdmin
[12:55:30 INF] No value resolved on 4ca80061-7951-4ecc-bf40-b7afcd297119: ServerAdmin mapping ResourceItem in
OSCore didn't yield one match.

Name                           Value
----                           -----
SystemTimeZone                 UTC
LocalAccount                   {}
DomainName                     rps.master
ComputerName                   DEMO
DomainAdmin
DscEncryptionCertificate       {Path, Password, Role, Name...}
NetworkConfig                  {}
IPAddress                      10.0.0.1
JoinDomain                     True
IsDC                           False
OutputPath                     c:\temp\6c94d0db-235c-418b-ad48-f40944899960\OSCore
ServerAdmin
```

Assigning a Partial

Once imported, the Partial Config is saved as a ResourceItem in RPS. To publish a configuration from RPS, you must first assign one or more ResourceItems representing partials to the desired computer TargetItem.

```
Import-Module .\Modules\Rps-Api
$vm = New-RpsTargetItem -Type $Rps.TargetTypes.VirtualMachine -Name DemoVM1 -Properties { IPAddress =
"10.0.0.17" }
$softwarePartial = Get-RpsResourceItem -Type $Rps.ResourceTypes.DscPartial -Name "SoftwareDistribution"
New-RpsResourceAssignment -ResourceItem $softwarePartial -TargetItem $vm
```

Publishing a Partial

RPS Publishes DSC Partials using the **Publish-RpsConfiguration** runbook. This runbook is triggered from a fresh RPS Install, from an assigned Task or TaskMap, or via RPS Web.

Additional Tips

1. Using statements at the top improve readability. Without them, you must fully qualify Rps specific objects.

   ```
   using namespace Rps.Api.PowerShell
   using namespace Rps.Api.Constants

   [DependsOn(DscPartialName = 'OSCore')] # instead of [Rps.Api.PowerShell.DependsOn(...)]
   ```

2. ConfigurationName - Use a meaningful name for the configuration. This name identifies the Partial Config in RPS and the `DependsOn` attribute.
3. For all resources you use in your partial you will need to import the module inside the configuration block. Include the `-ModuleVersion` in your import statements to avoid issues with multiple module versions.

```
    Import-DscResource –ModuleName 'RPS_RPSApi' -ModuleVersion 1.0
```

1. `$certificatesPath` indicates the location where certificates are stored. Do not modify its value.
2. configData - This is the basic configuration for a partial. You can add parameters to this HashTable but this is the minimum. This sets up the mof encryption for the target.

Authoring RPS DSC Resources

The Rapid Provisioning System use Desired State Configuration to manage itself and other target computers to configure and prevent the drift from discrete states. A major piece of implementing DSC is the development and maintenance of Resources. This document will outline the steps for authoring and consuming DSC resources within RPS.

Using the Resource Within A Partial

To use a configuration of a resource within a partial you simply need to import the DSC resource.

For example, when needing to use the RPS_RpsApi resource add the below line within your configuration.

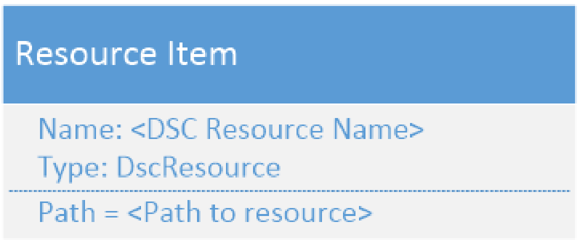
```
Configuration RPSPrivilegedAccount
{
    Import-DscResource -Module 'RPS_RpsApi'

    Node $TargetName
}
```

TERM	DEFINITION
DSC	Desired State Configuration.

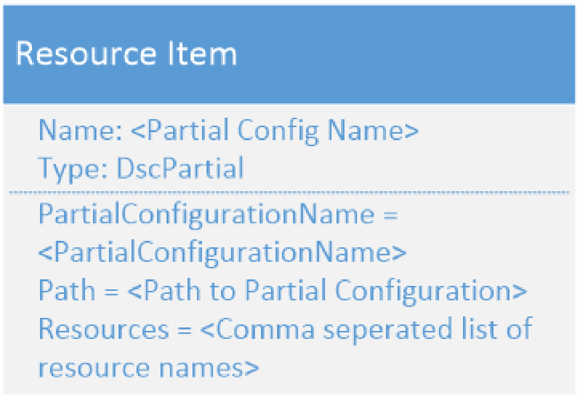
Adding the resource to the RPS CMDB

The entire RPS CMDB data structure used by DSC can be found within the RPS Configuration Management (DSC) Design document. This document can be found at $/Documents/Operations/RPS Configuration Management (DSC) Design.docx.

DSC resources are added to the CMDB as RPS resource items.



Resources used by partials are added by name as a comma-delimited list to the partial configuration Resource Item.


Resource Item
Name: <Partial Config Name>
Type: DscPartial
PartialConfigurationName = <PartialConfigurationName>
Path = <Path to Partial Configuration>
Resources = <Comma seperated list of resource names>

Using the resource within the initial Install

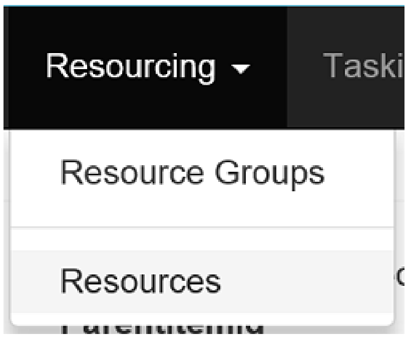
To add a DSC resource to the RPS offline build, add the new resource to the New-RpsXmlConfiguration script within the Setup directory matching the structure described in the section above.

Create a Host through Rapid Provisioning System (RPS)

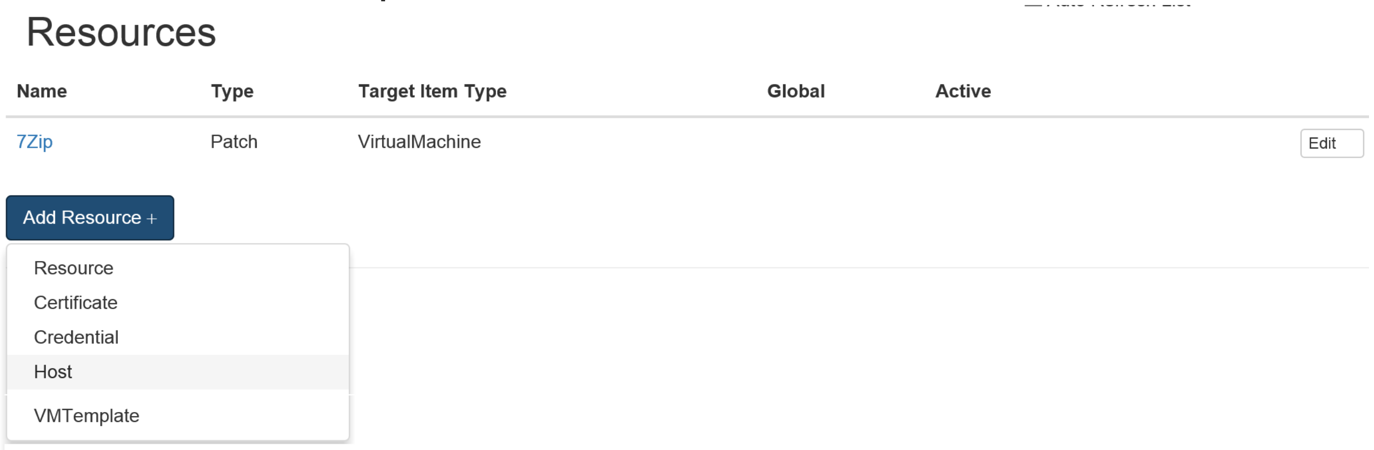
This guide shows the process to create a new Host for RPS to use during Virtual Machine deployment.

Create a Resource Item that represents the Host

1. Open a web browser to the RPS Website, for example: https://SMA.RPS.Local:8080
2. In RPS, choose **Resources** > **Resources**



3. Choose **Add Resource** > **VMTemplate**



4. Set the requested information, and then click **Save**

VMTEMPLATE SETTINGS	DESCRIPTIONS
Name (Required)	The desired name for Host in RPS
Type (Required)	Must be **Host**. Pre-populated
ComputerName (Required)	The name of the server acting as the Virtual Machine Host.
HostType (Required)	The type of host, e.g. **Hyper-V, ESXi**
Path (Required)	The path to save the virtual machines

NOTE

Click **Add Property** to add custom fields for your Credential.

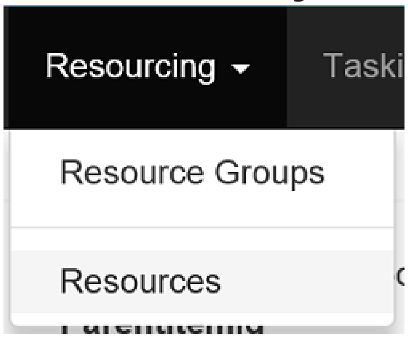
More Resources

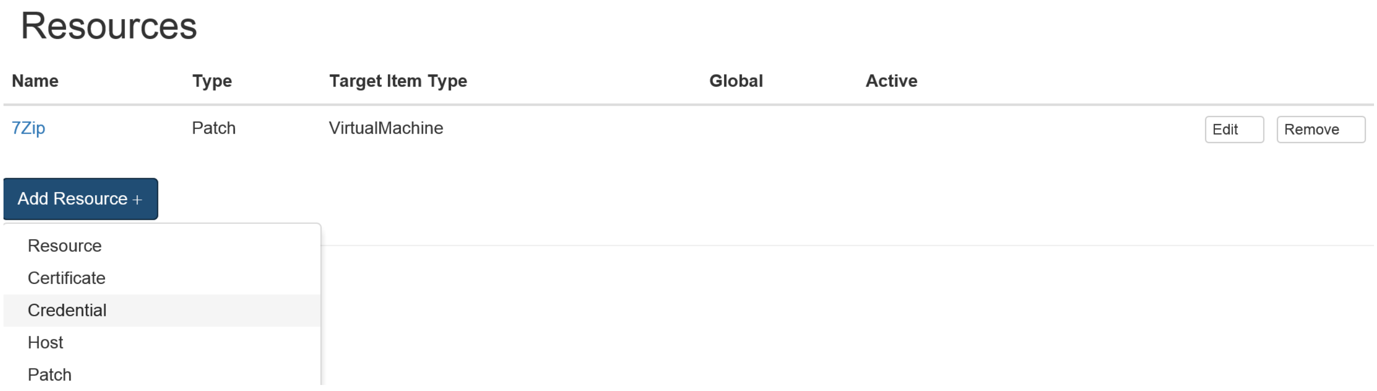
- Create a Hyper-V Virtual Machine through Rapid Provisioning System (RPS)
- Create RPS Credentials through Rapid Provisioning System (RPS)
- Create a Virtual Machine template through Rapid Provisioning System (RPS)

Create RPS Credentials through Rapid Provisioning System (RPS)

This guide shows the process to create new credentials to use in RPS.

Create a Resource Item that represent the Credential

1. Open a web browser to the RPS Website, for example: https://SMA.RPS.Local:8080
2. In RPS, choose **Resourcing** > **Resources**


3. Choose **Add Resource** > **Credential**


Name	Type	Target Item Type	Global	Active
7Zip	Patch	VirtualMachine		

4. Set the requested information, and then click **Save**

VMTEMPLATE SETTINGS	DESCRIPTIONS
Name (Required)	The name you want to give the Credential in RPS
Type (Required)	Must be **Credential**. Pre-populated
UserName (Required)	The UserName of the credential
Password (Required)	The Password of the credential. This password can be automatically generated by pressing the Generate Password button.
Roles (Required)	The role the credential will be used for. Multiple roles can be added, separated by a '|'

NOTE

Click **Add Property** to add custom fields for your Credential.

More Resources

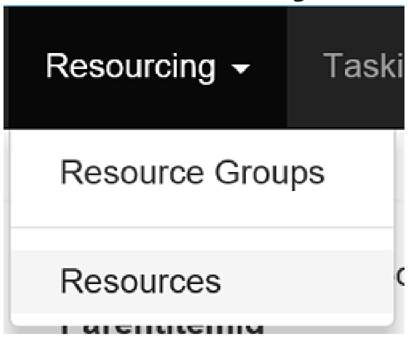
- Create a Host through Rapid Provisioning System (RPS)
- Create a Hyper-V Virtual Machine through Rapid Provisioning System (RPS)

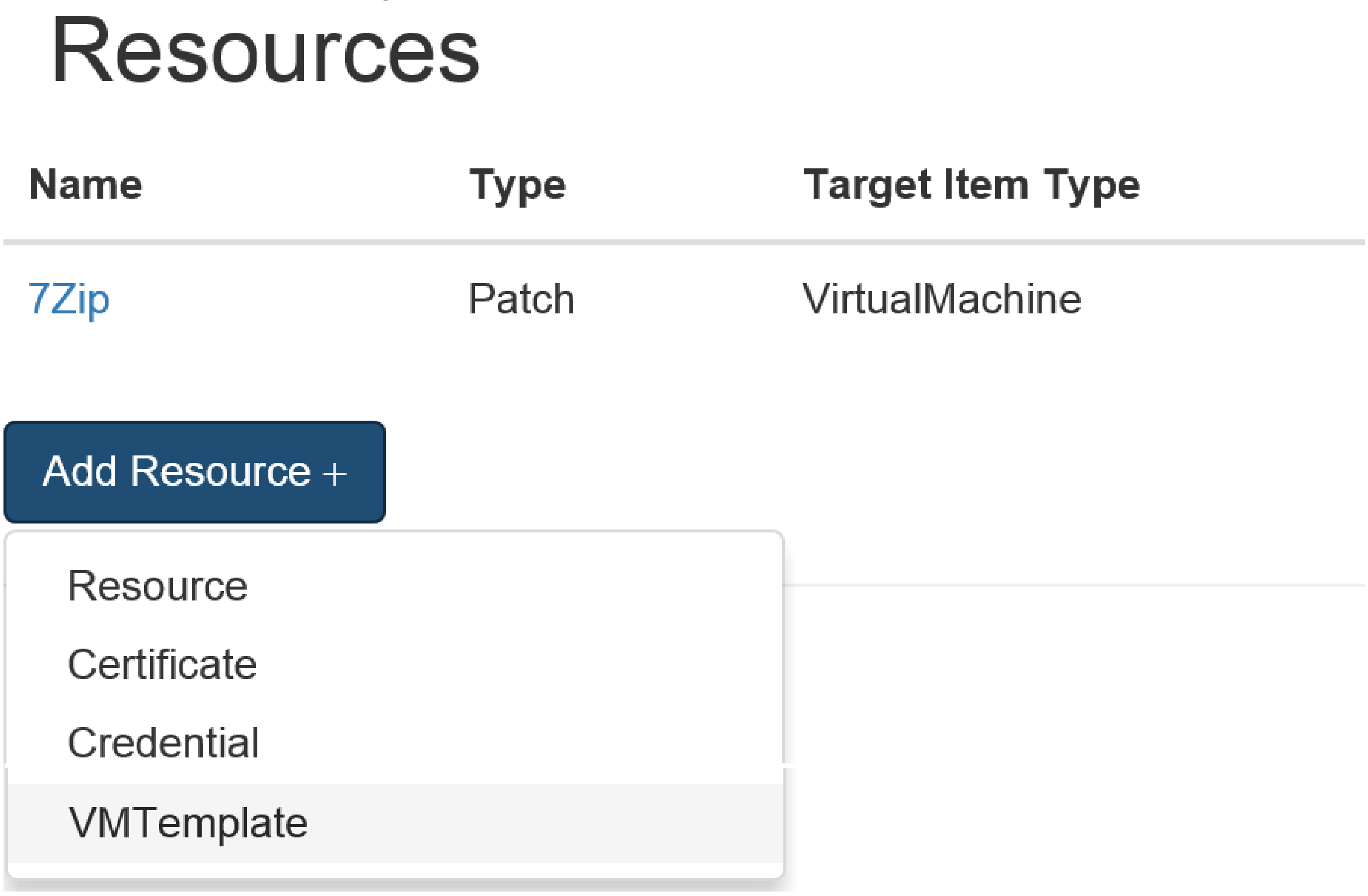
- Create a Virtual Machine template through Rapid Provisioning System (RPS)

Create a Virtual Machine template through Rapid Provisioning System (RPS)

This guide shows the process to create a new Virtual Machine template for RPS to use during Virtual Machine deployment.

Create a Resource Item that represent the Virtual Machine

1. Open a web browser to the RPS Website, for example: https://SMA.RPS.Local:8080
2. In RPS, choose **Resourcing** > **Resources**


3. Choose **Add Resource** > **VMTemplate**


Resources

Name	Type	Target Item Type
7Zip	Patch	VirtualMachine

4. Set the requested information, and then click **Save**

VMTEMPLATE SETTINGS	DESCRIPTIONS
Name (Required)	The desired name for the VMTemplate in RPS
Type (Required)	Must be **VMTemplate**. Pre-populated
Path (Required)	The path to the .vhdx file, relative to Content Store

NOTE

Click **Add Property** to add custom fields for your Virtual Machine Template.

More Resources

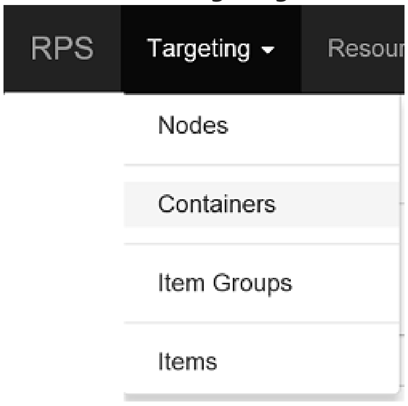
- Create a Host through Rapid Provisioning System (RPS)
- Create RPS Credentials through Rapid Provisioning System (RPS)
- Create a Hyper-V Virtual Machine through Rapid Provisioning System (RPS)

Create a Hyper-V Virtual Machine through Rapid Provisioning System (RPS)

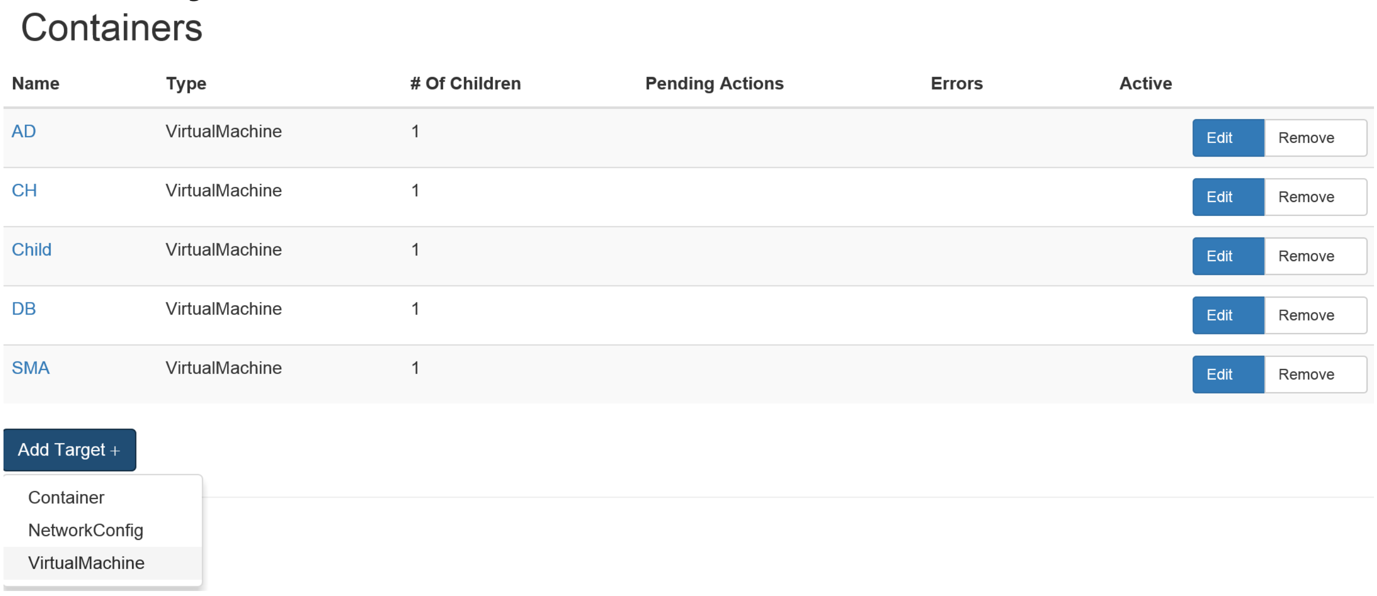
This guide shows the process to create a new Target Item to deploy as a Hyper-V Virtual Machine.

Create a Target Item that represents the Virtual Machine

1. Open a web browser to the RPS Website, for example: https://SMA.RPS.Local:8080
2. In RPS, choose **Targeting** > **Containers**



3. Choose **Add Target** > **Virtual Machine**



Containers

Name	Type	# Of Children	Pending Actions	Errors	Active	
AD	VirtualMachine	1				Edit Remove
CH	VirtualMachine	1				Edit Remove
Child	VirtualMachine	1				Edit Remove
DB	VirtualMachine	1				Edit Remove
SMA	VirtualMachine	1				Edit Remove

4. Set the requested information, and then click **Save**

VIRTUAL MACHINE SETTINGS	DESCRIPTIONS
Name (Required)	The desired name to give the Target Item (Virtual Machine) in RPS
Type (Required)	Must be **VirtualMachine**. Pre-populated
ComputerName (Required)	The name of the computer as it is in Active Directory
MemoryMB (Required)	The amount of memory the machine needs in **MB**
OSType (Required)	The Virtual Machines operating system, e.g. **Windows**
OSVersion (Required)	The version on the operating system, e.g. **8.1** for Windows Server 2012 R2

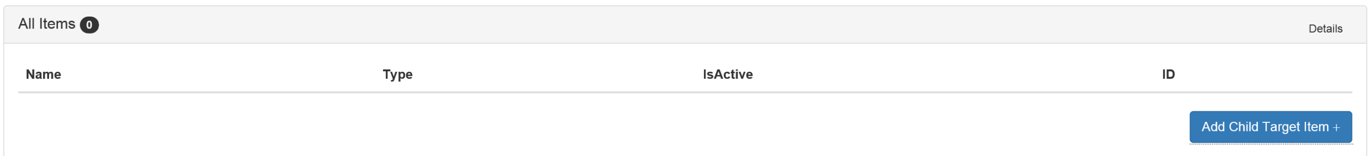
VIRTUAL MACHINE SETTINGS	DESCRIPTIONS
Architecture (Required)	The architecture of the Virtual Machine, e.g. **x86, x64**
IsCDN	[Boolean] Set to true if this Virtual Machine will have RPS role **Content Delivery Network**
IsSMA	[Boolean] Set to true is this Virtual Machine will have RPS role **Service Manager Automation (SMA)**
IsDB	[Boolean] Set to true if this Virtual Machine will have RPS role **SQL Server Database**
IsDC	[Boolean] Set to true if this Virtual Machine will have RPS role **Active Directory Domain Controller**

NOTE

Click **Add Property** to add custom fields for the Virtual Machine.

Create a child Target Item that represents the Network Configuration

1. Click on the new Virtual Machine
2. Under All Items, choose **Add Child Target Item**.



3. Set the requested information, then click **Save**.

NOTE

To add properties, click **Add Property** button. The required fields must be added to create a New Hyper-V Virtual Machine. The next update will use a template to pre-populate the fields.

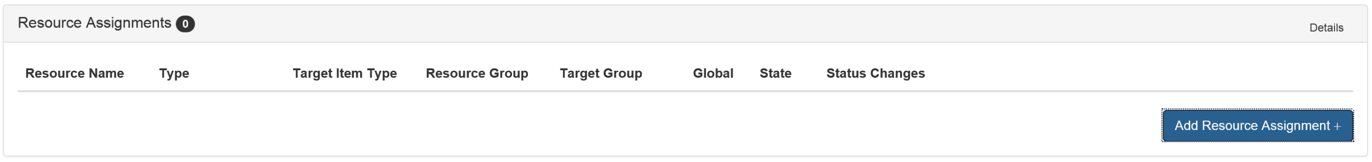
NETWORK CONFIGURATION SETTINGS	DESCRIPTIONS
Name (Required)	The desired name to give the Network Adapter in RPS
Type (Required)	Must be **NetworkConfiguration**
Alias (Required)	The name of the Network Adapter on the Host you are creating the Virtual Machine
IpAddress (Required)	The IP Address of the Virtual Machine
Subnet (Required)	The subnet of the Virtual Machine
DnsServer (Required)	The IP Address of the DNS Server
MacAddress	The desired MAC Address for the Virtual Machine. If left blank, the Virtual Machine will get a dynamic MAC Address
DHCP	The IP Address of the DHCP server

NOTE

Click **Add Property** to add custom fields for your Virtual Machine.

Add the Hyper-V Virtual Machine Template

1. Under **Resource Assignments**, click **Add Resource Assignment**.



Resource Name	Type	Target Item Type	Resource Group	Target Group	Global	State	Status Changes

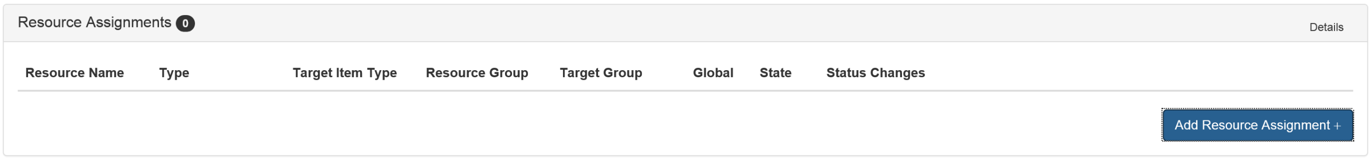
2. Click on the **Resource Assignment** dropdown, choose the Resource Item that represents the Virtual Machine Template. Click **Save**.

NOTE

The Resource Item will have a type of **VMTemplate**. If the desired Virtual Machine template is not visible, then see: **New Virtual Machine Template Guide**.

Add the desired Hyper-v Host

1. Under **Resource Assignments**, click **Add Resource Assignment**.



Resource Name	Type	Target Item Type	Resource Group	Target Group	Global	State	Status Changes

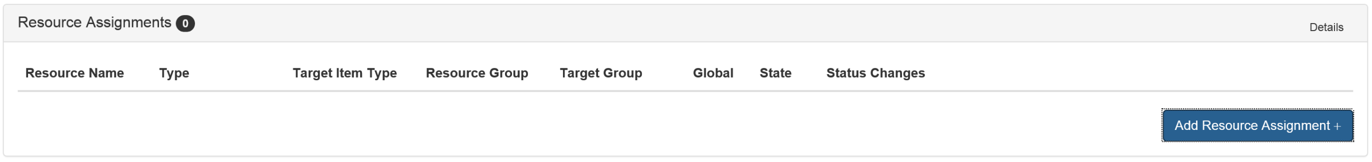
2. Click on **Resource Assignment** drop down, choose the Resource Item that represents the Host. Click **Save**.

NOTE

Resource Item will have a type of **Host**. If you do not see the Host you want to use then you can create your own following the **New Hyper-V Host Guide**.

Add Local Administrator Credentials

1. Under **Resource Assignments**, click **Add Resource Assignment**.



Resource Name	Type	Target Item Type	Resource Group	Target Group	Global	State	Status Changes

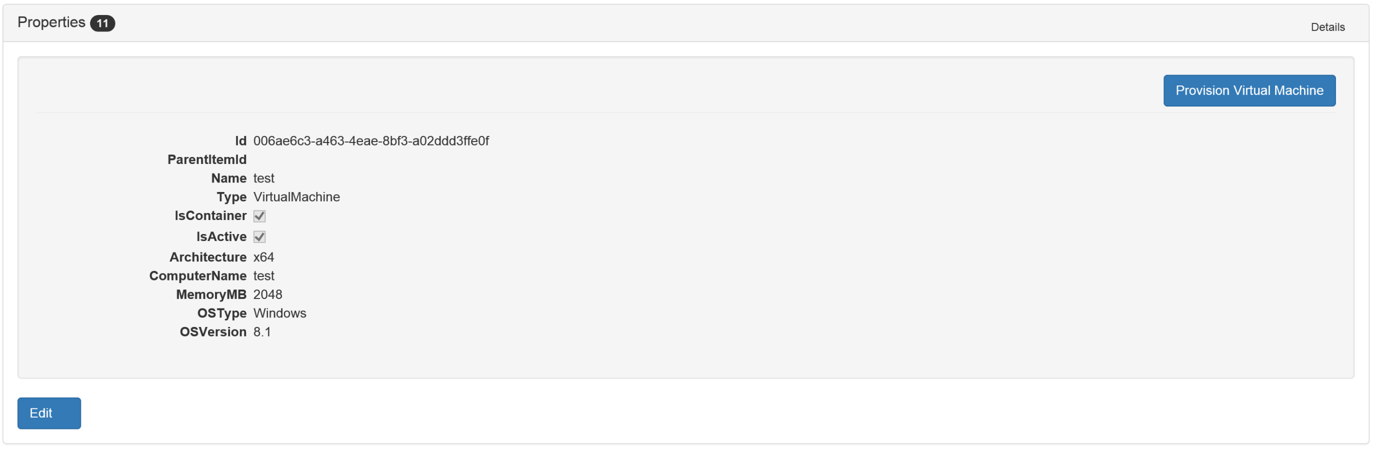
2. Click on **Resource Assignment** drop down, choose the Resource Item that represents the Local Admin Credentials. Click **Save**

NOTE

Resource Item will have a type of **Credential**. If you do not see the Credential you want to use then you can create your own following the **New RPS Credential Guide**.

Provision the Virtual Machine

1. Under **Properties**, click **Provision Virtual Machine**.



Id	006ae6c3-a463-4eae-8bf3-a02ddd3ffe0f
ParentItemId	
Name	test
Type	VirtualMachine
IsContainer	✓
IsActive	✓
Architecture	x64
ComputerName	test
MemoryMB	2048
OSType	Windows
OSVersion	8.1

NOTE

This action creates a new Task to create the VM on the assigned Hypervisor Host. RPS will begin processing the task in the background. Refresh the view to see the status of the task.

More Resources

- Create a Host through Rapid Provisioning System (RPS)
- Create RPS Credentials through Rapid Provisioning System (RPS)
- Create a Virtual Machine template through Rapid Provisioning System (RPS)

RPS Building iPXE ROMs

This guide shows how to configure an iPXE ROM with the specific options/features needed to be used within RPS.

Process Overview

The following steps are required to build a iPXE ROM that is trusted by the RPS root certificate:

- Install prerequisites
- Download iPXE source
- Build ROM

Prerequisites

- iPXE source
- Admin Workstation
- Linux workstation such as Ubuntu OR WSL (Windows Subsystem for Linux)
- Linux packages
- GCC
- binutils
- make
- syslinux (required for building ISOs)
- genisoimage (required for building ISOs)
- liblzma

- RPS public root certificate, Base64 encoded

  NOTE

  Most of the tools required to build an iPXE ROM are Linux based and can only be executed from Bash (Unix Shell). However, a Unix operating system is not required. The process in this document has been completed leveraging WSL (Windows Subsystem for Linux). WSL is available on a machine running 64-bit version of Windows 10 Anniversary Update build 14393 or later. If the development machine that the ROM is being built from does not have the required prerequisites an internet connection will be required to install them.

Installing the Tools in Bash

1. Update the package list, **sudo apt update**
2. Install GCC, **sudo apt install gcc**
3. Install binutils, **sudo apt install binutils**
4. Install make, **sudo apt install make**
5. Install syslinux, **sudo apt install syslinux**
6. Install genisoimage, **sudo apt install genisoimage**
7. Install liblzma, **sudo apt install liblzma-dev**
8. Install git, **sudo apt install git**

Download iPXE source and Build ROM that Trusts the RPS CA Root Certificate

1. Navigate to the directory the source will be downloaded to and clone iPXE repository: **git clone git://git.ipxe.org/ipxe.git**
2. After source is downloaded navigate to ipxe/src: **cd ipxe/src**

3. To build a ROM that trusts the RPS root certificate run the following:

```
make bin-x86_64-efi/ipxe.efi TRUST=<path to certificate> CERT=<path to certificate>
```

b. Here is an example of building an iPXE, EFI compatible, ROM with the RPS certificate in the local folder.

```
make bin-x86_64-efi/ipxe.efi TRUST=RPSbase64Ca.cer CERT=RPSbase64Ca.cer
```

> **NOTE**
>
> The ROM will be in the bin-x86_64-efi folder.

iPXE ROM Build Command Examples

All the examples are executed from the ipxe/src folder.

1. Create an iPXE bootable ISO that trust the RPS root cert

```
make bin/ipxe.iso TRUST=RPSbase64Ca.cer CERT=RPSbase64Ca.
```

2. Create iPXE ROMs for all the compatible ESXi network adapters

```
make bin/8086100f.mrom bin/808610d3.mrom bin/10222000.rom bin/15ad07b0.

certutil.exe -encode CaRootCert.cer base64Ca.cer
```

> **NOTE**
>
> Ensure the RPS root cert is Base64 encoded. If not run the following command from a Windows environment:

More Resources

- RPS PXE Document
- Configuring ESXi VMs to Use iPXE

RPS PXE

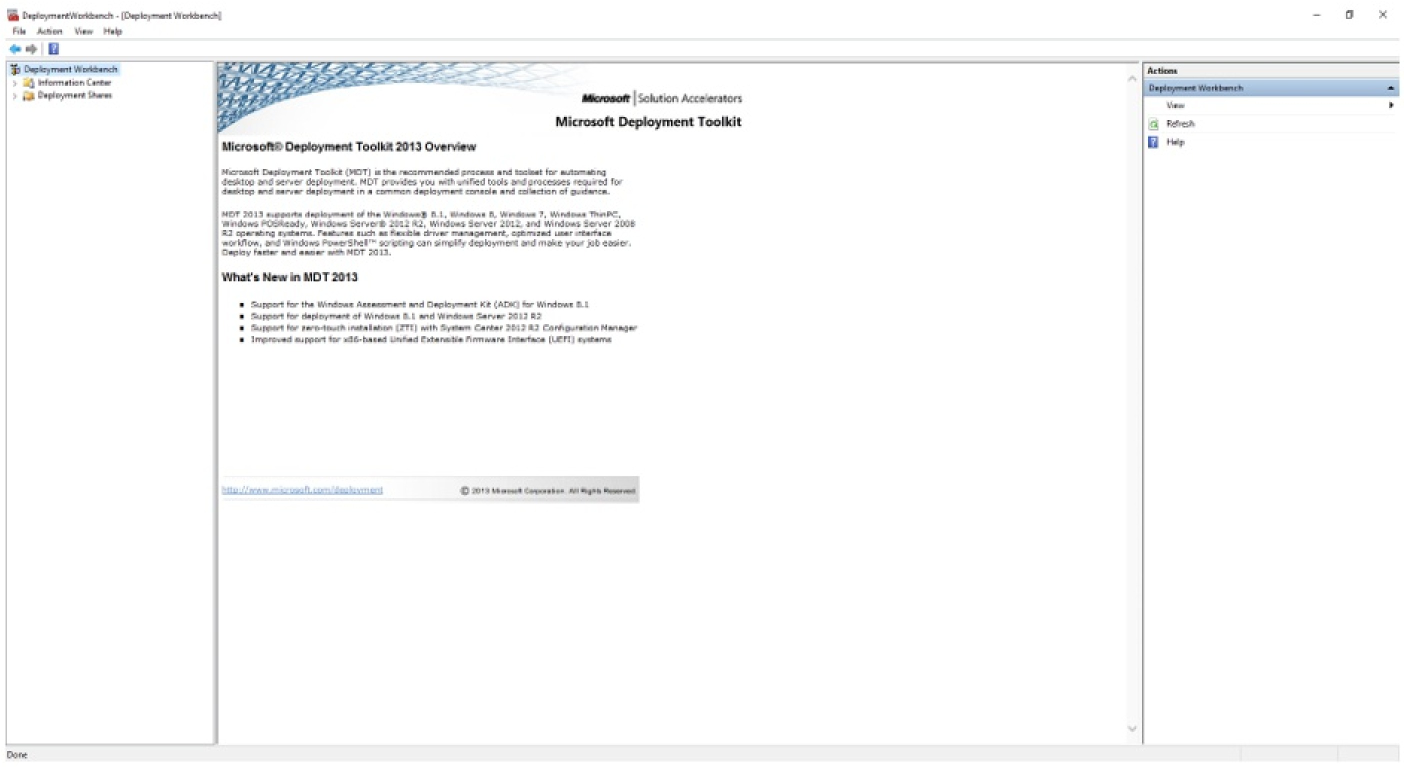
One of the major features of the Rapid Provisioning System (RPS) is to allow users of the system to perform bare metal provisioning of computing devices through the use of a Pre-Boot Execution Environment (PXE). The decision to use a basic installation of Windows Deployment Services and Microsoft Deployment Toolkit was made to allow for the smallest possible footprint, as well as for ease of use. Additionally, some environments that utilize RPS are very unpowered and overprovisioned, and simply do not have the resources available for a major component such as System Center Configuration Manager (SCCM) to be dedicated to perform deployments.

MDT and creating a WIM

Two components are required for setting up the Microsoft Deployment Toolkit (MDT) environment, the Deployment Toolkit itself, and the Windows Assessment and Deployment Kit (ADK) which has all of the necessary tools required to allow a user to create and customize a deployment environment using MDT.

- MDT can be downloaded from the following link: https://www.microsoft.com/en-us/download/details.aspx?id=50407
- The ADK can be downloaded from the following link: https://www.microsoft.com/en-us/download/details.aspx?id=39982

The ADK needs to be installed first, and then MDT can be installed. Once it is installed, open the program called "Deployment Workbench". The interface looks like the following:



Microsoft Deployment Toolkit

Microsoft® Deployment Toolkit 2013 Overview

Microsoft Deployment Toolkit (MDT) is the recommended process and toolset for automating desktop and server deployment. MDT provides you with unified tools and processes required for desktop and server deployment in a common deployment console and collection of guidance.

MDT 2013 supports deployment of the Windows® 8.1, Windows 8, Windows 7, Windows ThinPC, Windows POSReady, Windows Server® 2012 R2, Windows Server 2012, and Windows Server 2008 R2 operating systems. Features such as flexible driver management, optimized user interface workflow, and Windows PowerShell™ scripting can simplify deployment and make your job easier. Deploy faster and easier with MDT 2013.

What's New in MDT 2013

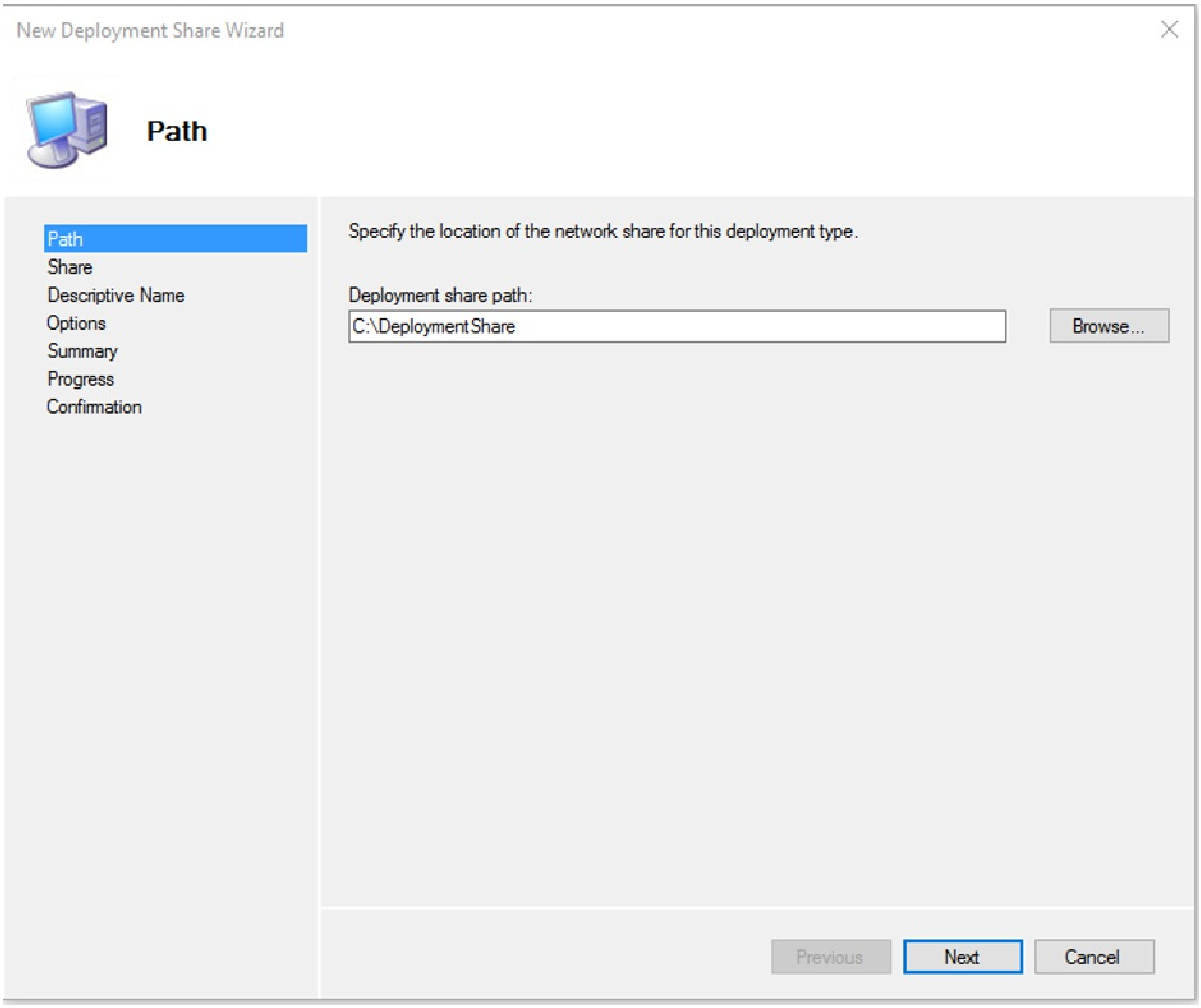
- Support for the Windows Assessment and Deployment Kit (ADK) for Windows 8.1
- Support for deployment of Windows 8.1 and Windows Server 2012 R2
- Support for zero-touch installation (ZTI) with System Center 2012 R2 Configuration Manager
- Improved support for x86-based Unified Extensible Firmware Interface (UEFI) systems

http://www.microsoft.com/deployment © 2013 Microsoft Corporation. All Rights Reserved

Figure 1 Deployment Workbench

Creating a Deployment Share

The first step in setting up the environment is to create a new deployment share. To do this, right click on "Deployment Shares" in the left pane and select "New Deployment Share". The following wizard will appear:


New Deployment Share Wizard

Path

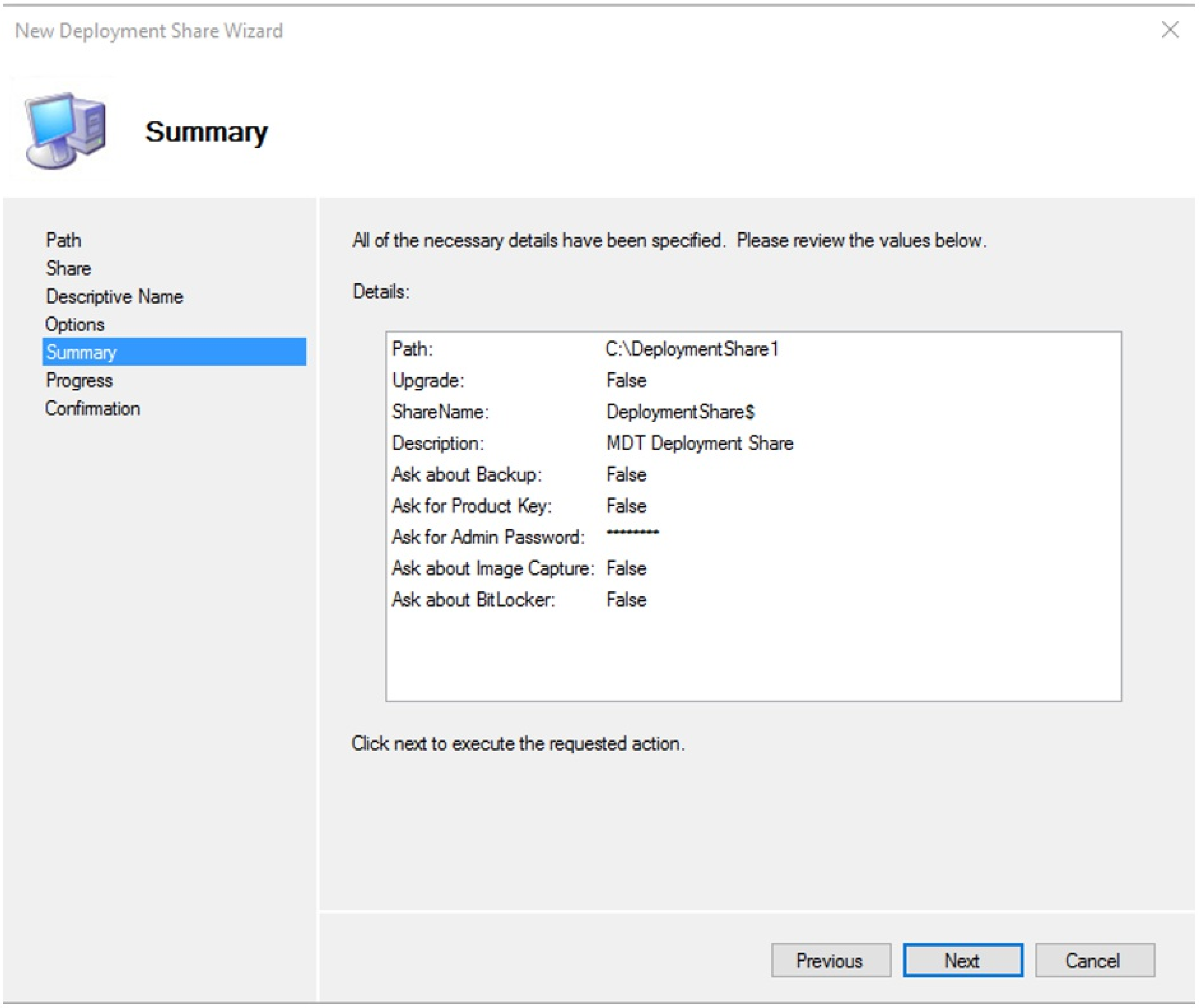
Specify the location of the network share for this deployment type.

Deployment share path:

C:\DeploymentShare

Figure 2 New Deployment Share

Continue through the wizard, and at the options screen, uncheck all of the options. The summary screen before the Deployment Share gets created should look like this:


New Deployment Share Wizard

Summary

All of the necessary details have been specified. Please review the values below.

Details:

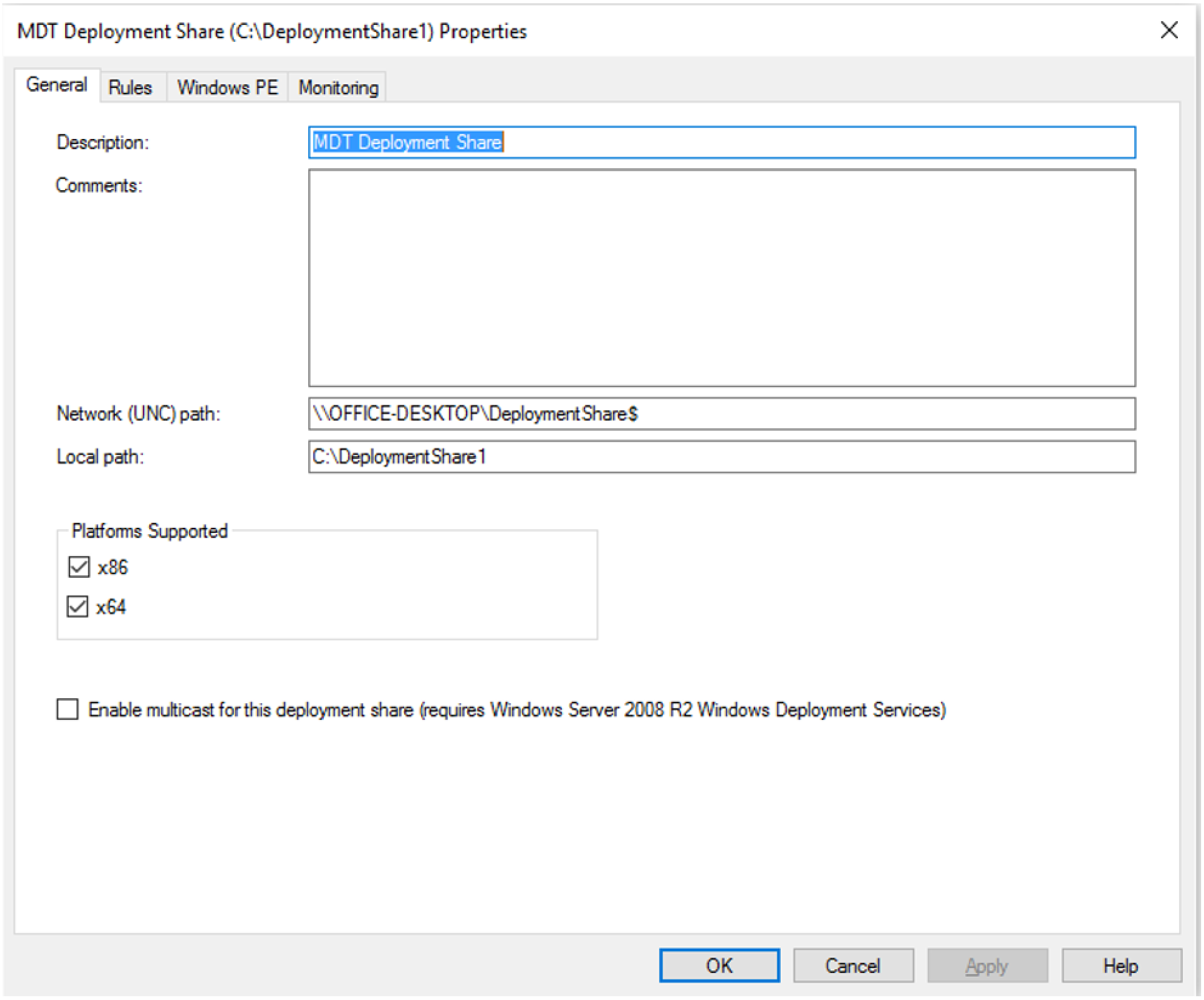
Path:	C:\DeploymentShare1
Upgrade:	False
ShareName:	DeploymentShare$
Description:	MDT Deployment Share
Ask about Backup:	False
Ask for Product Key:	False
Ask for Admin Password:	*********
Ask about Image Capture:	False
Ask about BitLocker:	False

Click next to execute the requested action.

Figure 3 Deployment Share Options

Customizing a Deployment Share

1. Once the Deployment Share gets created, it will appear in the left pane under "Deployment Shares." To customize it, right-click your deployment share, and click properties. The deployment Share Properties Screen will appear.


MDT Deployment Share (C:\DeploymentShare1) Properties

General | Rules | Windows PE | Monitoring

Description: MDT Deployment Share

Comments:

Network (UNC) path: \\OFFICE-DESKTOP\DeploymentShare$

Local path: C:\DeploymentShare1

Platforms Supported

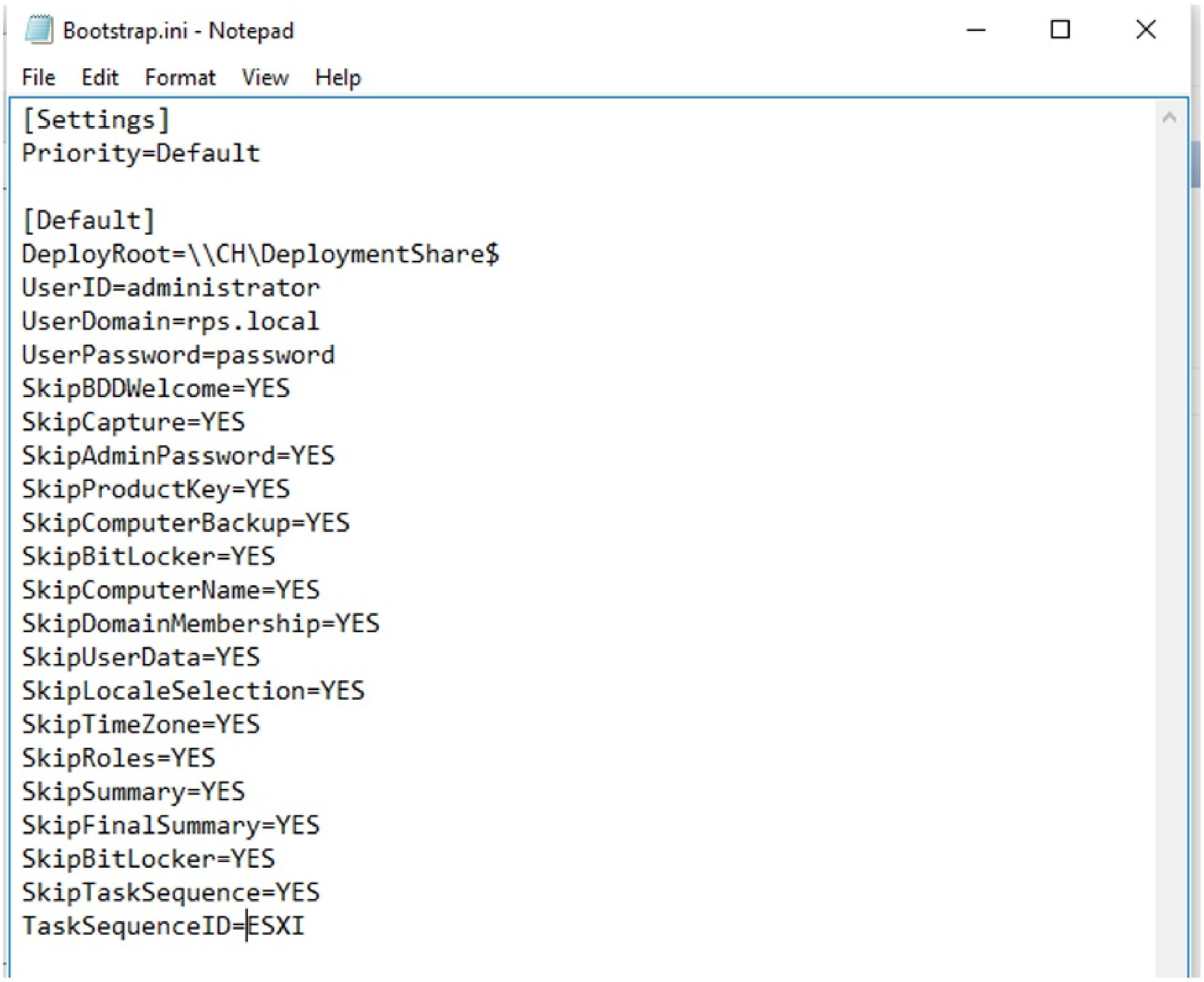
☑ x86

☑ x64

☐ Enable multicast for this deployment share (requires Windows Server 2008 R2 Windows Deployment Services)

Figure 4 Deployment Share Properties

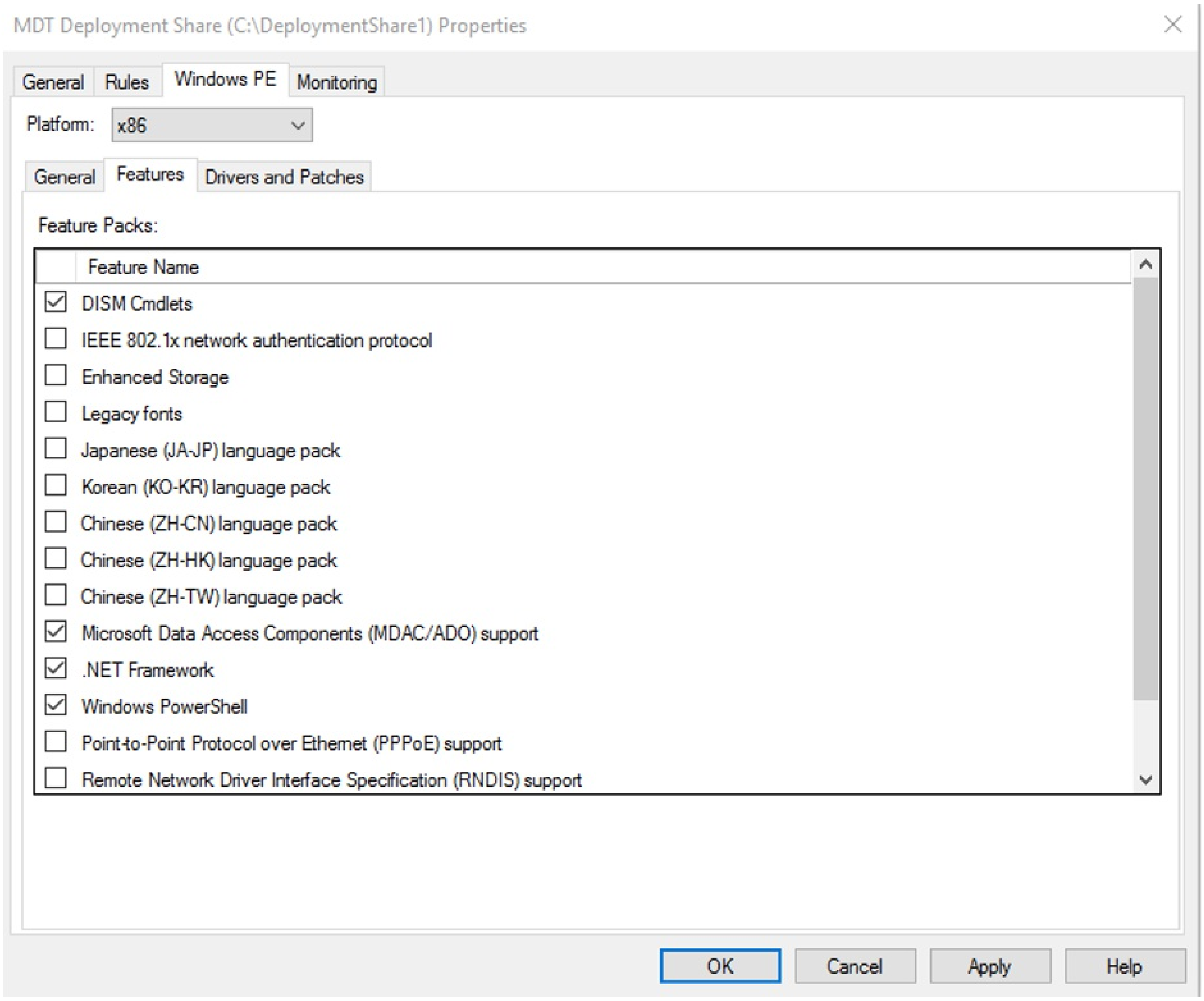
2. On the general tab, select which platforms you will support. Generally, you only need to select “x86” because then you only need to maintain one image, and it will work both on x86 and x64 platforms. On the Rules tab, click the “Edit Bootstrap.ini” button, and edit the “DeployRoot” path to be the path where the deployment share will exist in your deployment environment. In the same file, edit the UserID field to be the user which will access the deployment share from WinPE, the UserDomain to be the User’s domain, the UserPassword to be the User’s password, and the TaskSequenceID to be the ID of the Task Sequence that will be used in order to run deployments. This will be covered in a later step.



```
[Settings]
Priority=Default

[Default]
DeployRoot=\\CH\DeploymentShare$
UserID=administrator
UserDomain=rps.local
UserPassword=password
SkipBDDWelcome=YES
SkipCapture=YES
SkipAdminPassword=YES
SkipProductKey=YES
SkipComputerBackup=YES
SkipBitLocker=YES
SkipComputerName=YES
SkipDomainMembership=YES
SkipUserData=YES
SkipLocaleSelection=YES
SkipTimeZone=YES
SkipRoles=YES
SkipSummary=YES
SkipFinalSummary=YES
SkipBitLocker=YES
SkipTaskSequence=YES
TaskSequenceID=ESXI
```

Figure 5 Bootstrap.ini

3. Save the changes you make to the Bootstrap.ini file, and continue on to the Windows PE tab. Navigate to the features tab, and select the DISM Cmdlets, .NET Framework, and Windows PowerShell check boxes.


MDT Deployment Share (C:\DeploymentShare1) Properties

Platform: x86

Feature Packs:

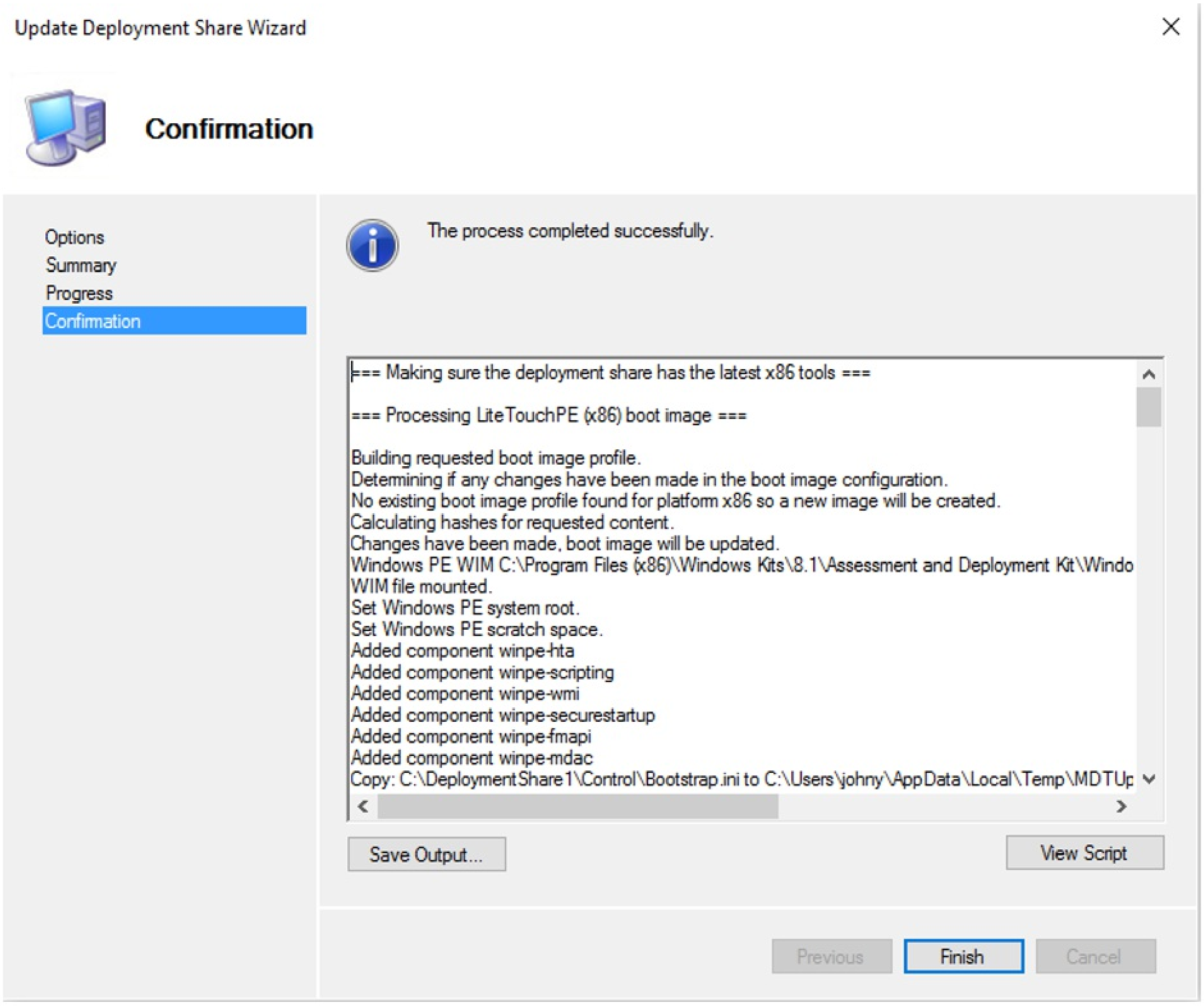
	Feature Name
✓	DISM Cmdlets
☐	IEEE 802.1x network authentication protocol
☐	Enhanced Storage
☐	Legacy fonts
☐	Japanese (JA-JP) language pack
☐	Korean (KO-KR) language pack
☐	Chinese (ZH-CN) language pack
☐	Chinese (ZH-HK) language pack
☐	Chinese (ZH-TW) language pack
✓	Microsoft Data Access Components (MDAC/ADO) support
✓	.NET Framework
✓	Windows PowerShell
☐	Point-to-Point Protocol over Ethernet (PPPoE) support
☐	Remote Network Driver Interface Specification (RNDIS) support

Figure 6 Windows PE Features

Generating Boot Media

Prior to completing these steps for generating boot media, you need to perform steps 1-3 for customizing a deployment share.

4. The last step is to generate the Windows Imaging Format (WIM) Boot media that will be used to run deployments. To do this, right click on your deployment share, select “Update Deployment Share”, select the “Completely regenerate boot media” Radio button, and complete the wizard. The final output will look like the following:



The process completed successfully.

```
=== Making sure the deployment share has the latest x86 tools ===

=== Processing LiteTouchPE (x86) boot image ===

Building requested boot image profile.
Determining if any changes have been made in the boot image configuration.
No existing boot image profile found for platform x86 so a new image will be created.
Calculating hashes for requested content.
Changes have been made, boot image will be updated.
Windows PE WIM C:\Program Files (x86)\Windows Kits\8.1\Assessment and Deployment Kit\Windo
WIM file mounted.
Set Windows PE system root.
Set Windows PE scratch space.
Added component winpe-hta
Added component winpe-scripting
Added component winpe-wmi
Added component winpe-securestartup
Added component winpe-fmapi
Added component winpe-mdac
Copy: C:\DeploymentShare1\Control\Bootstrap.ini to C:\Users\johny\AppData\Local\Temp\MDTUp
```

Figure 7 Generating Boot Media

5. The WIM that is generated will be located in your deployment share folder, under the “Boot” Folder. The last step is that this WIM that is generated needs to be imported into Windows Deployment Services (WDS) on the server that is going to be responsible for running deployments. To do this, open the WDS console, expand Servers, expand your WDS server, right click on Boot Images, select “Import Boot Image”, and then navigate to your deployment share and select the WIM that you created.

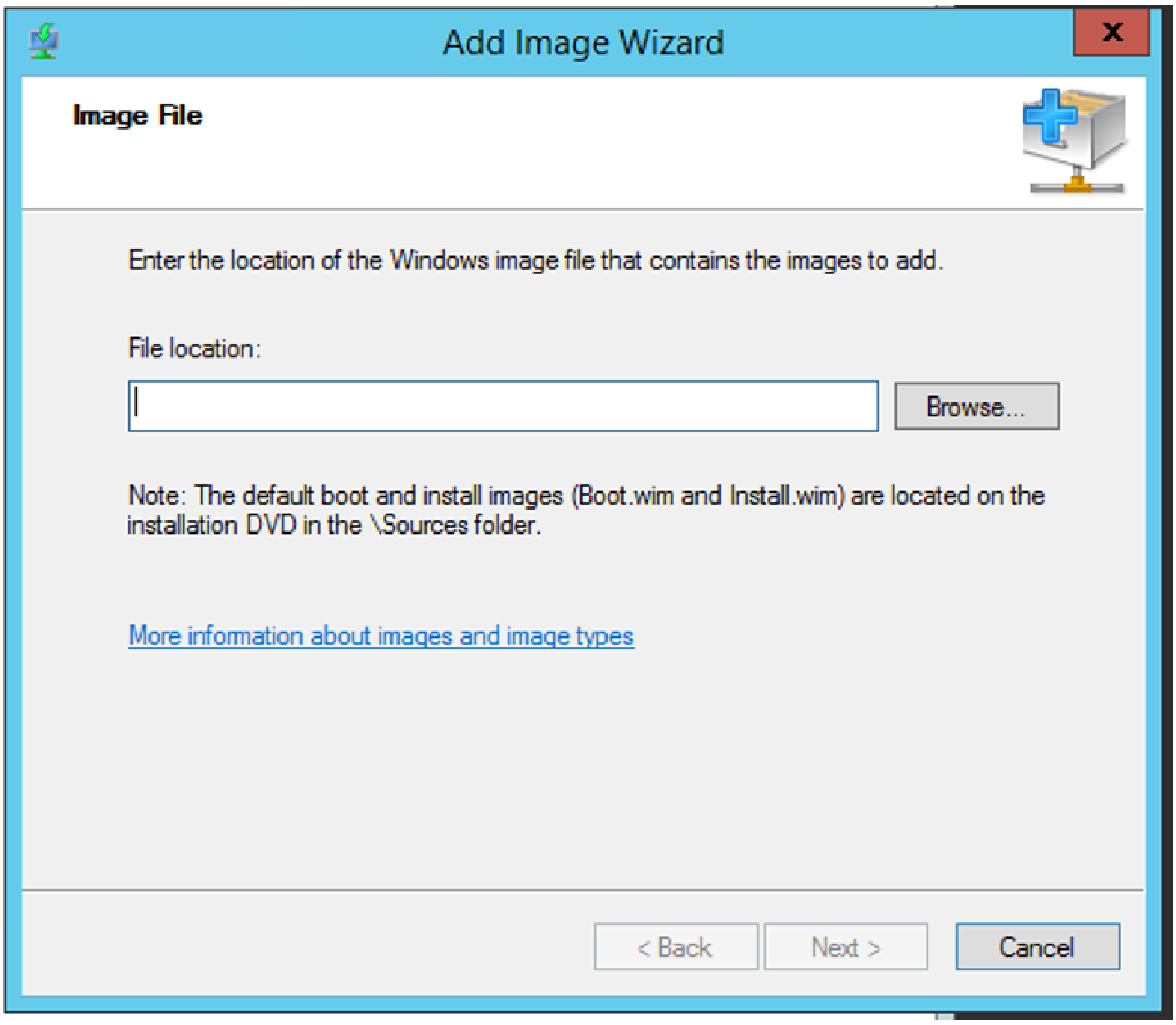

Add Image Wizard

Image File

Enter the location of the Windows image file that contains the images to add.

File location:

Note: The default boot and install images (Boot.wim and Install.wim) are located on the installation DVD in the \Sources folder.

More information about images and image types

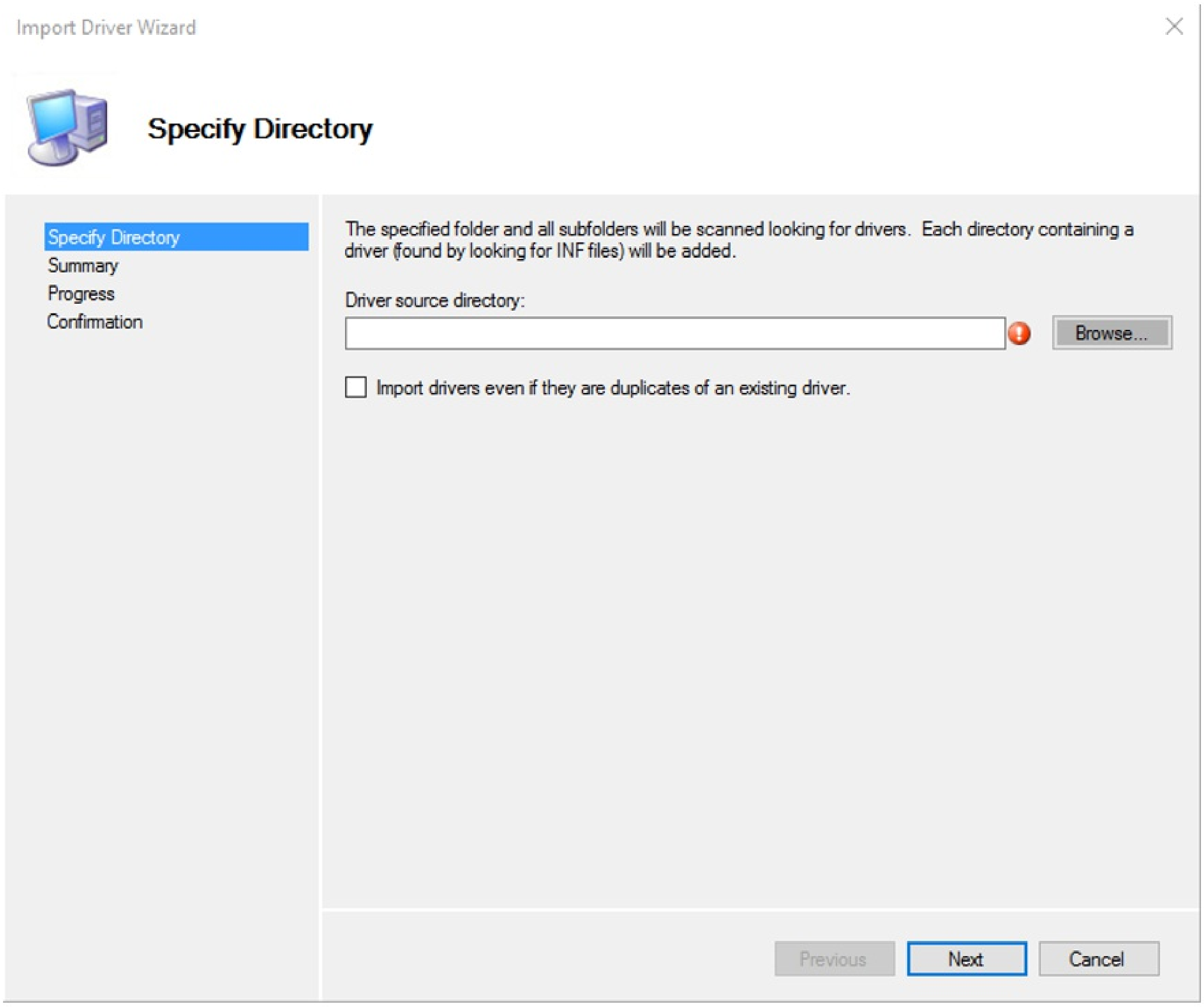
Figure 8 WIM Import

Adding Drivers to WIM

There are cases where the default network and storage drivers built into the WIM are not sufficient for running deployments, and the client machine is either unable to communicate to the deployment server over the LAN, or cannot access its storage. In these cases, additional drivers need to be injected into the WIM so that connectivity can be restored.

Adding Drivers to MDT

In order to inject drivers into your WIM, you need to have the drivers downloaded and available on the machine where MDT is installed. To inject the drivers, open the deployment workbench, expand your deployment share, right-click on “Out-of-Box Drivers”, and then select “Import Drivers”. The following wizard will appear:


Import Driver Wizard

Specify Directory

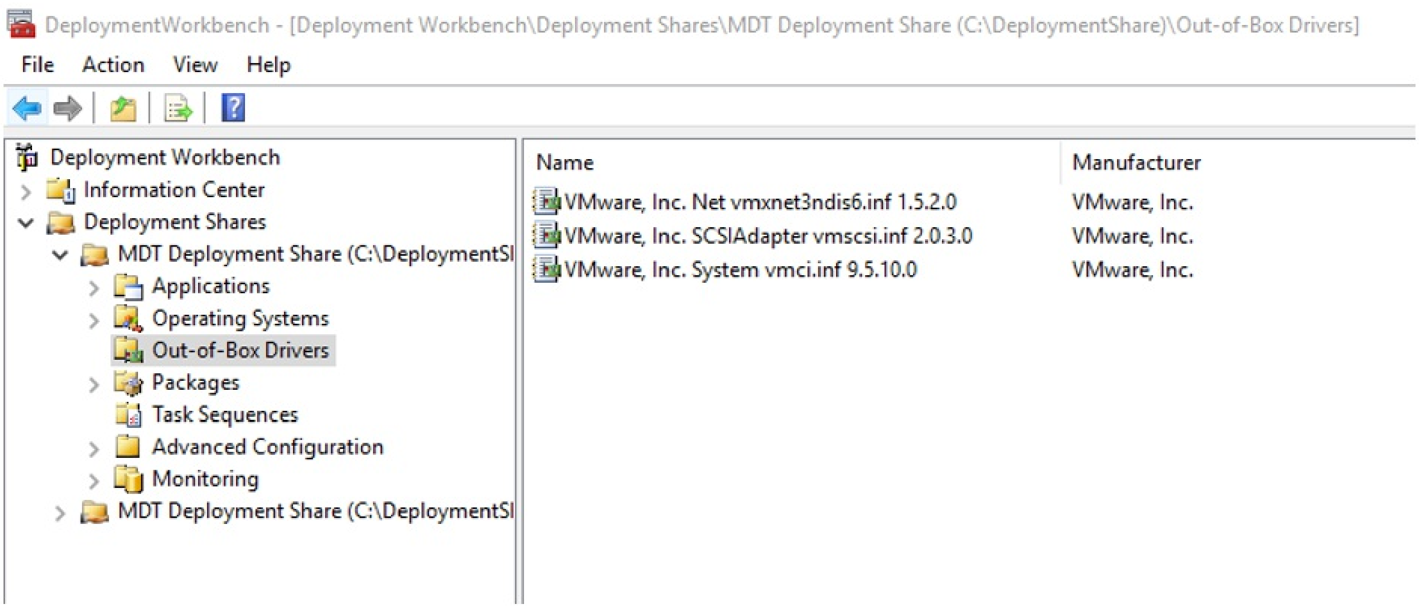
The specified folder and all subfolders will be scanned looking for drivers. Each directory containing a driver (found by looking for INF files) will be added.

Driver source directory:

☐ Import drivers even if they are duplicates of an existing driver.

Figure 9 Import Drivers

Select the browse button, navigate to the folder where your third party drivers are located, select the folder and then complete the wizard. If the drivers are imported successfully, they will appear in the middle pane of the deployment workbench as follows:


DeploymentWorkbench - [Deployment Workbench\Deployment Shares\MDT Deployment Share (C:\DeploymentShare)\Out-of-Box Drivers]

File Action View Help

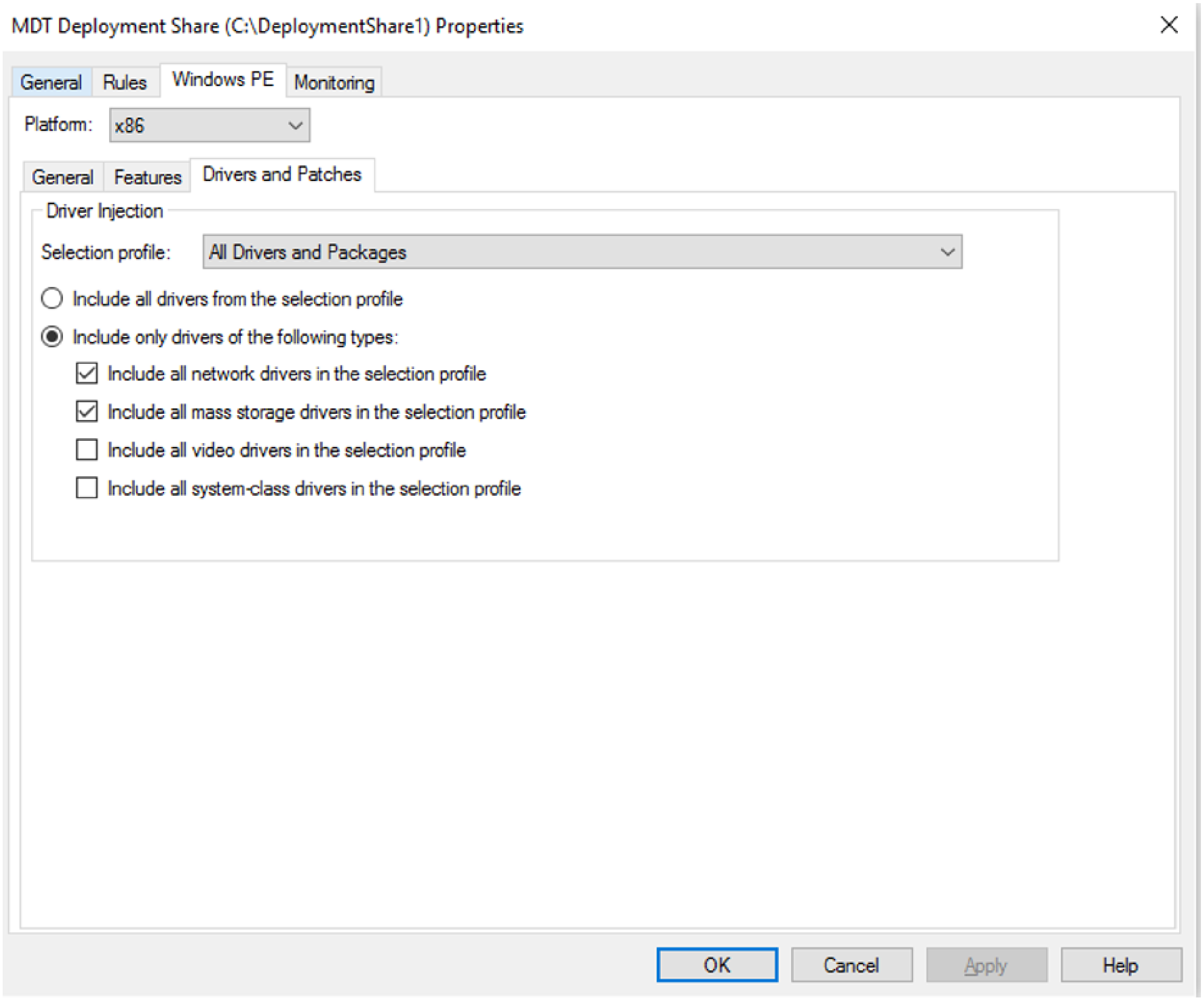
- Deployment Workbench
  - Information Center
  - Deployment Shares
    - MDT Deployment Share (C:\DeploymentSI
      - Applications
      - Operating Systems
      - Out-of-Box Drivers
      - Packages
      - Task Sequences
      - Advanced Configuration
      - Monitoring
    - MDT Deployment Share (C:\DeploymentSI

Name	Manufacturer
VMware, Inc. Net vmxnet3ndis6.inf 1.5.2.0	VMware, Inc.
VMware, Inc. SCSIAdapter vmscsi.inf 2.0.3.0	VMware, Inc.
VMware, Inc. System vmci.inf 9.5.10.0	VMware, Inc.

Figure 10 Third Party Driver

Configuring MDT to Use Third Party Drivers

This process just makes the drivers available in your deployment workbench, but has not yet injected the drivers into the WIM. To do so, right-click on your deployment share, select Properties, and navigate to the Windows PE tab, and then the Drivers and Patches tab. Select which type of drivers you want added to the WIM (you can either select all drivers that are available in your deployment workbench, or specify a specific type over driver, and then click OK.


MDT Deployment Share (C:\DeploymentShare1) Properties

General | Rules | Windows PE | Monitoring

Platform: x86

General | Features | Drivers and Patches

Driver Injection

Selection profile: All Drivers and Packages

- ( ) Include all drivers from the selection profile
- (●) Include only drivers of the following types:
  - ☑ Include all network drivers in the selection profile
  - ☑ Include all mass storage drivers in the selection profile
  - ☐ Include all video drivers in the selection profile
  - ☐ Include all system-class drivers in the selection profile

Figure 11 Driver Settings

Once you determine your settings, you need to regenerate the WIM so that the newly added drivers are injected into it. To do so, follow the process documented above in the WIM creation process.

Adding Additional Components to WIM

There are cases where additional components need to be added to your WIM, that cannot be done using the deployment workbench. One such case is adding the environment variable connection string to the registry of the WIM, so that clients that are in the Windows Pre-Boot Environment (WinPE), can import the RPS DAC dll and interface with the CMDB.

Mounting the WIM

To do this, create an empty folder on the server with MDT installed, in a location that is easily accessible. A good example of this would be something like “C:\mount”. Next, open an administrator command prompt window, and enter the following command:

```
Cd C:\Program Files (x86)\Windows Kits\8.1\Assessment and Deployment Kit\Deployment Tools\x86\DISM
```

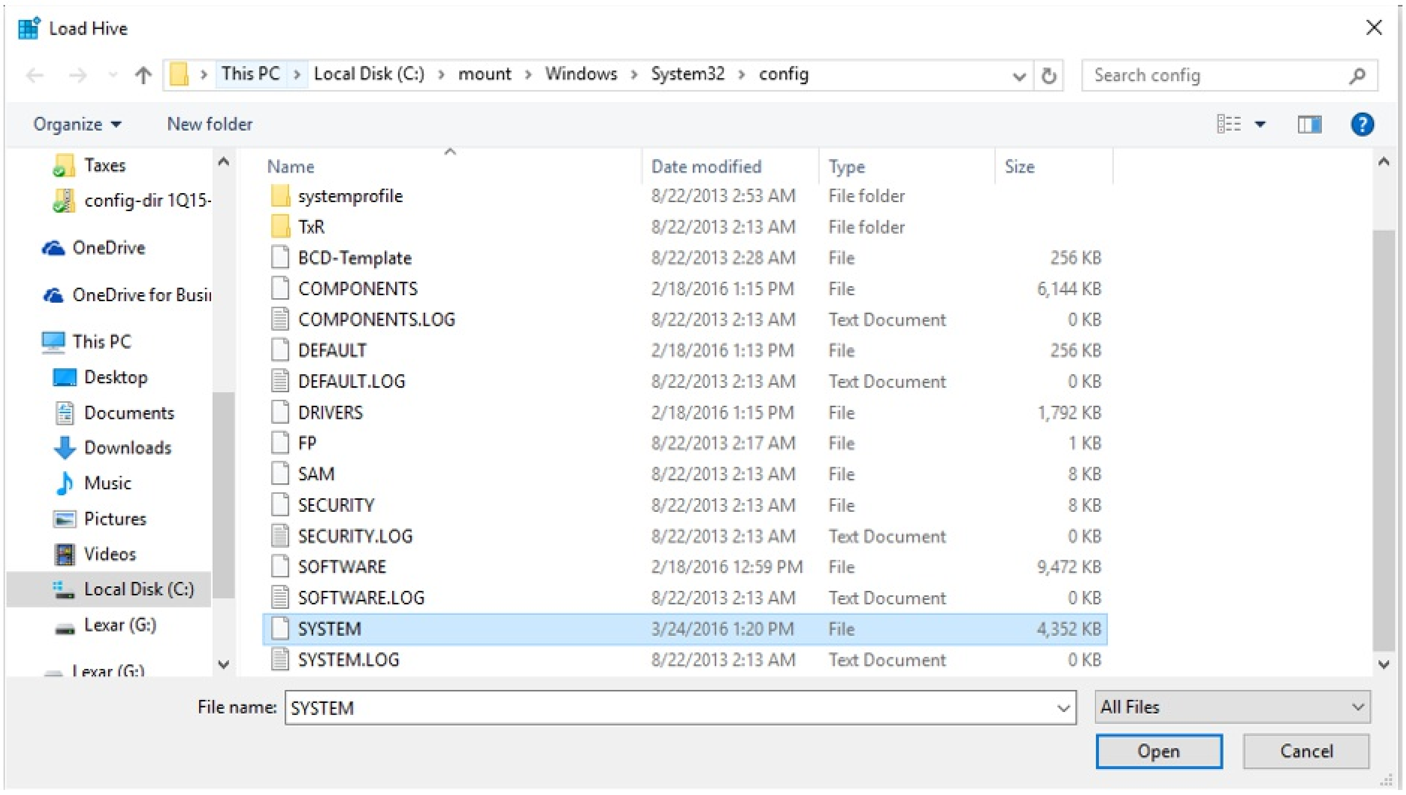
Once in the DISM directory, issue the following command:

```
dism /mount-image /imagefile:C:\DeploymentShare\Boot\LiteTouchPE_x86.wim /index:1 /mountdir:C:\mount
```

Make sure to replace the imagefile path to the location of your WIM, and the mountdir path to where you created that folder.

Loading the WIM Registry

Navigating to the empty folder you created will now show you the contents of your WIM. The connection string that is used by the RPS API is a PowerShell environment variable, which cannot be set inside of WinPE, unless already exists in the registry of the WIM file. In order to do this, open the registry editor on the machine where you mounted the WIM, select HKEY_LOCAL_MACHINE, then click File>Load Hive. Navigate to your mount directory, then to Windows, System32, config, and select the file called "System".



Name	Date modified	Type	Size
systemprofile	8/22/2013 2:53 AM	File folder	
TxR	8/22/2013 2:13 AM	File folder	
BCD-Template	8/22/2013 2:28 AM	File	256 KB
COMPONENTS	2/18/2016 1:15 PM	File	6,144 KB
COMPONENTS.LOG	8/22/2013 2:13 AM	Text Document	0 KB
DEFAULT	2/18/2016 1:13 PM	File	256 KB
DEFAULT.LOG	8/22/2013 2:13 AM	Text Document	0 KB
DRIVERS	2/18/2016 1:15 PM	File	1,792 KB
FP	8/22/2013 2:17 AM	File	1 KB
SAM	8/22/2013 2:13 AM	File	8 KB
SECURITY	8/22/2013 2:13 AM	File	8 KB
SECURITY.LOG	8/22/2013 2:13 AM	Text Document	0 KB
SOFTWARE	2/18/2016 12:59 PM	File	9,472 KB
SOFTWARE.LOG	8/22/2013 2:13 AM	Text Document	0 KB
SYSTEM	3/24/2016 1:20 PM	File	4,352 KB
SYSTEM.LOG	8/22/2013 2:13 AM	Text Document	0 KB

File name: SYSTEM
All Files

Figure 12 WIM Registry

Adding new Registry Keys to WIM Registry

After selecting Open, the registry editor will prompt for a key name, enter "WIM", and then select Ok. Expand WIM>ControlSet001>Control>Session Manager>Environment. Right Click on the Environment folder and select New>String Value. For the name of the registry key enter "RPSDbAddress" and for the Value, enter the following connection string:

```
Data Source=DB\SMA;Initial Catalog=Rps.Database;
MultipleActiveResultSets=True;user=user;password=password
```

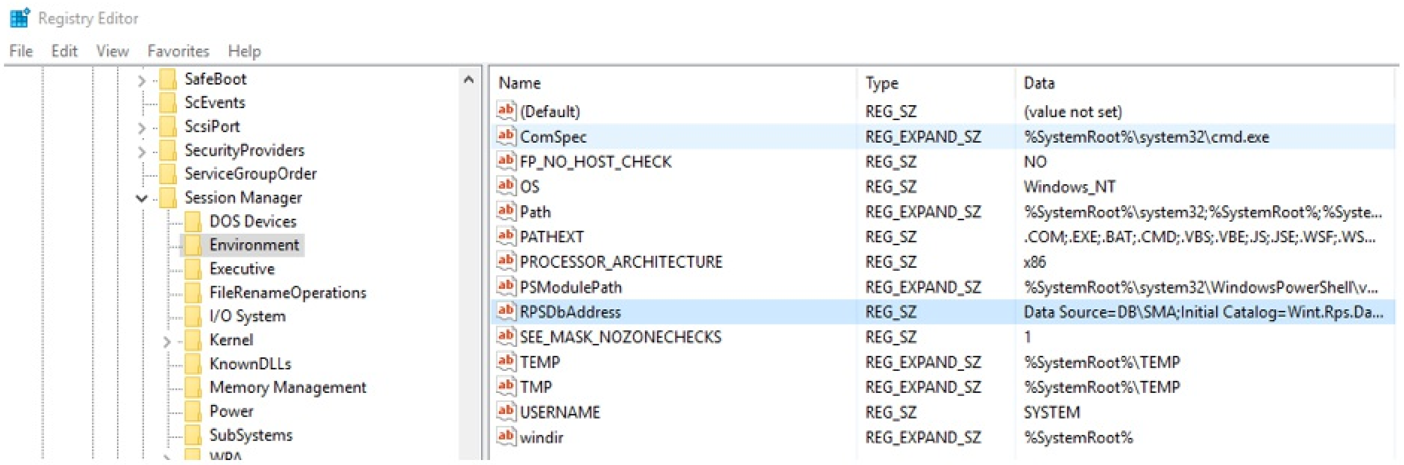
Make sure the Data Source is the name of your SQL Server\InstanceName, the catalog is the name of the RPS CMDB Database, and the user and password are the credentials of a local SQL Account with access to read and write to the database.

⚠ IMPORTANT

The user provided in the connection string must be a local SQL account, as domain accounts are not supported when using SQL connection strings and supplying the username and password as part of the string.

Once you have entered the connection string, click OK to save the registry key. The key should now appear under the Environment folder.

```
Data Source = DB\Sma; Initial Catalog = Rps.Database;
MultipleActiveResultSets = True; user = user; password = password
```



Name	Type	Data
(Default)	REG_SZ	(value not set)
ComSpec	REG_EXPAND_SZ	%SystemRoot%\system32\cmd.exe
FP_NO_HOST_CHECK	REG_SZ	NO
OS	REG_SZ	Windows_NT
Path	REG_EXPAND_SZ	%SystemRoot%\system32;%SystemRoot%;%Syste...
PATHEXT	REG_SZ	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WS...
PROCESSOR_ARCHITECTURE	REG_SZ	x86
PSModulePath	REG_EXPAND_SZ	%SystemRoot%\system32\WindowsPowerShell\v...
RPSDbAddress	REG_SZ	Data Source=DB\SMA;Initial Catalog=Wint.Rps.Da...
SEE_MASK_NOZONECHECKS	REG_SZ	1
TEMP	REG_EXPAND_SZ	%SystemRoot%\TEMP
TMP	REG_EXPAND_SZ	%SystemRoot%\TEMP
USERNAME	REG_SZ	SYSTEM
windir	REG_EXPAND_SZ	%SystemRoot%

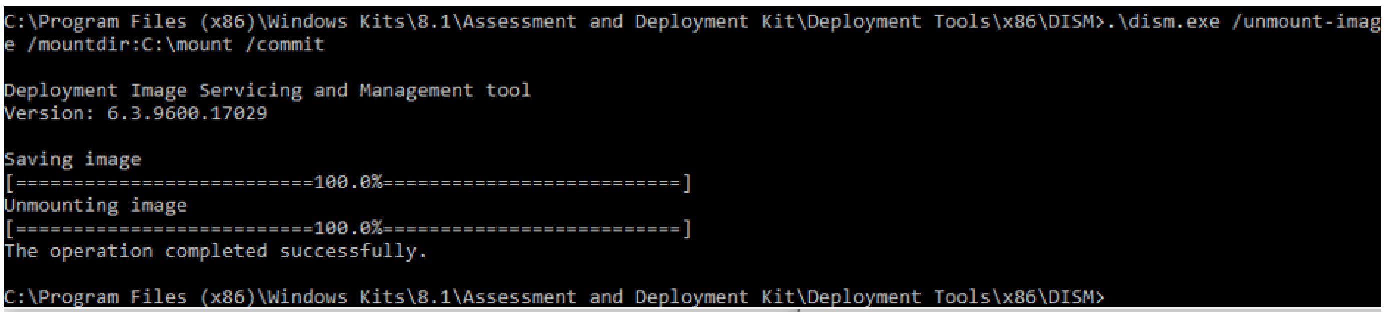
Figure 13 RPS Registry Key

Unloading Registry and Unmounting WIM

Navigate back up to the WIM folder that was created when the registry was mounted, and select File>Unload Hive. Select "Yes" when you are prompted to unload the hive, and then close the registry editor. Navigate back to the elevated command prompt window that you used to mount the WIM, and issue the following command, making sure to indicate the correct mount directory.

```
.\dism.exe /unmount-image /mountdir:C:\mount /commit
```

A successful commit will result in the following:



```
C:\Program Files (x86)\Windows Kits\8.1\Assessment and Deployment Kit\Deployment Tools\x86\DISM>.\dism.exe /unmount-imag
e /mountdir:C:\mount /commit

Deployment Image Servicing and Management tool
Version: 6.3.9600.17029

Saving image
[==========================100.0%==========================]
Unmounting image
[==========================100.0%==========================]
The operation completed successfully.

C:\Program Files (x86)\Windows Kits\8.1\Assessment and Deployment Kit\Deployment Tools\x86\DISM>
```

Figure 14 Commit WIM Changes

The WIM now has the proper connection string stored as an environment variable and will be able to successfully connect to the CMDB to interact with the API. If you need to add any additional third party pieces of software, you can simply copy the executables into the Windows/System32 Folder, so that they can be called from scripts within WinPE.

Task Sequences

Task sequences are a set of instructions that the devices that boot into WinPE will execute. Using the Deployment Workbench, a user can easily add and remove steps to be performed while the client is in WinPE, as well as make task sequence variables available to the client to be consumed during execution.

Creating a Task Sequence

In order to create a Task Sequence, open the Deployment Workbench, expand your Deployment Share, right-click on Task Sequences, and select “New Task Sequence.” A wizard will open that will prompt for a Task Sequence ID as well as a Task Sequence Name.

For the ID, enter a short acronym that can easily be reference in the bootstrap.ini file discussed earlier. For example, if this task sequence will be used to deploy Windows 10, a good ID would be “Win10”. The name is for your own knowledge, so you can be as descriptive as you would like in this field.

Complete the wizard to create your task sequence, accepting the defaults.

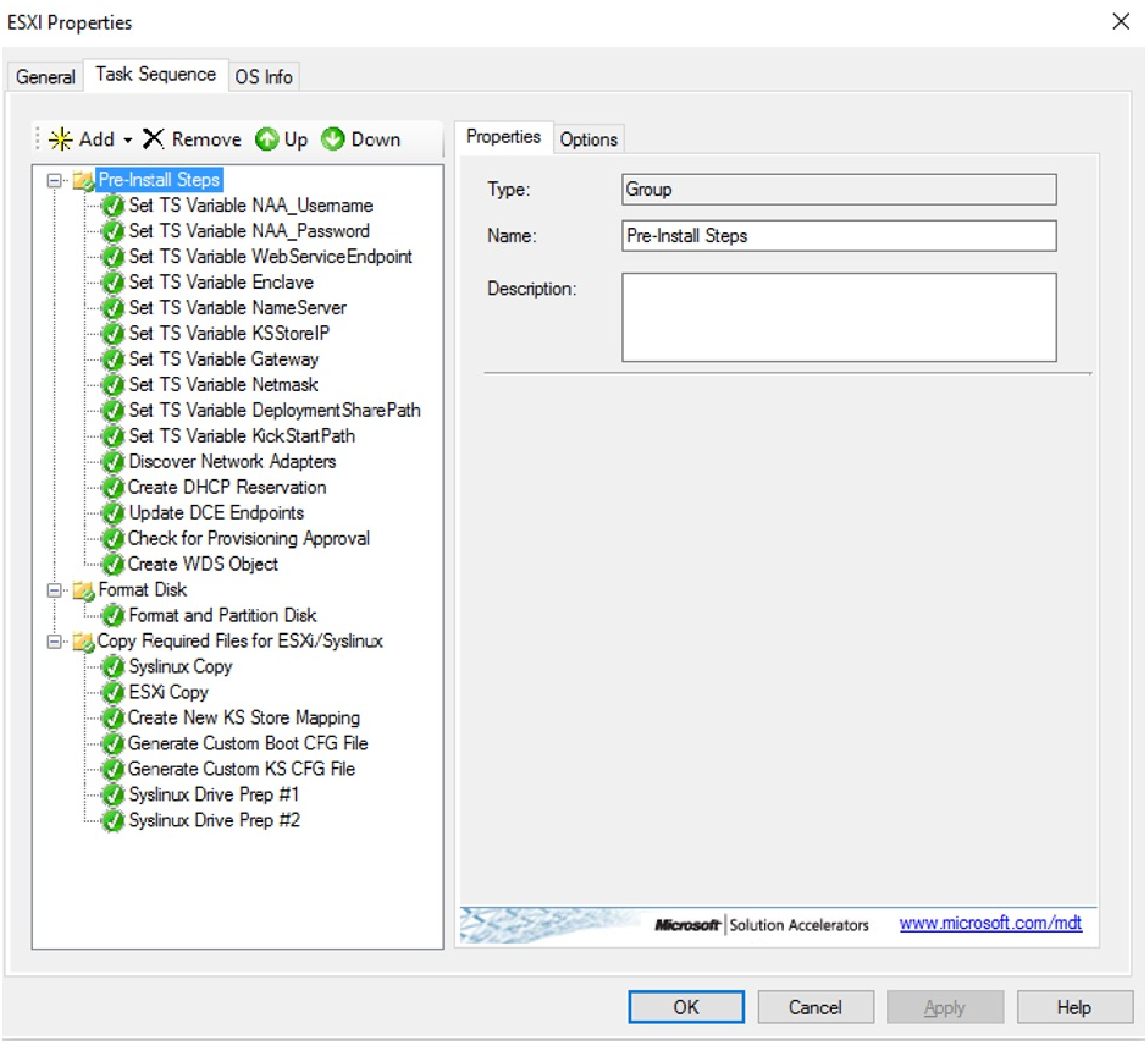
Editing a Task Sequence

Once the task sequence has been created, you can begin editing it to suite your environment and the steps that you want to clients to execute in order to deploy an operating system to your clients.

To being editing your Task Sequence, right-click on the Task Sequence in the Deployment Workbench and select "Properties", the navigate to the "Task Sequence" tab. The following window will appear.

NOTE

This is an example of a task sequence where task sequence variables and steps have already been supplied.



- Pre-Install Steps
  - Set TS Variable NAA_Username
  - Set TS Variable NAA_Password
  - Set TS Variable WebServiceEndpoint
  - Set TS Variable Enclave
  - Set TS Variable NameServer
  - Set TS Variable KSStoreIP
  - Set TS Variable Gateway
  - Set TS Variable Netmask
  - Set TS Variable DeploymentSharePath
  - Set TS Variable KickStartPath
  - Discover Network Adapters
  - Create DHCP Reservation
  - Update DCE Endpoints
  - Check for Provisioning Approval
  - Create WDS Object
- Format Disk
  - Format and Partition Disk
- Copy Required Files for ESXi/Syslinux
  - Syslinux Copy
  - ESXi Copy
  - Create New KS Store Mapping
  - Generate Custom Boot CFG File
  - Generate Custom KS CFG File
  - Syslinux Drive Prep #1
  - Syslinux Drive Prep #2

Type: Group

Name: Pre-Install Steps

Description:

Figure 15 Task Sequence

In order to add steps to your Task Sequence, click on the "Add" button at the top, select "General" and then select "Set Task Sequence Variables". Task Sequence Variables are pieces of information that clients will need to perform various functions. For example, if the client is going to need to interface with SMA, it will need to know the Web Service Endpoint of the SMA server. Instead of hardcoding that address into the script that will interface with SMA, it can be supplied as a variable that can be consumed by the client, should it ever need to change.

Generally it is recommended that you supply all of your necessary Task Sequence Variables as the first steps in the Task Sequence, so that they are available to the client before any actual deployment logic is performed. After you have inputted all of your necessary Task Sequence Variables, you can proceed to adding steps that require PowerShell Scripts to be executed, as well as any other options that are required. All the available options are available under the "Add" button.

NOTE

If you are adding PowerShell scripts as steps in your task sequence, for the value of the "PowerShell script" field, the name of the script should be entered in the following manner:

`%SCRIPTROOT%\FilenameOfScript.ps1`

All of the scripts required by your task sequence should be placed in the "Scripts" folder of your Deployment Share.

Credential Masking

Clients that boot into WinPE are not domain joined, and usually need domain account credentials in order to perform various functions throughout the Task Sequence. One such example is if the client needs to interface with SMA in order to start a Runbook. Storing the password as clear text is not recommended, and one method of getting around this is to use PowerShell Obfuscation.

Encoding

Credentials within PowerShell as stored and consumed as Secure Strings. In order to do this, open a PowerShell session and issue the following command, replacing "Password" with the password you want to secure:

```
$pass = "Password" | ConvertTo-SecureString -AsPlainText –Force
```

Converting to Masked Credential

If you return the $pass variable created above, you will note that it does not return the value of your "Password", but instead returns a PowerShell Object of Secure String. What we want to return is the hash that is generated from securing the password. In order to do this, in the PowerShell session that you have open, issue the following command:

```
$hash = ConvertFrom-SecureString $pass -Key (1..16)
```

Now if you return $hash, a very long string of characters will be returned, which is the hashed version of your secure password. You can then take this hashed value and store it as a Task Sequence Variable in your task sequence, to be consumed and utilized by clients booting into WinPE.

Converting Masked Credential Back to a Secure String

In order to convert the hashed value back into a Secure String, the following command is used:

```
$pass = $hash | ConvertTo-SecureString -Key (1..16)
```

Workflow Examples

The following section will provide some very basic examples of how to perform certain operations in PowerShell scripts that run within WinPE, how to interact with the RPS API, and how to interface with the Controller.

Using Masked Credential

The previous section described how to create a password hash to be consumed as a Task Sequence Variable. The following syntax is used to read the task sequence variable and use it to create a credential object for PowerShell to utilize within WinPE. Note the name of the Task Sequence Variable in question is "Password" and the value is the hash that was created by following the steps from the previous section.

```
$MS_ConfigMgr_Env = New-Object -ComObject Microsoft.SMS.TSEnvironment
$Password = $MS_ConfigMgr_Env.Value ("Password")
$Username = $Ms_ConfigMgr_Env.Value ("Username")
$cred = $Password | ConvertTo-SecureString -key(1..16)
credentials = New-Object System.Management.Automation.PSCredential ($UserName, $cred)
```

Importing RPS API

The following syntax is used to import the RPS API so that clients within WinPE can interact with the DAC and CMDB.

NOTE

The RPS API should be placed in the Deployment Share, under the Applications Folder, and then in its own folder called DAC.

```
$MS_ConfigMgr_Env = New-Object -ComObject Microsoft.SMS.TSEnvironment
$deploymentSharepath = $MS_ConfigMgr_Env.Value("DeploymentSharePath")
Import-Module "$deploymentSharePath\Applications\DAC\Rps.Api.dll"
```

Importing SMA Module

The following syntax is used to import the SMA Module so that clients within WinPE can interact Start Runbooks.

NOTE

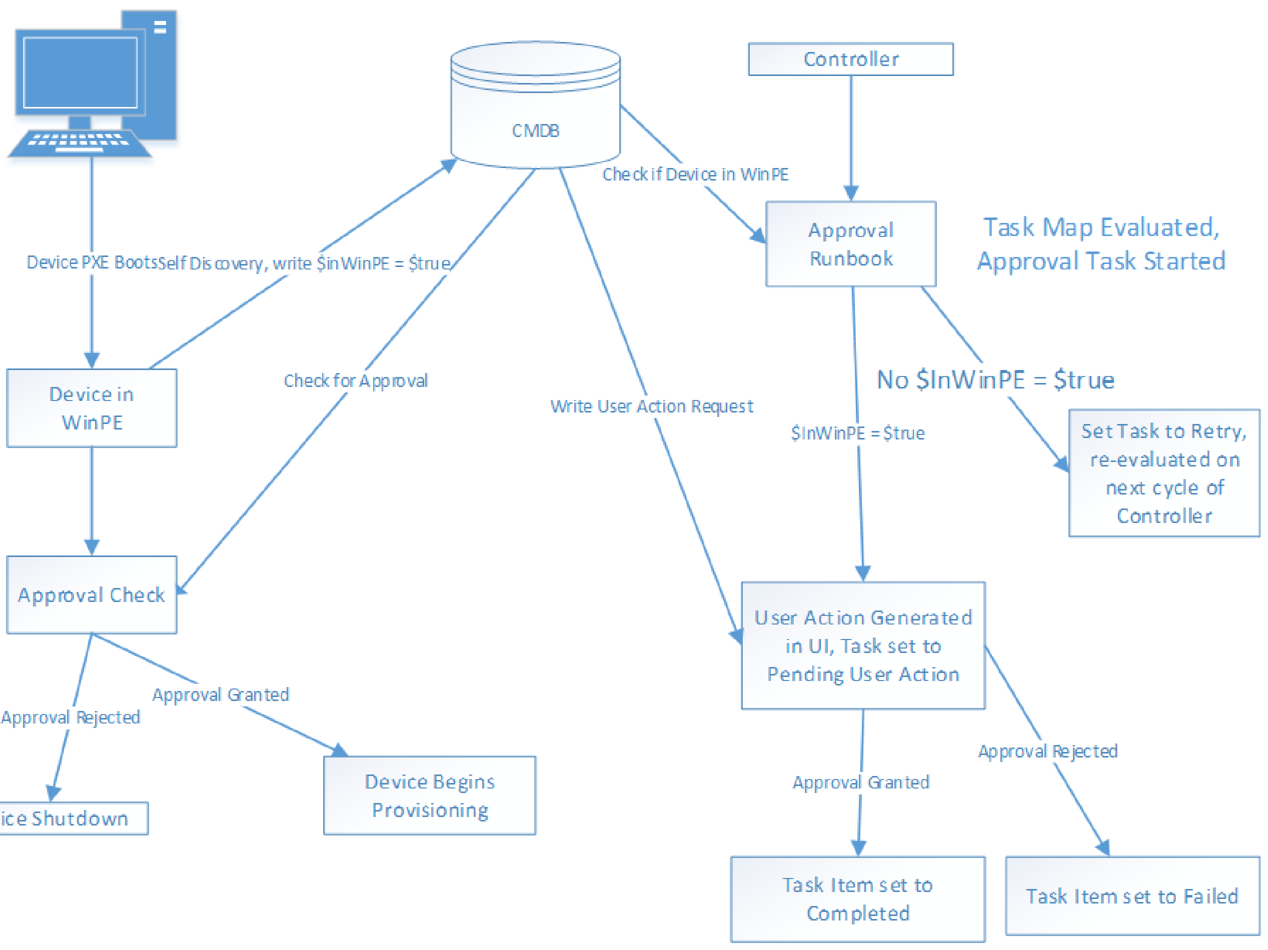
The SMA module should be placed in the Deployment Share, under the Applications Folder, and then in its own folder called SMA.

The following code will also start a runbook.

```
$MS_ConfigMgr_Env = New-Object -ComObject Microsoft.SMS.TSEnvironment
$deploymentSharePath = $MS_ConfigMgr_Env.Value("DeploymentSharePath")
$WebServiceEndpoint = $MS_ConfigMgr_Env.Value("WebServiceEndpoint")
$UserName = $MS_COnfigMgr_Env.Value("UserName")
$Password = $MS_COnfigMgr_Env.Value("Password")

$cred = $Password | ConvertTo-SecureString -key(1..16)
$credentials = New-Object System.Management.Automation.PSCredential($UserName, $cred)

$runbookName = "Sample-Runbook""

$params = @{"parameter1" = $parameter1; "parameter2" = $parameter2}

Import-Module
"$deploymentSharePath\Applications\SMA\Microsoft.SystemCenter.ServiceManagementAutomation\Microsoft.SystemCent
er.ServiceManagementAutomation.psd1"

Start-SmaRunbook -Credential $credentials -WebServiceEndpoint $WebServiceEndpoint -Name $runbookName -
Parameters $params
```

Bare Metal Provisioning Approval

On headless systems, administrators using RPS need a way to approve provisioning of client devices before their hard drives are wiped and a new Operating System installed on the hard drive. The following set of steps give an example of how users utilizing the RPS toolset would be able to perform an example approval process. The diagram below is an example overview of this process.


CMDB
Controller
Check if Device in WinPE
Approval Runbook
Task Map Evaluated, Approval Task Started
Device PXE BootsSelf Discovery, write $inWinPE = $true
Device in WinPE
Check for Approval
Write User Action Request
No $InWinPE = $true
$InWinPE = $true
Set Task to Retry, re-evaluated on next cycle of Controller
Approval Check
User Action Generated in UI, Task set to Pending User Action
Approval Granted
Approval Rejected
Device Shutdown
Device Begins Provisioning
Approval Rejected
Approval Granted
Task Item set to Completed
Task Item set to Failed

Client Generates Flag that it is in WinPE

First, the client needs to set a flag for its Target Item that it has booted into WinPE, so that the controller is aware and can generate the approval. For the first PowerShell script that runs in the Task Sequence, some kind of code similar to the following should be used. In short, the client is querying the CMDB for its Target Item, and setting a property called “InWinPE” to $true.

```
# find the Client

$Client = Get-RpsTargetItem -Name $ComputerName

if($Client)
{
    $Client.IsActive = $true
    $Client.InWinPE = $true
    $Client.Update()
}
```

Approval Runbook

The following is an example Runbook that is run by the Controller to check if a client has booted into WinPE, and if so, generate an approval task for an administrator either Approve or Reject.

NOTE

This should be the first step in the TaskMap associated with the client device, so that no automations proceed on the client until this approval has been granted.

```
    workflow New-BareMetalApprovalTask
{
    param
    {
        [parameter(Mandatory = $true)][string]$taskAssignmentId]
    }

# Disables serialization of objects in certain circumstances allowing for method calls.
    $PSDisableSerializationPreference = $true

    $taskassignment = Get-RpsTaskAssignment -Id $taskAssignmentId
    $targetitem = Get-RpsTargetItem -Id $taskassignment.TargetItemId

    if($targetitem.InWinPE -eq $true)
    {
        New-RpsTaskAssignmentUserAction -TaskAssignmentStatusId $taskAssignmentId
        $null = ($targetitem.ApprovalAssignmentId = "$taskAssignmentId")
        $null = $targetitem.Update()
        $null = ($taskassignment.StatusMessage = "Approval Generated")
        $null = ($taskassignment.TaskState = "PendingUserAction")
        $null = taskassignment.Update()
    }
    else
    {
        $null = ($taskassignment.TaskState = "Retry")
        $null = ($taskassignment.StatusMessage = "Target Item is not in WinPE")
        $null = $taskassignment.Update()
    }
}
```

Client Checks for Approval

Prior to the step in the Task Sequence where the client wipes its hard drive and lays down the new Operating System, it needs to verify that approval has either been received or denied. To do this, a PowerShell script similar to the following example should be created and placed on the Deployment Share, and then called from the task sequence.

```
# Obtain TS vars
$MS_ConfigMgr_Env = New-Object -ComObject Microsoft.SMS.TSEnvironment
$deploymentSharePath = $MS_ConfigMgr_Env.Value("DeploymenySharePath")

# Set Computer Name
$ComputerName = "$ClientComputerName"

# Import DAC
Import-Module "$deploymentSharePath\Application\DAC\Rps.Api.dll"

# find the client in CMDB
$Client = Get-RpsTargetItem -Name $ComputerName

# find the Task Assingment
$TaskAssignment = $Client.Properties | where Name -eq "ApprovalAssignmentId"
while($TaskAssignment)
{
    Start-Sleep -Seconds 5
    $Client = Get-RpsTargetItem -Name $ComputerName
    $TaskAssignment = $DCE.Properties | where Name -eq "ApprovalAssignmentId"
}
$TaskAssignment = $TaskAssignment.Value

# Monitor for Approval/Rejection
if ($TaskAssignment)
{
    do
    {
        Start-Sleep -Seconds 5
        $Status = (Get-RpsTaskAssignment -Id $TaskAssignment).TaskState
    }
    while (($Status -ne "Completed") -and ($Status -ne "Cancelled))

    if($Status -eq "Completed")
    {
        $DCE.InWinPE = $null
        $DCE.Update()
    }

    if($Status -eq "Cancelled")
    {
        $DCE.InWinPE = $null
        $DCE.Update()
        wpeutil shutdown
    }
}
```

Glossary

TERM	DEFINITION
MDT	Microsoft Deployment Toolkit.
ADK	Assessment and Deployment Kit.
WIM	Windows Imaging Format – A Bootable file allowing a client to enter WinPE.
WDS	Windows Deployment Services – A windows service used to perform provisioning of clients.
SMA	Microsoft Systems Management Automation – a platform for automating tasks via PowerShell workflow.

TERM	DEFINITION
Web Service	A web-based receiver that enables connectivity from other applications.
Runbook	A "task" executed within SMA, may contain multiple workflows.
Runbook Worker	The SMA service that processes and executes Runbooks.
WinPE	Windows Pre-Boot Environment – An Environment that runs in memory on the client that is used to provision the client.
SCCM	Microsoft System Center Configuration Manager.
Controller	The components used to receive/manage data from the worker or Runbook tasks. In this case SMA has Runbooks that act as the controller(s).
CMDB	Configuration Management Database.
TaskMap	A set of tasks that are run by the Controller to execute automations on a target.
RPS	Rapid Provisioning System – A Toolset used to perform automations.

More Resources

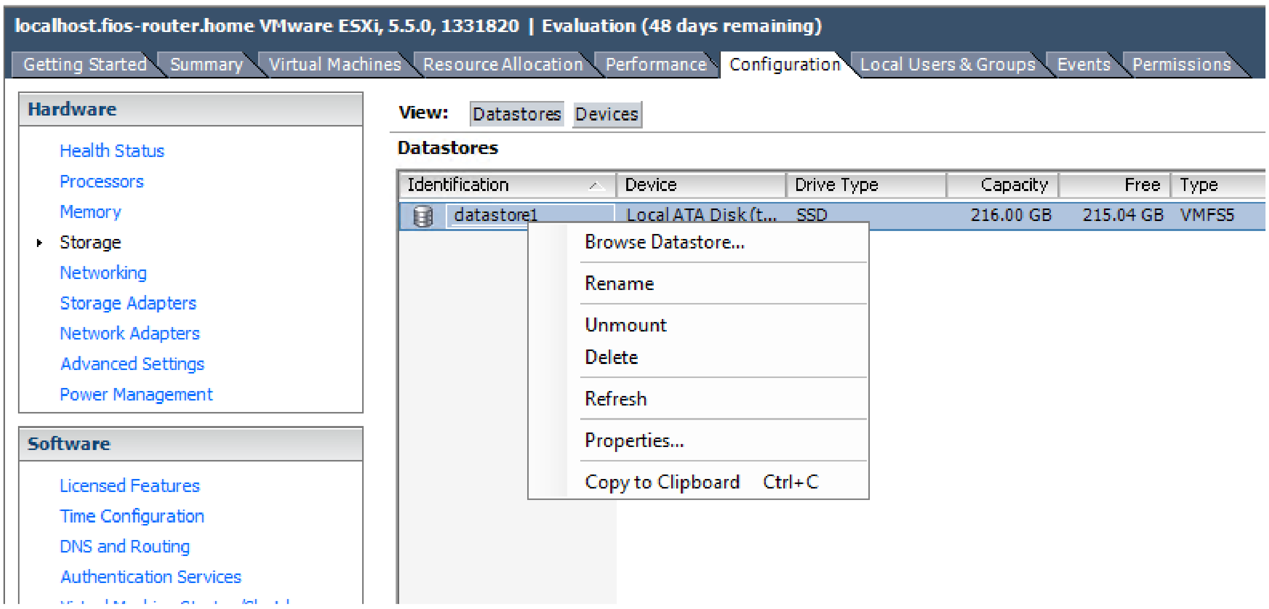
- RPS Building iPXE ROMs
- Configuring ESXi VMs to Use iPXE

Configuring ESXi VMs to Use iPXE

Updating ESXi VM to Use iPXE

The following steps are required to update an ESXi VM to use iPXE:

1. Select the ESXi host.
2. Select the Configuration tab.
3. Right click on the datastore, the ROMs will be copied to and select Browse Datastore...

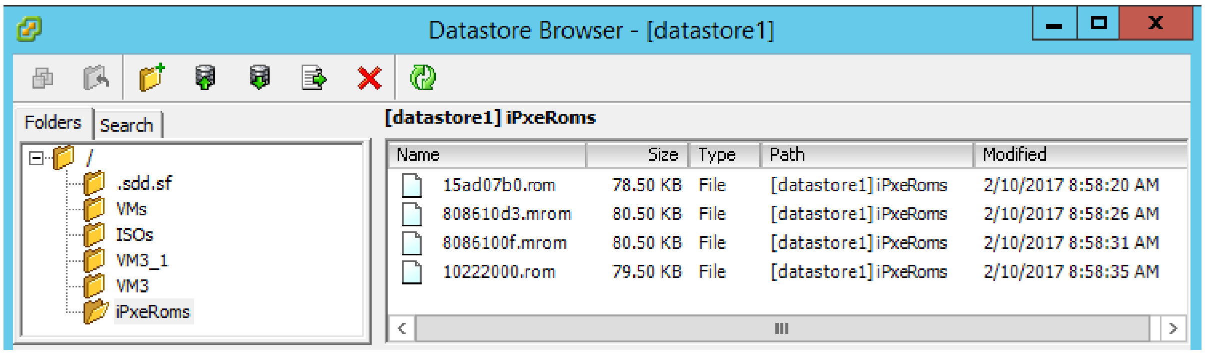


localhost.fios-router.home VMware ESXi, 5.5.0, 1331820 | Evaluation (48 days remaining)

Datastores

Identification	Device	Drive Type	Capacity	Free	Type
datastore1	Local ATA Disk (t...	SSD	216.00 GB	215.04 GB	VMFS5

1. The Datastore Browser window will appear.

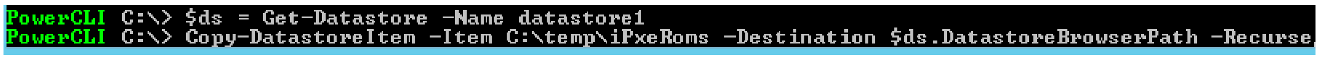


Datastore Browser - [datastore1]

[datastore1] iPxeRoms

Name	Size	Type	Path	Modified
15ad07b0.rom	78.50 KB	File	[datastore1] iPxeRoms	2/10/2017 8:58:20 AM
808610d3.mrom	80.50 KB	File	[datastore1] iPxeRoms	2/10/2017 8:58:26 AM
8086100f.mrom	80.50 KB	File	[datastore1] iPxeRoms	2/10/2017 8:58:31 AM
10222000.rom	79.50 KB	File	[datastore1] iPxeRoms	2/10/2017 8:58:35 AM

1. Create a folder if desired and select the Upload files to this datastore button to copy the iPXE ROMs to the ESXi host. The above screen shot shows all supported iPXE ROMs copied to the iPxeRoms folder.
2. The following commands show PowerCLI being used to copy the iPXE ROMs to the ESXi host.



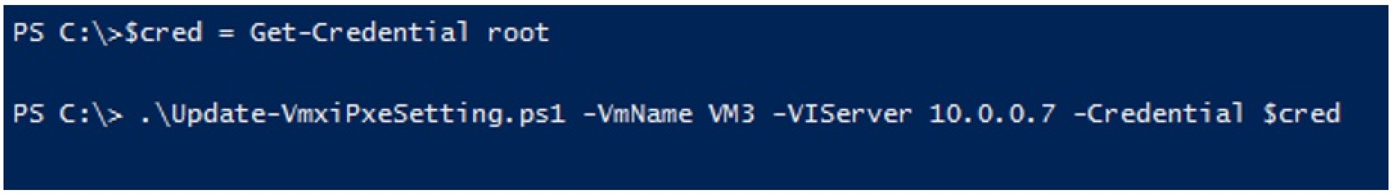
```
PowerCLI C:\> $ds = Get-Datastore -Name datastore1
PowerCLI C:\> Copy-DatastoreItem -Item C:\temp\iPxeRoms -Destination $ds.DatastoreBrowserPath -Recurse
```

NOTE

The Update-VmxiPxeSetting.ps1 script will perform all the necessary actions to update the VMs network adapter with the iPXE rom. The only parameters needed are the VM's name (VmName), ESXi host's name (VIServer), and credentials to connect to the ESXi host (Credential). The script requires PowerCLI to be installed.

Example Update

Here is an example of updating VM3 to leverage an iPXE ROM:



```
PS C:\>$cred = Get-Credential root

PS C:\> .\Update-VmxiPxeSetting.ps1 -VmName VM3 -VIServer 10.0.0.7 -Credential $cred
```

More Resources

- RPS Building iPXE ROMs
- RPS PXE Document

Using the Provisioning Service

Introduction

The RPS Provisioning Service is an HTTPS-based Web API hosted in IIS for use in brokering information from the RPS CMDB to a pre-execution environment such as iPXE for installation of a defined image and configuration. For instance, iPXE can be configured to "point to" the Provisioning Service which will return a boot script file for the MAC address requested. In this manner, iPXE will download and boot the image according to the script which has been defined in the CMDB.

Provisioning Service reads of RPS CMDB data

The Provisioning Service uses the RPS CMDB to query for several entity types and records:

1. Target Item (i.e. **MACAddress** property on a **NIC**)
2. Target Item's parent (i.e. **Computer** that *owns* the **NIC**)
3. Resource Item that is of type **BaseImage**
4. Resource Assignment that assigns the **BaseImage** to the Target Item
5. The **iPxeScript** property on the **BaseImage** Resource Item
6. The **{CustomProperty}** property on the **BaseImage** Resource Item

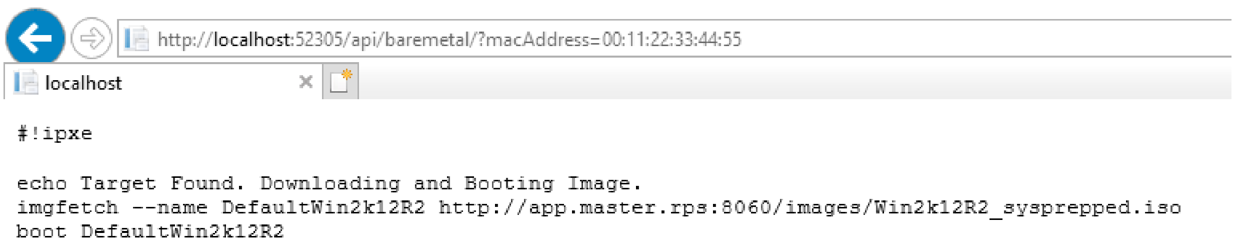
Provisioning Service writes to CMDB

In order to facilitate Target Item retrieval later in the provisioning process (including scenarios where certain environments are unable to perform requests using a URL query string with the MAC Address value), the Provisioning Service can take other inputs to be saved as properties on the parent Target Item. For instance, when iPXE first requests with its MAC Address, it can also send its **SMBiosProductName**, **SMBiosCurrentSpeed**, and **provisionIPAddress**, and potentially other key-value pairs. In this manner, later queries (e.g. through a Runbook) for the target without the MAC Address may be possible to uniquely identify said target.

Baremetal Provisioning Scenarios

Target found

The ideal scenario consists of all records and properties being found (e.g. **NIC** and **iPxeScript** on the **BaseImage**). When the **NIC** Target Item, the **Computer**, and the Resource Assignment that assigns the **BaseImage** to the **Computer** are found, the **iPxeScript** property of **BaseImage** will be returned from the service:


http://localhost:52305/api/baremetal/?macAddress=00:11:22:33:44:55
localhost

```
#!ipxe

echo Target Found. Downloading and Booting Image.
imgfetch --name DefaultWin2k12R2 http://app.master.rps:8060/images/Win2k12R2_sysprepped.iso
boot DefaultWin2k12R2
```

NOTE

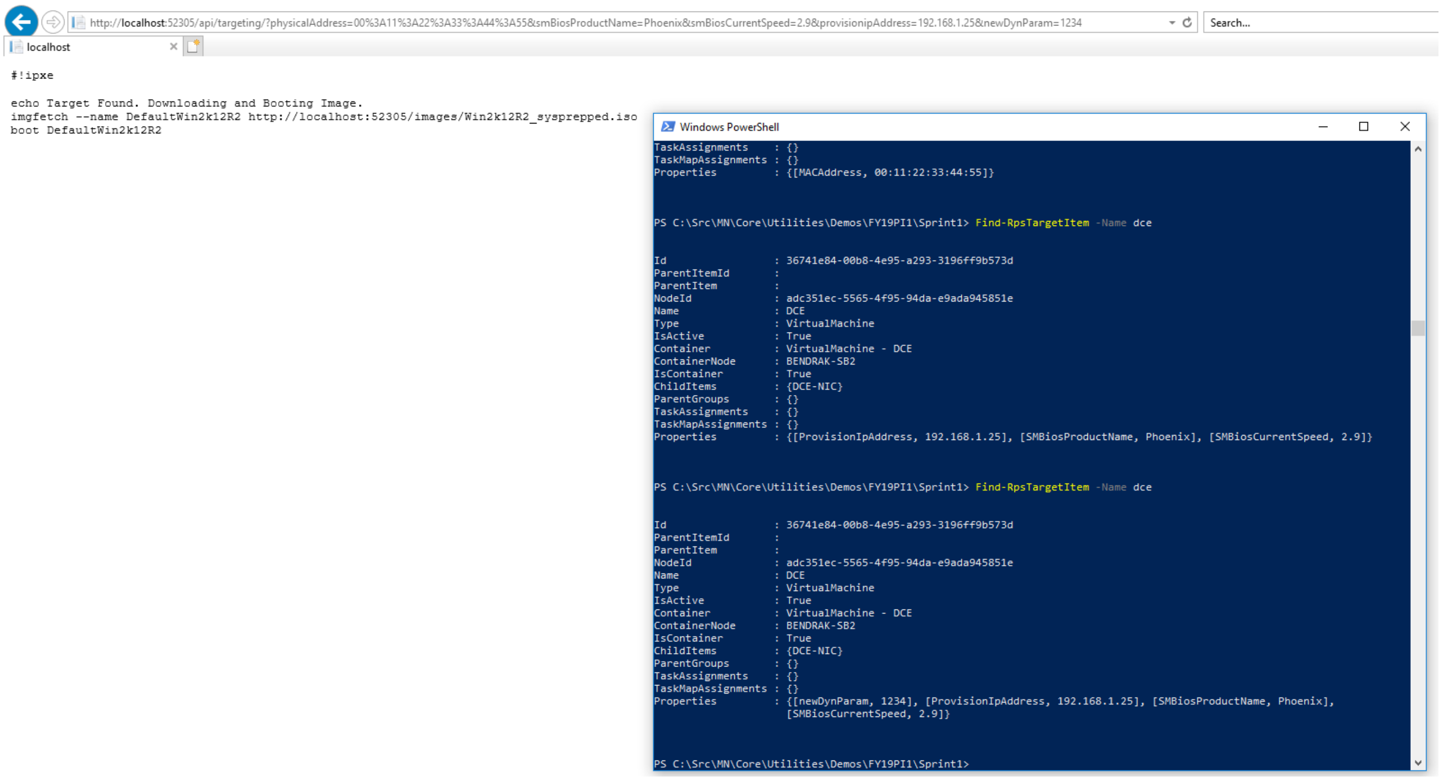
The examples include screenshots from a local iisexpress instance. When it is deployed in the RPS solution on IIS, the server and port name will ultimately differ.

In the case where the required records/properties are found, the **iPxeScript** stored in the CMDB is returned to the requesting client, verbatim.

When the script is executed by iPXE, the environment will attempt to download and install the image from specified in the iPXE Script.

Target found and additional query parameters added

If the Target Item is found, we can also specify additional parameters to add as properties to the parent Target Item. This example shows a different browser and a different *Target Found* iPXE Script, and also shows where **SMBiosProductName**, **SMBiosCurrentSpeed**, and **provisionIPAddress** where added to the CMDB through the Provisioning Service, and then a **newDynParam** added as well to showcase dynamic parameters.


http://localhost:52305/api/targeting/?physicalAddress=00%3A11%3A22%3A33%3A44%3A55&smBiosProductName=Phoenix&smBiosCurrentSpeed=2.9&provisionipAddress=192.168.1.25&newDynParam=1234
Search...
localhost
Windows PowerShell

```
#!ipxe

echo Target Found. Downloading and Booting Image.
imgfetch --name DefaultWin2k12R2 http://localhost:52305/images/Win2k12R2_sysprepped.iso
boot DefaultWin2k12R2
```

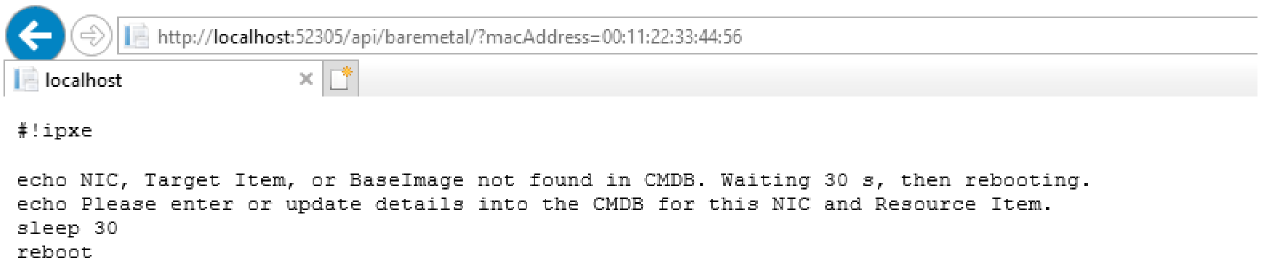
```
TaskAssignments    : {}
TaskMapAssignments : {}
Properties         : {[MACAddress, 00:11:22:33:44:55]}



PS C:\Src\MN\Core\Utilities\Demos\FY19PI1\Sprint1> Find-RpsTargetItem -Name dce


Id                 : 36741e84-00b8-4e95-a293-3196ff9b573d
ParentItemId       :
ParentItem         :
NodeId             : adc351ec-5565-4f95-94da-e9ada945851e
Name               : DCE
Type               : VirtualMachine
IsActive           : True
Container          : VirtualMachine - DCE
ContainerNode      : BENDRAK-SB2
IsContainer        : True
ChildItems         : {DCE-NIC}
ParentGroups       : {}
TaskAssignments    : {}
TaskMapAssignments : {}
Properties         : {[ProvisionIpAddress, 192.168.1.25], [SMBiosProductName, Phoenix], [SMBiosCurrentSpeed, 2.9]}



PS C:\Src\MN\Core\Utilities\Demos\FY19PI1\Sprint1> Find-RpsTargetItem -Name dce


Id                 : 36741e84-00b8-4e95-a293-3196ff9b573d
ParentItemId       :
ParentItem         :
NodeId             : adc351ec-5565-4f95-94da-e9ada945851e
Name               : DCE
Type               : VirtualMachine
IsActive           : True
Container          : VirtualMachine - DCE
ContainerNode      : BENDRAK-SB2
IsContainer        : True
ChildItems         : {DCE-NIC}
ParentGroups       : {}
TaskAssignments    : {}
TaskMapAssignments : {}
Properties         : {[newDynParam, 1234], [ProvisionIpAddress, 192.168.1.25], [SMBiosProductName, Phoenix],
                     [SMBiosCurrentSpeed, 2.9]}



PS C:\Src\MN\Core\Utilities\Demos\FY19PI1\Sprint1>
```

NOTE

The **macAddress** can be encoded with a **%3A** value in place of **:**. This is acceptable, as are **:** or **-**, so long as **macAddress** is explicitly specified in the URL query string.

Target not found

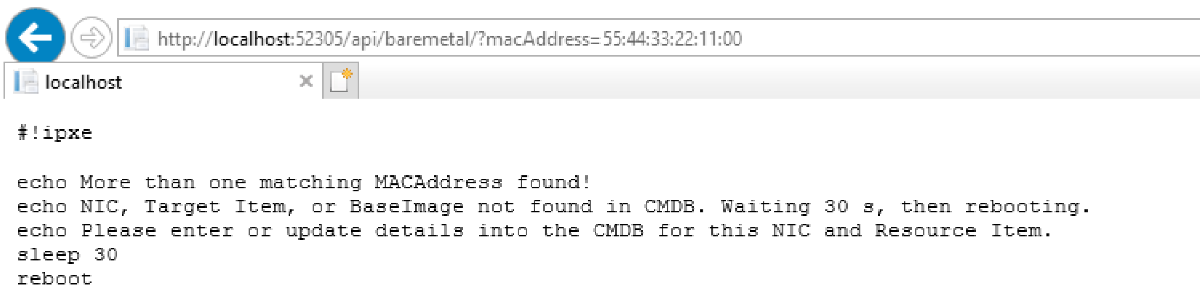
If the MAC address is not found as a property of any Target Item, a default iPXE script will be returned to the client. This script prints a message, will sleep for 30 s and then reboot:


http://localhost:52305/api/baremetal/?macAddress=00:11:22:33:44:56
localhost

```
#!ipxe

echo NIC, Target Item, or BaseImage not found in CMDB. Waiting 30 s, then rebooting.
echo Please enter or update details into the CMDB for this NIC and Resource Item.
sleep 30
reboot
```

Duplicate MAC found

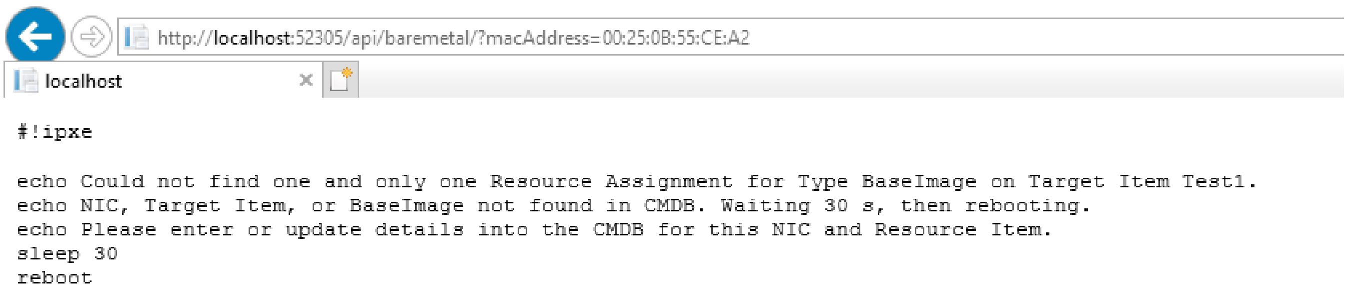
If more than one of the same MAC address is found in the CMDB, a default iPXE script will be returned to the requesting client:


http://localhost:52305/api/baremetal/?macAddress=55:44:33:22:11:00
localhost

```
#!ipxe

echo More than one matching MACAddress found!
echo NIC, Target Item, or BaseImage not found in CMDB. Waiting 30 s, then rebooting.
echo Please enter or update details into the CMDB for this NIC and Resource Item.
sleep 30
reboot
```

Resource Assignment cannot be determined

If there is not one (and only one) Resource Assignment, a default iPXE script will be returned to the requesting client:

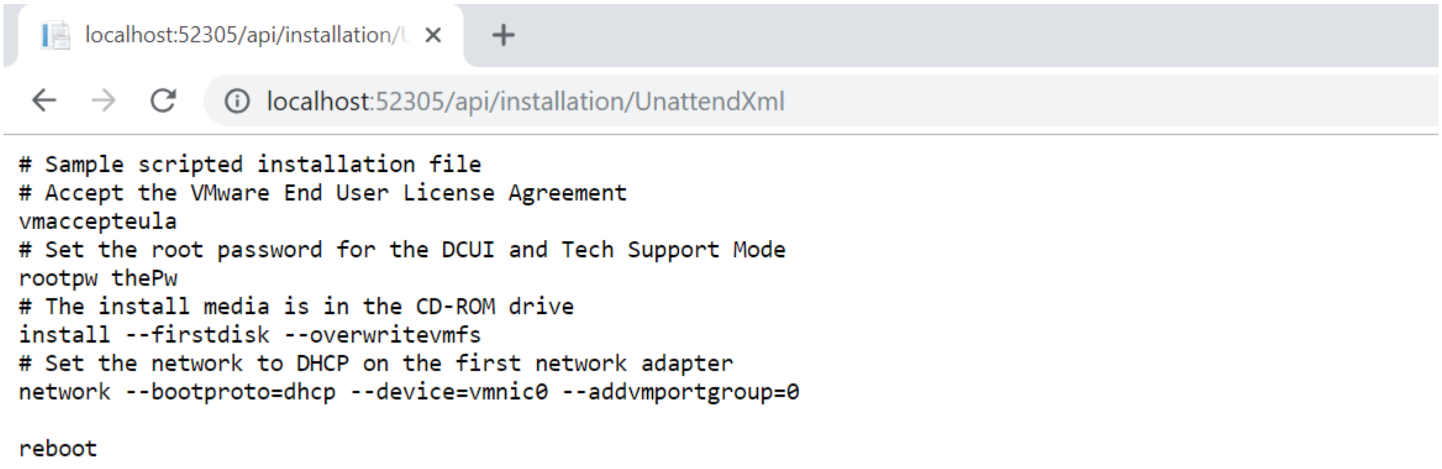

http://localhost:52305/api/baremetal/?macAddress=00:25:0B:55:CE:A2
localhost

```
#!ipxe

echo Could not find one and only one Resource Assignment for Type BaseImage on Target Item Test1.
echo NIC, Target Item, or BaseImage not found in CMDB. Waiting 30 s, then rebooting.
echo Please enter or update details into the CMDB for this NIC and Resource Item.
sleep 30
reboot
```

Installation Provisioning Scenarios

The subsequent iPXE operations, the Provisioning Service can be used to receive another scripted file for unattended installations which is commonly referred to as a *Kickstart* file. This script is to be hosted in the CMDB on the **BaseImage**, similar to the **iPxeScript**. Since the environment in this phase of the installation is not equipped to use dynamic parameters (such as the **macAddress** resolved by iPXE), the Provisioning Service will attempt to use the client's IP Address which ought to have been saved to the Target Item (e.g. DCE) during the baremetal provisioning sequence.

If the custom script is already stored on the **BaseImage**,the **ResourceAssignment** has a **ResourceStatus** of **Approved** and the **provisionIpAddress** on the Target Item matches that of the client, the client will receive this custom script, like:

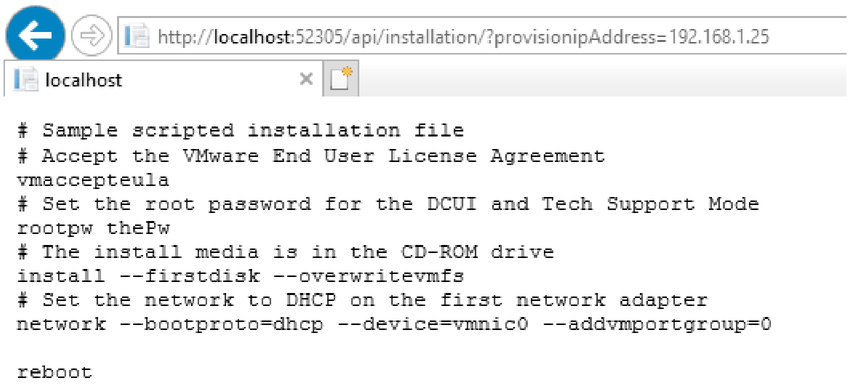

localhost:52305/api/installation/U
localhost:52305/api/installation/UnattendXml

```
# Sample scripted installation file
# Accept the VMware End User License Agreement
vmaccepteula
# Set the root password for the DCUI and Tech Support Mode
rootpw thePw
# The install media is in the CD-ROM drive
install --firstdisk --overwritevmfs
# Set the network to DHCP on the first network adapter
network --bootproto=dhcp --device=vmnic0 --addvmportgroup=0

reboot
```

NOTE

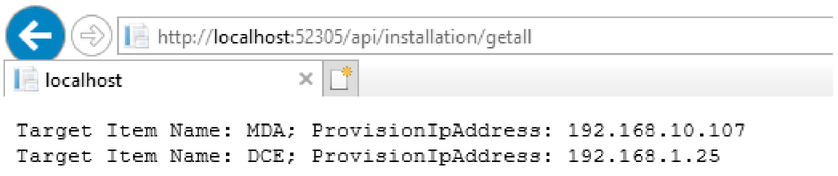
The Provisioning service can return any custom script. You can do this by calling https://localhost/api/installation/UnattendXml. This will return the value of the property **UnattendXml** stored on the **BaseImage**.

Of course, the IP Address can be specified explicitly, like:


http://localhost:52305/api/installation/?provisionipAddress=192.168.1.25
localhost

```
# Sample scripted installation file
# Accept the VMware End User License Agreement
vmaccepteula
# Set the root password for the DCUI and Tech Support Mode
rootpw thePw
# The install media is in the CD-ROM drive
install --firstdisk --overwritevmfs
# Set the network to DHCP on the first network adapter
network --bootproto=dhcp --device=vmnic0 --addvmportgroup=0

reboot
```

The **GetAll** action can also be used to see all Target Items with a **provisionIpAddress**:



```
Target Item Name: MDA; ProvisionIpAddress: 192.168.10.107
Target Item Name: DCE; ProvisionIpAddress: 192.168.1.25
```

Summary

The RPS Types have been updated to account for expected **Computer** definitions (e.g. MDA, DCE) as well as the Resource Item type for **BaseImage**. These definitions help facilitate the creation of appropriate records in the CMDB, including the customization of what is contained in the iPXE script. Without a matching Target Item (and parent Computer), and assigned Resource Item (**BaseImage**) in the CMDB, the Provisioning Service will default to functionality which informs, then reboots the client.

More Resources

- See the ProvisioningServiceDemo.ps1 in `Utilities\Demos\Provisioning` for examples when using RPS

RPS Tasking Guide

Last updated on August 26, 2021.

RPS tasking can provide a controlled, predictable method to automate complex tasks in correct sequence. A working knowledge of RPS entities and architecture is recommended prior to reading this guide. This guide provides an overview of RPS Tasking architecture, a list of supporting documentation, RPS task assignment creation, and examples in PowerShell code.

Overview

The center of RPS' Tasking capabilities is a **Task Assignment**. The task assignment is the smallest unit of work, a **Task Item**, assigned to a single **Target Item**. To perform a complex sequence of steps, RPS uses a **Task Map**, which describes a set of steps, what order they should be performed in, and what target items those steps apply to. The Task Map is just the blueprint of the process or orchestration, such as provisioning a new server stack, performing complex patching, or maintenance routines like issuing new certificates.

The result of assigning a task map to a target item is a **Task Map Assignment**, which is a set of task assignments that together, complete the complex process. A major benefit of RPS is the flexibility of Task Maps, which allows Administrators to author simple, reusable PowerShell Runbooks, and compose them together into a complex process across multiple devices.

Glossary

The following terms are used in the Tasking Guide

TERM	DESCRIPTION	ALIASES (*DEPRECATED**)
Runbook	PowerShell script which runs a simple task, usually run in SMA	
SMA	Server Management Automation - hosts & runs runbooks, used by RPS' Master Controller to run RPS Task Assignments	
Master Controller	RPS System Runbook which issues and monitors SMA jobs	
Task Item	Identifies a task (runbook)	TaskItem, Task
Task Map	Identifies a set of steps, what order they should be performed in, and what target items those steps apply to	TaskMap, Map
Step	Identify a single step within a Task Map	Task Map Step, TaskMapStep, *TaskMapDefinition**
Step Dependency	A dependency between a step and its dependent (previous) step	TaskMapStepDependency, *TaskMapDefDependency**
Step Filter	A filter to narrow what target items the step applies to	TaskMapStepFilter, *TaskMapDefFilter**
Target Item	A device which is the target of a task item or task map	TargetItem, Target
Task Assignment	Assignment of a task item to a target. Corresponds to a single job in SMA	TaskAssignment, Assignment

TERM	DESCRIPTION	ALIASES (*DEPRECATED**)
Map Assignment	The assignment of a task map to a target and its descendants. The result is a set of Task Assignments that together make up a complex orchestration	TaskMapAssignment, Orchestration
Direct Assignment	The assignment of a task item directly to a target, without the use of a Task Map	

Target Item User Actions

The states in which actions on a Target Item may be classified are detailed below.

STATE	DESCRIPTION	CRITERIA
Pending	An action on a Target Item that is Pending	Task Assignment(s) assigned to the Target Item are Pending user action
Optional	An action on a Target Item that is Optional	Task Assignment(s) assigned to the Target Item have a state of Not Ready and are flagged as Optional
Retry	An action on a Target Item that is Retryable	Task(s) assigned to the Target Item have a state of Cancelled or ErrorStop

Task Map Authoring

A **Task Map** is a reusable blueprint for a set of steps that need to be performed on one or more devices. The Rps-Api module and cmdlets allow a user to quickly create complex task maps.

Example: Create a Task Map

This example demonstrates the method for creating a Task Map using the Rps-Api module. The example is a Task Map which first configures a hypervisor, and then creates a new VM.

Step 1: Create Task Map

Provide the Name and Type

```
$map = New-RpsTaskMap -Name "Provision-VMs" -Type "Provision"
```

Step 2: Create Task Items

Next, create a Task Item for the runbooks that do the work.

```
$task1 = New-RpsTaskItem -WorkflowName "Configure-HyperV"
$task2 = New-RpsTaskItem -WorkflowName "New-VirtualMachine"
```

Step 3: Create Steps

Next, create a step for each of the Task Items. You can specify the Task Item using the "RunbookName", "TaskItem", or "TaskItemId" parameter.

```
$step1 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Configure-HyperV" -TargetItemType "Computer"
$step2 = New-RpsTaskMapStep -TaskMap $map -TaskItem $task1 -TargetItemType "Computer"
$step3 = New-RpsTaskMapStep -TaskMapId $map.Id -TaskItemId $task2.Id -TargetItemType "VirtualMachine"
```

NOTE

The **TargetItemType** parameter is required and is used to determine which type of Target Item to assign to. To use more advanced filtering, see Step 5: Create Filters, below.

Step 4: Create Dependencies

Create a dependency between $step1 and $step2. RPS uses dependencies to evaluate when a task assignment is ready to be run. In this example, step 2 depends on step1, so it will run only after step 1 has completed.

```
New-RpsTaskMapStepDependency -PreviousStep $step1 -Step $step2
```

Step 5: Create Filters

Use Step Filters to further restrict what Target Items a Step applies to. For our example, we may only want to provision Hyper-V VMs, so we'll make sure the Target VMs have a property called "HostType" with a value of "HyperV"

```
New-RpsTaskMapStepFilter -TaskMapStep $step2 -PropertyName "HostType" -PropertyValue "HyperV"
```

Example: Simplified Task Map

The example above can be simplified to allow easier Task Map creation. Here's an example of the same Task Map, simplified.

```
$map = New-RpsTaskMap -Name "Provision-VMs" -Type "Provision"
$step1 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Configure-HyperV" -TargetItemType "Computer"
$step2 = New-RpsTaskMapStep -TaskMap $map -RunbookName "New-VirtualMachine" -TargetItemType "VirtualMachine"
-Dependencies @($step1) -Filters @{ HostType = "HyperV" }
```

NOTE

The example assumes that the Task Items were created. Typically, Runbooks will be authored and imported into RPS as Task Items before Task Maps are created.

Target Matching

When a Task Map is assigned to a Target Item, each Step is compared to the Target and its descendant items. By default, RPS doesn't enforce that a step matches a target. By default, RPS also allows a Step to match multiple Target Items, though sometimes the desired behavior is to match a single item only.

To designate a Step as required, use the optional parameter, `IsTargetRequired`.

To designate a Step to allow or prevent multiple matching targets, use the optional parameter, `AllowMultipleTargets`.

WARNING

A Task Map will fail if it is assigned to the same Target Item multiple times while `AllowMultipleTargets = false`, because this configuration does not allow multiple targets.

Example: Required Target Item, No Multiples

In our sample Task Map, we probably want the `Configure-HyperV` step to apply to a single Computer hosting HyperV. We will change the first Step to require a Target and disallow multiple matches. We won't change the second step, meaning that this Task Map will work for a Computer with zero or more VMs.

```
$map = New-RpsTaskMap -Name "Provision-VMs" -Type "Provision"
$step1 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Configure-HyperV" -TargetItemType "Computer" -
IsTargetRequired $true -AllowMultipleTargets $false
$step2 = New-RpsTaskMapStep -TaskMap $map -RunbookName "New-VirtualMachine" -TargetItemType "VirtualMachine"
-Dependencies $step1 -Filters @{ HostType = "HyperV" }
```

Task Map Assignment

After creating a Task Map, assign it to a Target Item to create a **Task Map Assignment**. The map assignment will contain one or more Task Assignments that apply to the Target Item and its descendants.

NOTE

In v2.2 and earlier, RPS only allowed Task Maps to be assigned to root Target Items (Containers). As of v2.3, RPS allows a Task Map to be assigned to any Target Item, allowing more flexibility in how Target Items are structured, and more reuse of Task Maps and Runbooks. This has the side effect that $taskMap.GetContainers() may no longer retrieve the actual targets of the Task Map.

Assign Task Map to Target Item

The example below will create a Task Map, a Target Computer with 2 VMs, and assign the Map to the Target.

```
# Create Task Map
$map = New-RpsTaskMap -Name "Provision-VMs" -Type "Provision"
$step1 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Configure-HyperV" -TargetItemType "Computer" -
IsTargetRequired $true -AllowMultipleTargets $false
$step2 = New-RpsTaskMapStep -TaskMap $map -RunbookName "New-VirtualMachine" -TargetItemType "VirtualMachine"
-Dependencies $step1 -Filters @{ HostType = "HyperV" }

# Create Computer & 2 VMs
$server = New-RpsTargetItem -Type Computer -Name "Server-01"
$vm1 = New-RpsTargetItem -ParentItem $server -Type VirtualMachine -Name "VM-01" -Properties @{ HostType =
"HyperV" }
$vm2 = New-RpsTargetItem -ParentItem $server -Type VirtualMachine -Name "VM-02" -Properties @{ HostType =
"HyperV" }

# Assign Task Map to Computer
New-RpsTaskAssignment -TaskMap $map -TargetItem $server
```

The result of this assignment is a Map Assignment consisting of 3 Task Assignments.

STEP	TASK	TARGET	DEPENDENCIES
1-1	Configure-HyperV	Server-01	
2-1	New-VirtualMachine	VM-01	Step 1-1
2-2	New-VirtualMachine	VM-02	Step 1-1

Assign Task Map to Target Group

A Task Map can also be assigned to a Target Group. The Target Group is used simply as a collection of related Target Items. The assignment will result in a new Map Assignment to each Target in the Group, and each Target will be validated separately.

The example below will create a Task Map, a Target Group with 2 Servers, and assign the Map to the Group.

```
# create a server with 2 VMs
$vmProps = @{ Type = "VirtualMachine"; Properties = @{ HostType = "HyperV" } }
$server = New-RpsTargetItem -Type Computer -Name "Server01"
$vm1 = New-RpsTargetItem -ParentItem $server -Name "VM-01" @vmProps
$vm2 = New-RpsTargetItem -ParentItem $server -Name "VM-02" @vmProps

# create a server with 1 VM
$server2 = New-RpsTargetItem -Type Computer -Name "Server02"
$vm3 = New-RpsTargetItem -ParentItem $server2 -Name "VM-03" @vmProps

# create a group with both servers
$group = New-RpsTargetGroup -Name "Servers" -Type "Server"
$group.AddChildren($server, $server2)
$group.Update()

# assign the map to the group
New-RpsTaskAssignment -TaskMap $map -TargetGroup $group
```

Assign a Task Map to run on the Local Node

By default, Task Assignments will be executed on the Target's Node. Optionally, a Task Map can also be assigned to a Target but executed on the Local (current) Node, instead of the Target's Node. This may be useful when building a child Node from a parent Node. The work to provision the child Node's Virtual Machines will happen locally (on the parent node), but the target computers are assigned to the child node.

Assign a Task Map to a Target and run on the Local Node by using the `-RunOnLocalNode` switch with the `New-RpsTaskAssignment` cmdlet.

Example: Assign a Task Map to Child Target but run on Parent Node

```
# Create Task Map
$map = New-RpsTaskMap -Name "Provision-VMs" -Type "Provision"
$step1 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Configure-HyperV" -TargetItemType "Computer" -
IsTargetRequired $true -AllowMultipleTargets $false
$step2 = New-RpsTaskMapStep -TaskMap $map -RunbookName "New-VirtualMachine" -TargetItemType "VirtualMachine"
-Dependencies $step1 -Filters @{ HostType = "HyperV" }

# Create Computer & 2 VMs
$server = New-RpsTargetItem -Type Computer -Name "Server-01" -Node $childNode
$vm1 = New-RpsTargetItem -ParentItem $server -Type VirtualMachine -Name "VM-01" -Properties @{ HostType =
"HyperV" }
$vm2 = New-RpsTargetItem -ParentItem $server -Type VirtualMachine -Name "VM-02" -Properties @{ HostType =
"HyperV" }

# Assign Task Map to Computer but run on current Node
New-RpsTaskAssignment -TaskMap $map -TargetItem $server -RunOnLocalNode
```

Scheduled Task Assignments

RPS v2.3 supports the ability to schedule task assignments. These task assignments will not be executed until the scheduled date.

To schedule a Task Map Assignment, use the optional parameter, `StartDate`.

Example: Scheduled Task Map Assignment

```
New-RpsTaskAssignment -TaskMap $map -TargetItem $server -StartDate "8/27/18 16:00"
```

Assignment Validation

When assigning a Task Map to a Target Item, there may be some steps that don't meet the Task Map's requirements. For example, if a Step requires a matching Target, but none is found, the assignment will fail and an error returned to the user.

Invalid Targets

This code:

```
# Assign Task Map to VM1
New-RpsTaskAssignment -TaskMap $map -TargetItem $vm1
```

Will result in this message:

> WARNING: Error assigning Task Map: Provision-VMs to Container: VirtualMachine-VM-01
>
> **Step requires a matching target item.**

Duplicate Assignments

RPS does not allow the same Task Map to be assigned to the same Target Item multiple times.

This code:

```
# Assign Task Map to Server01 Twice
New-RpsTaskAssignment -TaskMap $map -TargetItem $server
New-RpsTaskAssignment -TaskMap $map -TargetItem $server
```

Will result in this message:

> WARNING: Error assigning Task Map: Provision-VMs to Container: Computer-Server01
>
> **Container is already assigned to the Task Map.**

NOTE

The restriction of duplicate assignments is largely in place for backwards-compatibility reasons, and will likely be removed from RPS in the future. Conceptually, a Task Map represents a process that may run multiple times on the same target devices. For now, the two alternatives are to clone a task map in order to run it again, or reset the task map assignment so it will run again.

Dependency Scope

If we change the "Provision-VMs" Task Map to include a third step which makes sure the VM is started and waits for the OS to boot, it would have a dependency on Step 2.

```
$map = New-RpsTaskMap -Name "Provision-VMs" -Type "Provision"
$step1 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Configure-HyperV" -TargetItemType "Computer" -
IsTargetRequired $true -AllowMultipleTargets $false
$step2 = New-RpsTaskMapStep -TaskMap $map -RunbookName "New-VirtualMachine" -TargetItemType "VirtualMachine"
-Dependencies $step1
$step3 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Start-VirtualMachine" -TargetItemType
"VirtualMachine" -Dependencies $step2
```

If we assign that map to our server and 2 VMs, the following Task Assignments will be created:

STEP	TASK	TARGET	DEPENDENCIES
1-1	Configure-HyperV	Server-01	
2-1	New-VirtualMachine	VM-01	Step 1-1
2-2	New-VirtualMachine	VM-02	Step 1-1

STEP	TASK	TARGET	DEPENDENCIES
3-1	Start-VirtualMachine	VM-01	**Steps 2-1, 2-2**
3-2	Start-VirtualMachine	VM-02	**Steps 2-1, 2-2**

Looking at the dependencies, you can see that the VM-01 won't start (Step 3-1) until VM-02 has been created (Step 2-2). Likewise, VM-02 waits on VM-01. If we add more Steps that apply to VMs and more VMs to our Server, we'd be creating many dependencies that aren't necessary.

Example 1: Use Scope to narrow Dependencies

Instead of creating dependencies to every target that matched a previous step, we can specify a more narrow Scope when we define a Dependency. In RPS v2.3, we've included a new `Scope` parameter when creating Dependencies.

```
$map = New-RpsTaskMap -Name "Provision-VMs" -Type "Provision"
$step1 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Configure-HyperV" -TargetItemType "Computer" -
IsTargetRequired $true -AllowMultipleTargets $false
$step2 = New-RpsTaskMapStep -TaskMap $map -RunbookName "New-VirtualMachine" -TargetItemType "VirtualMachine"
-Dependencies $step1
$step3 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Start-VirtualMachine" -TargetItemType
"VirtualMachine"
New-RpsTaskMapStepDependency -Step $step3 -PreviousStep $step2 -Scope Self
```

Example 2: Use Scope in-line

NOTE

Scope can be added to a taskmapstep as a taskmapstep parameter. The scope will act on the dependencies parameters supplied.The scope will default to target if not specified.

```
New-RpsTaskItem -WorkflowName "Configure-HyperV"
New-RpsTaskItem -WorkflowName "New-VirtualMachine"
New-RpsTaskItem -WorkflowName "Start-VirtualMachine"

$map = New-RpsTaskMap -Name "Provision-VMs" -Type "Provision"
$step1 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Configure-HyperV" -TargetItemType "Computer" -
IsTargetRequired $true -AllowMultipleTargets $false
$step2 = New-RpsTaskMapStep -TaskMap $map -RunbookName "New-VirtualMachine" -TargetItemType "VirtualMachine"
-Dependencies $step1 -Scope Self
$step3 = New-RpsTaskMapStep -TaskMap $map -RunbookName "Start-VirtualMachine" -TargetItemType
"VirtualMachine"
```

If we assign that map to our server and 2 VMs, the following Task Assignments will be created:

STEP	TASK	TARGET	DEPENDENCIES
1-1	Configure-HyperV	Server-01	
2-1	New-VirtualMachine	VM-01	Step 1-1
2-2	New-VirtualMachine	VM-02	Step 1-1
3-1	Start-VirtualMachine	VM-01	Step 2-1
3-2	Start-VirtualMachine	VM-02	Step 2-2

Table: Dependency Scopes

The following Scopes are supported:

SCOPE	DESCRIPTION
Target	Indicates that dependencies will be to the Target of the Task Map, which is the assigned Target Item and all its descendants. This is the default value and the behavior of RPS prior to v2.3.
Parent	Indicates that dependencies will be to the Parent Target Item of the previous step and any descendants.
Self	Indicates that dependencies will be to the Target Item of the previous step and any descendants.

Assignment Code Examples

Assign Task Item to Target Item for Execution

Both the Task Item and the Target Item objects must exist prior to assigning a Task Item to a Target Item. The following example shows the method of assignment. You can add a `StartDate` to the task assignment so Master-Controller wont start execution until the DateTime has been met.

```
$taskAssignments = New-RpsTaskAssignment -TaskItem $AlwaysCompletes -TargetItem $targetItem -StartDate (Get-Date).AddMinutes(15)
```

Assign Task Map to Target Item for Execution

Both the Task Map and the Target Item objects must exist prior to assigning a Task Map to a Target Item. The following example shows the method of assignment.

```
$taskAssignments = New-RpsTaskAssignment -TaskMap $TaskAssignmentTaskMap -TargetItem $targetItem
```

Assign Task Item to Target Group for Execution

Both the Task Item and the Target Group objects must exist prior to assigning a Task Item to a Target Group. The following example shows the method of assignment.

```
$taskAssignments = New-RpsTaskAssignment -TaskItem $AlwaysCompletes -TargetGroup $targetGroup
```

Assign Task Map to Target Group for Execution

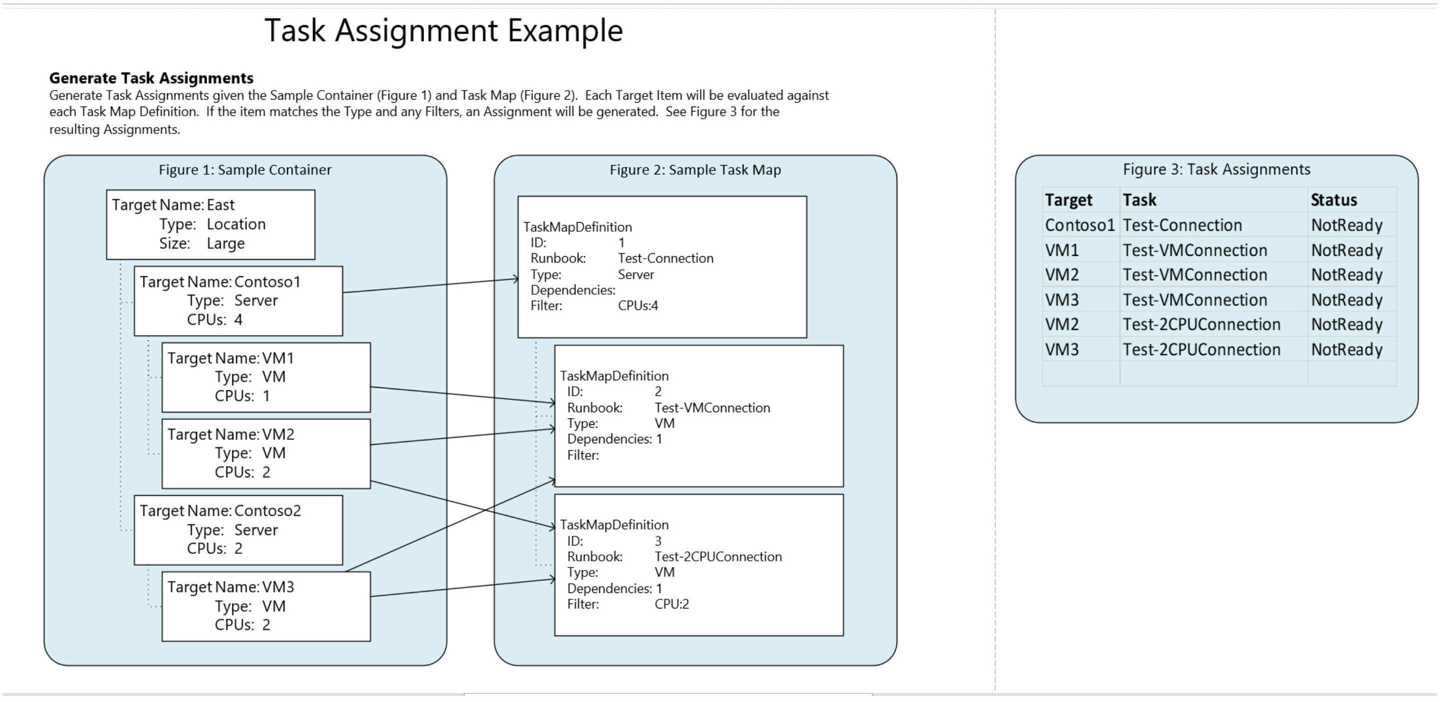
Both the Task Map and the Target Group objects must exist prior to assigning a Task Map to a Target Group. The following example shows the method of assignment.

```
$taskAssignments = New-RpsTaskAssignment -TaskMap $TaskAssignmentTaskMap -TargetGroup $targetGroup
```

More Resources

- RPS Software Design
- RPS Master Controller Design
- RPS Task Assignment Diagram
- Authoring RPS Runbooks

RPS Task Assignment Diagram



Task Assignment Example

Generate Task Assignments

Generate Task Assignments given the Sample Container (Figure 1) and Task Map (Figure 2). Each Target Item will be evaluated against each Task Map Definition. If the item matches the Type and any Filters, an Assignment will be generated. See Figure 3 for the resulting Assignments.

Figure 1: Sample Container
Target Name: East Type: Location Size: Large
Target Name: Contoso1 Type: Server CPUs: 4
Target Name: VM1 Type: VM CPUs: 1
Target Name: VM2 Type: VM CPUs: 2
Target Name: Contoso2 Type: Server CPUs: 2
Target Name: VM3 Type: VM CPUs: 2
Figure 2: Sample Task Map
TaskMapDefinition ID: 1 Runbook: Test-Connection Type: Server Dependencies: Filter: CPUs:4
TaskMapDefinition ID: 2 Runbook: Test-VMConnection Type: VM Dependencies: 1 Filter:
TaskMapDefinition ID: 3 Runbook: Test-2CPUConnection Type: VM Dependencies: 1 Filter: CPU:2

Figure 3: Task Assignments

Target	Task	Status
Contoso1	Test-Connection	NotReady
VM1	Test-VMConnection	NotReady
VM2	Test-VMConnection	NotReady
VM3	Test-VMConnection	NotReady
VM2	Test-2CPUConnection	NotReady
VM3	Test-2CPUConnection	NotReady

More Resources

- RPS Software Design
- RPS Master Controller Design
- RPS Tasking Guide
- Authoring RPS Runbooks

Authoring RPS Runbooks

Runbooks are PowerShell scripts (or workflows) that can be executed within a job host such as SMA. **RPS Runbooks** are PowerShell Runbooks that are designed to run within the context of RPS. In general, RPS Runbooks provide concise, reusable functionality, and use the Rps-Api PowerShell module to access and update CMDB data.

The terms "Runbook", "Workflow" and "Task Item" are all used interchangeably within RPS to refer to an RPS Runbook. The assignment of a single Runbook to a Computer (Target) for execution is a **Task Assignment**, which is executed as one job in SMA. The Task Assignment is the smallest unit of work in RPS, and its status is tracked in the RPS CMDB.

Glossary

The following are common terms used within RPS, Runbooks and Task Automation

TERM	DEFINITION
RPS	Rapid Provisioning System
CMDB	RPS [Content Management] Database
Rps-Api	PowerShell Module used to access and manipulate RPS configuration and task data
SMA	Server Management Automation - hosts & runs runbooks, used by RPS' Master Controller to run RPS Task Assignments
Runbook	PowerShell script which runs a simple task, usually run in SMA
Master Controller / MC	RPS System Runbook which issues and monitors SMA jobs
Task Item	Identifies a task (runbook)
Task Map	Identifies a set of steps, what order they should be performed in, and what type of target items the steps apply to
Step	Identify a single step within a Task Map
Target Item	A device which is the target of a task item or task map
Task Assignment	Assignment of a task item to a target. Corresponds to a single job in SMA
Map Assignment	The assignment of a task map to a target and its descendants. The result is a set of Task Assignments that together make up a complex orchestration

Task Assignment States

The following table defines the Task Assignment States used by RPS. Within PowerShell, the available states can be accessed via the `$Rps.TaskStates` variable.

TERM	DEFINITION
NotReady	Task Assignment is new or is waiting on other jobs to complete, and not ready to be executed.

TERM	DEFINITION
Ready	Task Assignment is ready to be started, and will be started by SMA.
Running	Task Assignment was started as a job in SMA (or Direct) and is currently executing.
Completed	Task Assignment is complete, and processing of other Task Assignments in the Orchestration will continue.
ErrorContinue	Task Assignment encountered an error, but processing of Orchestration will continue.
ErrorStop	Task Assignment failed and processing of Orchestration should stop.
Cancelled	Task Assignment was stopped, either from SMA or manually, and processing of Orchestration may stop.
PendingUserAction	Task Assignment requires user approval before Orchestration can continue. See section on User Approvals.
Retry	Task Assignment should be retried due to a failure. See section on Retry.

Hosting Runbooks in SMA vs. Direct Execution

A full instance of RPS (RPS Node) includes the CMDB as a SQL Server database and an instance of SMA to host and execute RPS runbooks. Master Controller is a system runbook that runs every 60 seconds and monitors the CMDB and SMA to manage jobs. It is the coordinator between **Map Assignments** and individual SMA Jobs.

With RPS v2.3.2, some of the functionality that MC uses is exposed directly via the Rps-Api module, so that Task Assignments and Task Map Assignments (Orchestrations) can be executed directly in PowerShell. This is useful for an initial RPS Install process, where SMA and MC may not be available. It also helps for developing, testing and troubleshooting runbooks. See documentation around the `Invoke-RpsTaskAssignment` and `Invoke-RpsTaskMapAssignment` cmdlets for more information.

RPS Runbook Guidance

The following are the three primary guidelines for authoring RPS Runbooks.

1. Runbooks are PowerShell scripts saved as `.ps1` files in the `\Runbooks` folder, using the Verb-Noun naming convention.
2. Runbooks contain a single mandatory parameter, `[guid] $TaskAssignmentId`, to identify the Task Assignment.
3. Runbooks use the Rps-Api to access and update CMDB data, log important information, and update Task Assignment State.

Example Runbook

The following example is a simplified version of the `Wait-Random.ps1` runbook, which can be used for guidance as well as sample task execution.

```
<# ... #>
[CmdletBinding()]
param
(
    [Parameter(Mandatory = $true)]
    [System.Guid]
    $TaskAssignmentId
)

#Requires -Modules Rps-Api

# Load Task Assignment and Target Item
$workflowName = $MyInvocation.MyCommand
$taskAssignment = Get-RpsTaskAssignment -Id $TaskAssignmentId
if (-not $taskAssignment)
{
    Write-RpsLogItem @LogError -MessageTemplate $LoggingEvents.TaskAssignmentNotFound -Properties
($TaskAssignmentId, $workflowName)
    throw "Task Assignment not loaded properly."
}

$targetItem = $taskAssignment.TargetItem
Write-RpsLogItem @LogInformation -MessageTemplate $LoggingEvents.StartingRunbook -Properties $workflowName,
$TaskAssignmentId
Update-RpsTaskAssignment -TaskAssignment $taskAssignment -TaskState Running -StatusMessage "$workflowName is
running on $($targetItem.Name)"

# Generate a random sleep time from 1-5 sec
$wait = Get-Random -Min 1 -Max 5
$waitMessage = 'Assignment: {TaskAssignmentId} on {TargetItem} waiting for {Wait} sec'
$logProps = @($TaskAssignmentId, $targetItem.Name, $wait)
Write-RpsLogItem -Level Information -Component Master -MessageTemplate $waitMessage -Properties $logProps

# Sleep
Start-Sleep -s $wait

Update-RpsTaskAssignment -TaskAssignment $taskAssignment -TaskState Completed -StatusMessage "Completed"
```

Updating Task Assignment State from a Runbook

To reduce boilerplate code, RPS will manage the Task Assignment's state as follows:

1. When a Task Assignment is started (via SMA or Direct), its `TaskState` is automatically set to **Running**.
2. When a Task Assignment is completed (via SMA or Direct), and its `TaskState` is **Running**, the `TaskState` is automatically set to **Completed**.
3. When a Task Assignment fails (via SMA or Direct), and its `TaskState` is **Running**, the `TaskState` is automatically set to **ErrorStop**.

In the example runbook, notice that the `TaskState` is never updated to Running or Completed.

To set the state manually, for error handling or a custom workflow, use the `Update-RpsTaskAssignment` cmdlet and the common Task States listed above.

Example: Manually set State

This example manually sets the State to **ErrorContinue** if an error occurs in the `try` block.

```
try
{
    # do work here
}
catch
{
    # catch error but allow orchestration to continue
    # additional logging or error handling here
    Update-RpsTaskAssignment -TaskAssignment $taskAssignment -TaskState $Rps.TaskStates.ErrorContinue -
Message 'An error occurred in non-critical process.'
}
```

Logging from a Runbook

RPS supports structured logging via the `Write-RpsLogItem` cmdlet. See the Rps-Api documentation for full description and examples. For RPS Runbooks, it's important to understand how to associate information to the Task Assignment for diagnostics or telemetry.

To associate log data with an executing Task Assignment, you must supply a formatted message template with named replacement tokens and the Task Assignment's Id.

Example: Logging Information to the Task Assignment

This example, from the `Wait-Random` runbook, shows how to properly log information to a Task Assignment from a runbook. Notice the template is a string with replacement tokens and the properties are simply an object array to associate to the tokens in the template.

```
# Generate a random sleep time from 1-5 sec
$wait = Get-Random -Min 1 -Max 5
$waitMessage = 'Assignment: {TaskAssignmentId} on {TargetItem} waiting for {Wait} sec'
$logProps = @($TaskAssignmentId, $target.Name, $wait)
Write-RpsLogItem -Level Information -MessageTemplate $waitMessage -Properties $logProps
```

Advanced Runbook Guidance

1. Establishing a secure connection to a remote computer
2. Scheduled/Recurring Task Assignments
3. Retrying Task Assignments
4. Require User Approval

Connect to Target Computer

Many Runbooks will need to connect to the Target (Computer) to perform their duty. To connect, you must get the appropriate credential and then establish a secure connection. In RPS Runbooks, use the `Get-RpsCredential` to load the right credential for the target. Then use `New-SecureSession` from Rps-Api to make the connection.

```
#Requires -Modules Rps-Api

# Load assignment and target
$assignment = Get-RpsTaskAssignment -Id $TaskAssignmentId
$target = $assignment.TargetItem

try
{
    # Retrieve LocalAdmin credentials for target computer
    $localAdmin = Get-RpsCredential -TargetItem $target -Role 'LocalAdmin'

    # Establish connection to target
    $session = New-SecureSession -IPAddress $target.IPAddress -Type PsSession -Credential $localAdmin

    # Do work on target
}
finally
{
    if ($session)
    {
        # Clean up connection
        Remove-PSSession -Session $session
    }
}
```

Scheduled / Recurring Task Assignments

RPS supports scheduling of Task Assignments and Map Assignments. See the RPS Tasking Guide for examples of scheduling new Map Assignments and Task Assignments.

To have a Task Assignment repeat or recur, manually adjust the `StartDate` and set `TaskState` to **Ready** and RPS will run the Task Assignment again.

CAUTION

To prevent MC from inadvertently completing a recurring task, make sure to clear the SMAJobId when resetting the state.

Example: Set Task Assignment to Recur every 15 minutes

Use the following code in a runbook to have the Task Assignment recur.

TIP

To set a limit on execution counts, add a custom property to the TaskAssignment and logic to update it after each run.

```
$taskAssignment.StartDate = (Get-Date).AddMinutes(15)
$taskAssignment.TaskState = $Rps.TaskStates.Ready
$taskAssignment.SmaJobGuid = $null
Update-RpsTaskAssignment $taskAssignment
```

Retry Failed Task Assignments

RPS considers the **Retry** state similar to **Ready**. The only difference is that Retry indicates that the Task Assignment failed and is executing again, instead of running for the first time.

Starting with RPS v2.3.2 there is limited support for automatic retries, which was added to support some Rps Install features. To use an automatic retry, set a `RetryCount` property on the Task Assignment. If the Task Assignment fails, RPS will automatically retry it and decrement the `RetryCount` until all retries have been attempted.

This behavior can be simulated in runbooks and may be promoted to a built-in feature in RPS and SMA in a future release. This would also include a `RetryCount` option in the UI or PowerShell when designing a Task Map.

Example: Use RetryCount to limit failures

Place the following sample code into the error handling section of a runbook to implement a limited # of retries. This will only retry a TaskAssignment that already has the `RetryCount` set.

```
$retryCount = [Rps.Api.Utils.Extensions]::GetProperty($TaskAssignment, 'RetryCount', 0)
if ($retryCount -gt 0)
{
    $TaskAssignment.RetryCount = $retryCount - 1
    Update-RpsTaskAssignment -TaskAssignment $TaskAssignment -TaskState $Rps.TaskStates.Retry -StatusMessage
'Retry from Invoke-RpsTaskAssignment'
}
else
{
    throw "Error on $($computer.Name) in $runbookName. Error: $($TaskAssignment.StatusMessage)"
}
```

Require User Approval

Some Tasks may require manual user approval to continue or stop processing. This approval step requires setting the Task Assignment's state to **PendingUserAction** and setting a prompt for the operator. The operator then approves or denies the action using RPS Web or PowerShell. If approved, the Task's State will be set to **Completed**. If denied, the state is set to **Cancelled**.

Example: Update Task Assignment for User Approval

Setting the required approval using the New-RpsTaskAssignmentUserAction cmdlet will create a prompt for the user in RPS Web.

```
New-RpsTaskAssignmentUserAction -TaskAssignment $taskAssignment `
    -UserActionPrompt 'Approve Domain Join?' `
    -UserActionApproveLabel 'Approve' `
    -UserActionDenyLabel 'Deny'
```

Legacy Runbooks (Workflows) Guidance

Prior to RPS v2.1, all RPS Runbooks were authored as PowerShell workflows instead of pure PowerShell scripts. Some RPS Runbooks are still Workflows, and they require special guidance.

PSDisableSerializationPreference

PowerShell Workflows, and by inheritance SMA Runbooks, serialize objects without methods. This can limit the usability of some RPS objects' method calls. To use these methods, the PSDisableSerializationPreference needs to be set to true.

```
$PSDisableSerializationPreference = $true
```

Workflow Naming

As a matter of consistency and support for the DSC resource managing RPS, the Workflow name must match the script name.

Registering Runbooks in RPS

Runbooks in the `/Runbooks` folder are automatically imported into RPS during install, via the `/Setup/Configuration/Import-RpsTasks.ps1` script. Once a full RPS Node is operational and SMA is running, any new or modified scripts in the runbooks folder will be automatically registered and imported into RPS and SMA, using DSC as the folder monitor.

Manually publishing a Runbook in SMA

If you're authoring and testing a Runbook directly in SMA and don't want to wait for DSC to auto-register, you must manually import and publish the Runbook via SMA.

Example: Publish Runbook directly in SMA

```
$path = "$contentPath\Runbooks\$runbookName.ps1"
$wse = 'https://localhost:9090'
Import-SmaRunbook -Path $path -WebServiceEndpoint wse
Publish-SmaRunbook -Name $runbookName -WebServiceEndpoint $wse
```

More Resources

- Authoring RPS DSC Resources
- RPS Tasking Guide

How to Add a Node to an Existing RPS Environment

This guide will describe how to add a new unparented node to an existing RPS Environment. For the following steps, the example environment is as follows:

- APP.Master
    - APP.Region
        - APP-S.Region
        - APP-S2.Region 1. This node is added after the initial deployment and does not initially have a parent

1. Login to APP.Region and App-S2.Region as RPSAdmin

2. Run this command on App.Region to see the details of the App.Region node. These details will be used in the next step.

```
Get-RpsNode
```

3. Run this command on Site2 to create the Region node:

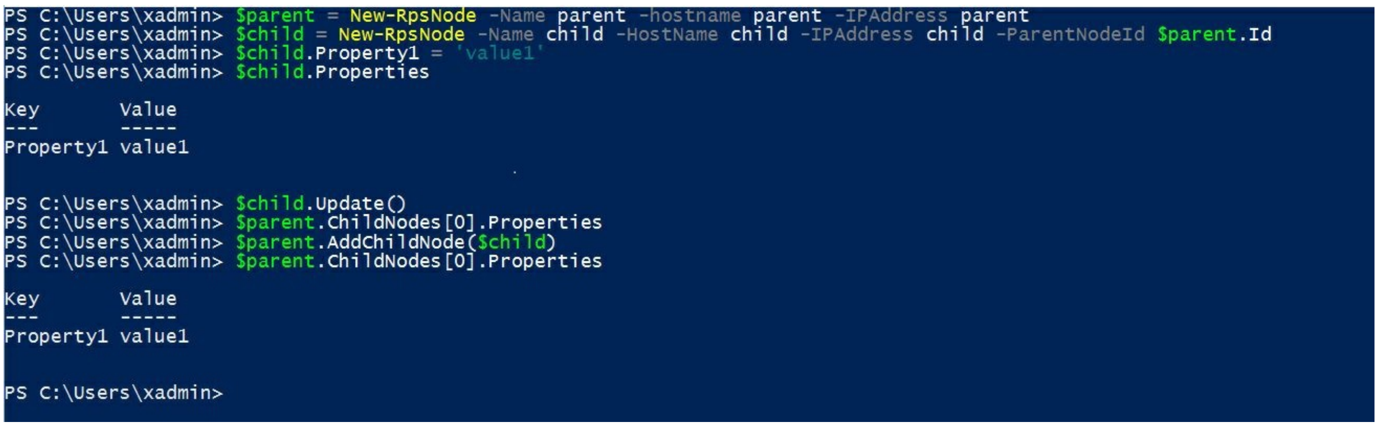
```
New-RpsNode -NodeId <id_of_app.region>  -Name Region -SyncEndpointUrl <sync_endpoint_url_of_app.region> -
CertificateThumbprint <certificate_thumbprint_of_app.region> -Hostname <hostname_of_app.region> -
IpAddress <ip_address_of_app.region>
```

4. Run these commands on Site2. The last command will show the details of the APP-S2 node. These details will be used in the next step.

```
$node = Get-RpsNode -Name Site2
$node.ParentNodeId = <id_of_app.region>
$node.Update()
$node
```

Known Issue: Making changes to child node prevents properties from being visible from ancestor nodes when in session
Current Workaround for displaying child node properties at parent node level: After making changes to any node that has a Parent Node use the following PowerShell command to fix the issue:

```
$parent.AddChildNode($child)
```



```
PS C:\Users\xadmin> $parent = New-RpsNode -Name parent -hostname parent -IPAddress parent
PS C:\Users\xadmin> $child = New-RpsNode -Name child -HostName child -IPAddress child -ParentNodeId $parent.Id
PS C:\Users\xadmin> $child.Property1 = 'value1'
PS C:\Users\xadmin> $child.Properties

Key       Value
---       -----
Property1 value1


PS C:\Users\xadmin> $child.Update()
PS C:\Users\xadmin> $parent.ChildNodes[0].Properties
PS C:\Users\xadmin> $parent.AddChildNode($child)
PS C:\Users\xadmin> $parent.ChildNodes[0].Properties

Key       Value
---       -----
Property1 value1


PS C:\Users\xadmin>
```

NOTE

This method works regardless of the ancestry depth. Re-adding a grandchild to its parent makes the properties visible when accessed from the parent and the grandparent.

5. Run the following command and restart RpsSync services to create the Site2 node:

```
New-RpsNode -NodeId <id_of_site2> -Name Site2 -SyncEndpointUrl <sync_endpoint_url_of_site2> -
CertificateThumbprint <certificate_thumbprint_of_site2> -Hostname APP-S2.region.rps -IpAddress
<ip_address_of_site2> -ParentNodeId <id_of_app.region>
```

   1. Restart the RpsSync service on both machines (APP.Region and APP-S2.region):
   2. Windows Key + R
   3. Services.msc
   4. Find RpsSync, right-click on it, and select Restart

6. On APP.region, after the sync service has restarted

   1. Windows Key + R
   2. iexplore https://app.region.rps:8080/
   3. Targeting > Nodes
   4. Find the Region node and click on it
   5. Make sure that the Site2 node is listed as a Child Node
   6. Dashboard > Sync
   7. Check for any sync errors to make sure it's synchronizing properly

7. On APP-S2.region, after the sync service has restarted

   1. Windows Key + R
   2. iexplore https://app-s2.region.rps:8080/
   3. Targeting > Nodes
   4. Find the Region node and click on it
   5. Make sure that the Site2 node is listed as a Child Node
   6. Dashboard > Sync
   7. Check for any sync errors to make sure it's synchronizing properly
   8. Resourcing > Resources
   9. Add Resource > Resource

8. Fill out the form with any information and click on the Save button

9. Restart the RpsSync service on both machines (APP.Region and APP-S2.region)

10. Check that your new Resource appears on both machines (you may need to give sync a few minutes). Do the following on each machine to check:

    1. Windows Key + R
    2. iexplore https://app.region.rps:8080/
    3. Resourcing > Resources
    4. Look for your new Resource

RPS Automation Package Guidelines

The RPS is designed to be a re-usable solution that offers similar, flexible functionality for programs, efforts, and automations requirements. It is a solution that was initially designed to survive the inherent constraints and challenges that have been observed within a mobile tactical network. The goal of this design is to create a stable, secure, and highly automated management infrastructure solution to provide the following (but not limited to) capabilities:

How to Load an Automation Package

An Automation Package is typically loaded using a Load Script. The load script is flexible, but is coded to simply place the components for a given automation package in the proper locations. This script is typically only executed one time to initially load a set of functionalities, scripts, metadata, or other configuration information into the RPS system. This script is run once during the initial load of the Automation Package. This can occur at initial system build, or after the system is already established, and should contain all logic necessary to add itself to the RPS system.

Some common actions a Load Script may take will include:

- Inserting MetaData into the RPS CMDB
- Placing Powershell Scripts and Modules into the appropriate locations defined for RPS, this will allow the system to automatically register and load them for use (See Section 2.5 for info on placement)
- Placing external software components used by those Scripts and Modules into the necessary locations as coded by the authors (E.g. VMWare tools being installed)
- Inserting TaskItem and TaskMap metadata into the database. These are the definitions of what should be run, targeting what devices, and in what order.

Once the load script is executed, it is typically removed and no longer required. The initial insert of data serves as a way to inject new business logic into the system. From there, the system will function based on the inserted POR authored Automation Package.

Post-Load Actions and Activity

Once a package is loaded, business requirements begin to dictate what will happen next. For example, if the Load Script is coded to activate automation work on load, then RPS actions will begin immediately processing.

If the Load Script is not coded to activate automation work on load, the following RPS components can be used to control and trigger automation execution

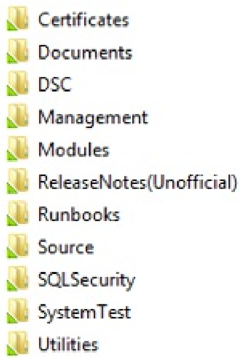
- IsActive settings
- PendingUserActions
- TaskAssignment creation
- TaskMapAssignment creation

Using these core functionalities, automation can be triggered, controlled, and centrally managed on an as-needed basis. Post-Load activities are controlled by the needs of the business, and should be coded to support the intended execution model.

Format of an Automation Package

The format of an automation package should follow the folder structure of the build output that RPS produces, and either be included with the initial set of installations of an RPS software stack, or placed into the same folders once RPS is already installed.

For example, using the figure below showing a sample build output from RPS, there are clearly defined locations for various components of an Automation Package. There are folders and descriptions for the intended location of specific components already defined. The format of the Automation package will vary depending on business use and intended uptake of the content.


Certificates
Documents
DSC
Management
Modules
ReleaseNotes(Unofficial)
Runbooks
Source
SQLSecurity
SystemTest
Utilities

Build Output and Automated Support

Some folders supplied as part of the RPS Build output will support automated control and placement of inserted code product. This section will explain which folders are supported and by what action or activity:

TERM	DEFINITION
Certificates	This is a general store for storage of certificates required by the system. Some are controlled automatically, but this is done per certificate currently.
DSC	This folder is used to hold both DSC configurations and Resources. The RPS support of DSC processes are coded to look to these locations for supporting DSC resources and configuration files.
Modules	Any PowerShell modules located in this directory will be automatically placed into the appropriate PowerShell Modules directory on the server RPS is installed on. Once this directory exists on a server with RPS installed, any newly added modules will be copied on next pass of the DSC configuration (typically 30 minutes).
Runbooks	Any runbooks in this directory will be automatically loaded and registered into SMA. Once this directory exists on a server with RPS installed, any newly added Runbooks will be registered to SMA on next pass of the DSC configuration (typically 30 minutes).

More Resources

- RPS Customization Guide - High Level Overview
- RPS IPSheet Parser

Certificate Usage

Introduction

The future security needs of the RPS security infrastructure are currently planned to depend on a Public Key Infrastructure (PKI). However, the current landscape of development for the project does not allow for the full implementation of PKI. In its absence, a Self-Signed Certificate strategy has been developed as a temporary measure to provide improved security over plain text secrets and ease the future adoption of full PKI.

The RPS Solution uses certificates for a variety of functions, including:

1. Web Site SSL binding for HTTPS encrypted transport between server (e.g. RPS Website) and client.
2. RPS Sync Service for client\server authentication between **subscriber** (e.g. RPS Sync Service on Region) and **distributor** (e.g. RPS Sync Service on Master) nodes. The certificate thumbprints for all trusted nodes are whitelisted in the RPS CMDB.
3. Rps Sync Service for HTTPS encrypted transport between server and client.
4. DSC mof file credentials encryption (By default DSC encrypts the entire mof file).
5. WinRM for HTTPS encrypted transport between server and client.
6. SQL for HTTPS encrypted transport between server and client.
7. Provisioning SSL binding for HTTPS encrypted transport between server (e.g. RPS Provisioning) and client.

IMPORTANT

Each certificate must have a certificate root that is trusted by the server (i.e. Trusted Root Certification Authorities).

By default, RPS includes a variety of certificates (even self-signed) to showcase functionality, and it is expected that these development or test certificates will be replaced with appropriate secure and trusted certificates to perform the various functions using the roles indicated.

WARNING

Use of self-signed or untrustworthy certificates presents a security risk for all assets and functions "secured" by said certificates.

The table below maps certificates in the ContentStore, as well as certificates generated by the deployment, to their role and corresponding function.

NOTE

The .pfx file is capable of storing both public and private keys whereas the .cer file is generated from the .pfx and contains only the public key.

Certificate roles and functions

CERTIFICATE ROLE	KEY USAGES	ENHANCED KEY USAGES	SIGNATURE ALGORITHM	PROVIDER NAME	KEY LENGTH	SCOPE	OTHER NOTES
RpsRoot	Certificate Signing, Off-line CRL Signing, CRL Signing (06)		SHA512	SKSP	4096	All Computers	At root of the cert chain
RpsSync		Client Authentication	SHA256	SKSP	2048	RpsSync hosts	Cert:\CurrentUser\My for Sync account

CERTIFICATE ROLE	KEY USAGES	ENHANCED KEY USAGES	SIGNATURE ALGORITHM	PROVIDER NAME	KEY LENGTH	SCOPE	OTHER NOTES
RpsSyncSSL	KeyEncipherment, DigitalSignature, NonRepudiation		SHA256	SKSP	2048	RpsSync hosts	RpsSync HTTPS endpoint
RpsGuiSSL	Digital Signature, Non-Repudiation, Key Encipherment (e0)		SHA256	SKSP	2048	RpsGui hosts	Rps Gui HTTPS binding
WinRm		Server Authentication	SHA256	ECP v1.0	2048	All computers	PowerShell HTTPS endpoint
DscEncryption	Key Encipherment, Data Encipherment (30)		SHA256	ECP v1.0	2048	All computers	Credential encryption in DSC .mof files
SqlSSL		Server Authentication	SHA256	ECP v1.0	2048	SQL hosts	SQL HTTPS endpoint
iPxeSSL	Digital Signature, Non-Repudiation, Key Encipherment (e0)		SHA256	SKSP	2048	iPxe hosts	iPxe HTTPS endpoint
ProvisioningSSL	KeyEncipherment, DataEncipherment	Server Authentication	SHA256	ECP v1.0	2048	Provisioning hosts	Rps Provisioning endpoint

NOTE

Key for table above:

SKSP = Microsoft Software Key Storage Provider

ECP v1.0 = Microsoft Enhanced Cryptographic Provider v1.0

Generating certificates

Certificates can be generated as part of the installer process or supplied from an external PKI. By default, the New-RpsNodeConfiguration.ps1 script will generate self-signed certificates for each role and server using the existing configuration data. If external certificates will be used, the certificate data file located at '{ContentRoot}\Setup\Configuration\RpsCertificateData.psd1' will need to be updated to store the certificate role and password information. The certificates themselves must also be stored in the following path: '{ContentRoot}\Certificates'. The naming convention required for each certificate file should be as follows: '{TargetItemName}_{CertificateRole}.pfx/cer'.

Certificate functions

As part of the Rps-Encryption module, the `Set-RpsCertificate` function allows you to create role driven certificates for use by RPS. This function will create a new certificate as well as import it into an existing Rps Session. For example:

```
Set-RpsCertificate `
    -Role RpsSync `
    -Path C:\ContentStore\Certificates\RpsSync.pfx `
    -Password $password `
    -SubjectName "CN=RpsSync" `
    -SigningCertificatePath C:\ContentStore\Certificates\RpsRoot.pfx `
    -SigningCertificatePassword $password
```

Also part of the Rps-Encryption module, the `New-RpsCertificate` function allows you to create template driven certificates. The function will generate certificates but do not import the certificate into an existing Rps session. For example:

```
New-RpsCertificate `
    -CertificateType SSL `
    -Password $password `
    -SubjectName "CN=RpsSync" `
    -SigningCertificatePath C:\ContentStore\Certificates\RpsRoot.pfx `
    -SigningCertificatePassword $password
```

As part of the Rps-Installer module, the `Import-RpsCertificate` function allows you to import an existing certificate into the Rps CMDB. For example:

```
Import-RpsCertificate `
    -Name AD.master.rps_DscEncryption `
    -Path C:\ContentStore\Certificates\AD.master.rps_DscEncryption.pfx `
    -Password $password `
    -Role DscEncryption
    -AssignTo (Find-RpsTargetItem -Name AD.master.rps -Type Computer)
```

The `New-RpsCertificate` function implements the `New-RpsSelfSignedCertificate` function in the Rps-Encryption Module. The `New-RpsSelfSignedCertificate` function is generic and allows the configuration of many different certificate settings.

SQL Encryption

SQL Transparent Data Encryption (TDE) encrypts data and log files to keep database data protected “at rest.” SQL TDE uses a symmetric database encryption key (DEK) stored in the master database to encrypt/decrypt data in real-time. The DEK is secured using a certificate managed by SQL. The database can only be recovered or moved to another instance or server with the exported certificate.

RPS Database Encryption

RPS is configured to use SQL TDE for both the SMA database and the RPS CMDB using DSC and T-SQL. The certificate used to secure the DEK is generated automatically with DSC, is called RpsDatabaseCertificate.crt, and is backed up to disk (by default in `C:\Backups\Certificates`). The server’s master key is backed up to RpsDatabaseMasterKey.crt using the password supplied for the RPS Configuration.

WARNING

The compromise of the TDE certificates could allow malicious users to retrieve unencrypted data. Follow proven certificate management and backup practices to mitigate security vulnerabilities while preserving the ability for a legitimate administrator to restore the RPS CMDB or SMA databases if needed.

RPS Customization Guide - High Level Overview

RPS provides the ability for the admin to customize some aspects of the web based GUI. This functionality allows the admin to have the application better fit within existing branding and theming guidelines that maybe already be established.

Footer

The footer for the web user interface is off by default. Within the Web.Config file you can enable it so that it will be shown on all pages within the web application.

For configuring the footer, you will need to add a section declaration and then create the config section called `footerConfig`.

Section declaration

Within the Web.Config you will want to add the following section declaration to the configuration > configSections element.

```
<section name="footerConfig" type="Rps.Web.ConfigSections.FooterSection"/>
```

Once it is added, your configSections element should look like this:

```
  <configSections>
    <section name="footerConfig" type="Rps.Web.ConfigSections.FooterSection"/>
  </configSections>
```

footerConfig Element

With the `footerConfig` section has been declared in the configSections element, you can add the following `footerConfig` element after the `appSettings` element:

```
  <footerConfig show="false">
    <links></links>
  </footerConfig>
```

Figure 2 FooterConfig within the Web.Config

To enable the footer to be displayed, set the show attribute to true.

Adding links within the footerConfig

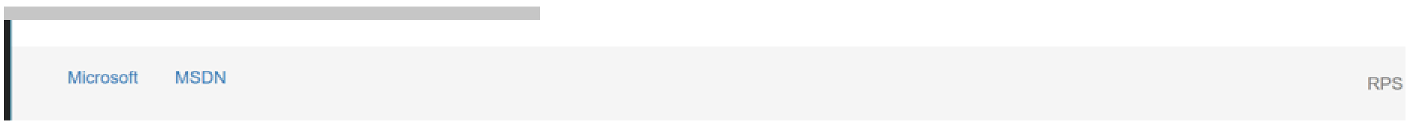
Within the `footerConfig` element there is a child element called `links`. This will take a variable number of `link` elements to be displayed on the footer.

A `link` element has two attributes, the text attribute and URL attribute. The text attribute controls the text that will be displayed to the user. The URL attribute is the actual location that user will be sent to when they click the displayed link.

For example, adding a link to the Microsoft & MSDN web sites, the configuration look like so:

```
      <add text="Microsoft" url="https://microsoft.com" />
      <add text="MSDN" url="https://msdn.microsoft.com" />
```

Resulting in a view to the user as such:



NOTE

The ASP.Net runtime monitors the Web.Config file to detect any changes that are made. This will cause the web application to restart and then load the changes. This should only take a matter of seconds for the application to restart.

Branding Customizations

RPS allows the header of the web application to be customized with specific branding so the application can look like existing properties within the organization.

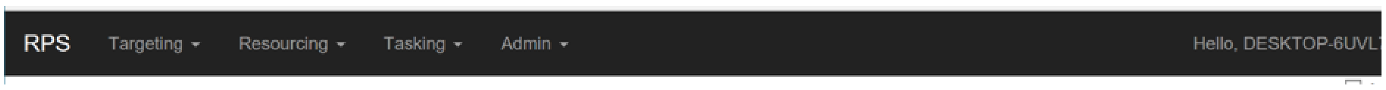


Figure 3 The RPS Header that can be configured.

Icon

The RPS Header can be configured to show an icon to show on the left side of the header. When using the setting, the value should represent a path that reflects the location of the image from within the application.

For example, if the image is stored at C:\WebApps\RPS\Images\brand.png and the root of the web application points to C:\WebApps\RPS then the path would be /Images/brand.png just like it would be defined within an HTML document.

We recommend an image of 50px by 50px for best results.

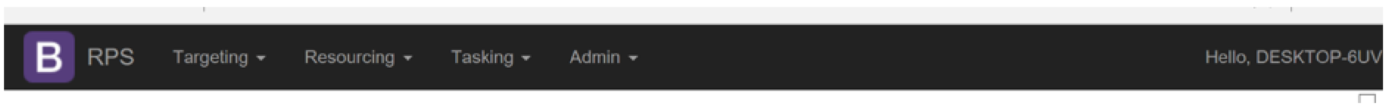
Setting Name

```
ui:BrandImage
```

Example

```
<add key="ui:BrandImage" value="/Images/brand.png" />
```

Will update the header to look like this:



Default:

No Image will be displayed

Brand Text

The brand text configuration option allows the name of the web application to be changed within the header.

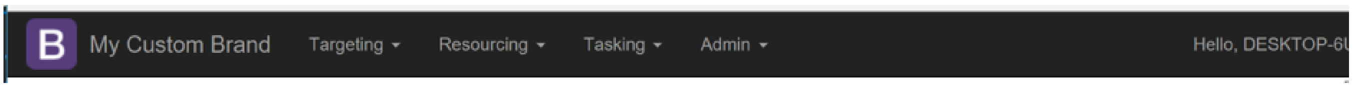
Setting Name

```
ui:Brand
```

Example

```
    <add key="ui:Brand" value="My Custom Brand" />
```

Will update the header to look like this:



Default:

The value "RPS" will be displayed.

Greeting Text

The greeting text configuration allows the administrator to update the text displaying before the logged in user's name. This can be set to anything or nothing if so configured.

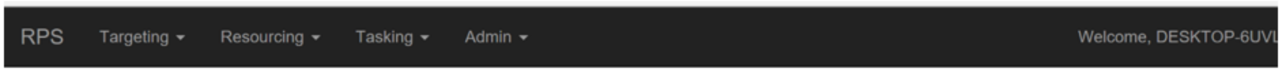
Setting Name

```
ui:Greeting
```

Example

```
<add key="ui:Greeting" value="Welcome, " />
```

Will update the header to look like this:



Default:

The value "Hello, " will be displayed before the user's name.

Theming

Theming within RPS allows the administrator to update some color and font aspects of the web application to best match the existing look and feel that is already established. A theme consists of background color, font color, font type and font size. There are two main areas for theming, the header and the content area. Each of the areas is configurable within the applications Web.Config.

Header

Header theming allows customization of background color, font and font-size for top most portion of the web application.

Background Color

Setting

```
ui:HeaderBackgroundColor
```

Example

```
<add key="ui:HeaderBackgroundColor" value="#FF00FF" />
```

This will turn the header background color to Fuchsia.

Default

The default color with be the HTML Hex color code #222222

Font

Setting

```
ui:HeaderFont
```

Example

```
<add key="ui:HeaderFont" value="Segoe UI" />
```

This will turn all the header fonts to Segoe UI.

Default

The default font is defined as `"Helvetica Neue", Helvetica, Arial, sans-serif`.

Font Color

Setting

```
ui:HeaderFontColor
```

Example

```
<add key="ui:HeaderFontColor" value="#F0F8FF" />
```

This will turn the header font color to Alice Blue.

Note: This will only change the text color for the header and the top navigation item. The dropdown list and text will not be affected.

Default

The default color with be the HTML Hex color code #9D9D9D

Font Size

This setting will take any valid CSS defined size. This includes absolute size values, relative size values, pixel and percentage sizes. The values can be found here. This setting is applied to all the fonts within the header.

Setting

```
ui:HeaderFontSize
```

Example

```
<add key="ui:HeaderFontSize" value="x-large" />
```

This will turn the header font size to extra-large.

Default

The default font size for the header text varies but ranges from 10px to 14px.

Footer

Footer theming allows customization of background color, font and font-size for top most portion of the web application.

Background Color

Setting

```
ui:FooterBackgroundColor
```

Example

```
<add key="ui:FooterBackgroundColor" value="#FF00FF" />
```

This will turn the footer background color to Fuchsia.

Default

The default color with be the HTML Hex color code #222222

Font

Setting

```
ui:FooterFont
```

Example

```
<add key="ui:FooterFont" value="Segoe UI" />
```

This will turn all the footer fonts to Segoe UI.

Default

The default font is defined as `"Helvetica Neue", Helvetica, Arial, sans-serif`.

Font Color

Setting

```
ui:FooterFontColor
```

Example

```
<add key="ui:FooterFontColor" value="#F0F8FF" />
```

This will turn the footer font color to Alice Blue.

Note: This will only change the text color for the footer and the top navigation item. The dropdown list and text will not be affected.

Default

The default color with be the HTML Hex color code #9D9D9D

Font Size

This setting will take any valid CSS defined size. This includes absolute size values, relative size values, pixel and percentage sizes. The values can be found here. This setting is applied to all the fonts within the footer.

Setting

```
ui:FooterFontSize
```

Example

```
<add key="ui:FooterFontSize" value="x-large" />
```

This will turn the footer font size to extra-large.

Default

The default font size for the footer text varies but ranges from 10px to 14px.

More Resources

- RPS Automation Package Guidelines
- RPS IPSheet Parser

RPS IPSheet Parser

The RPS IPSheet Excel file is a Microsoft Excel spreadsheet that contains CMDB endpoint data for a customer. The IPSheet will contain a set of worksheets for NIPR, SIPR, and Colorless. Each worksheet will contain Endpoint data for Subnets and IPAddresses needed for the CMDB. These Subnets and IPAddresses will be stored as Resource Groups and Items in the RPS database.

A sample IPSheet Excel file can be found at:

```
RPS > Source > Rps.Extension.IpSheet > Samples > TestIpSheet.xlsx
```

Assumptions

- The 1st column in the worksheet is ignored.
- Subnet/IPAddress are used as the name of the imported Subnet or IpAddress and are expected to be unique.
- Column headers are exactly, "Telephony Rng", "Subnet", "Mask", "Gateway", "Reserved", "Assignment"
- Subnet name is expected to be named "SIPR", "NIPR" or "Colorless", exactly.
- Reserved IPAddresses are expected to be listed in the same order as the Subnets are.
- Each set of reserved IpAddresses for a Subnet will list the subnet name in the first row ONLY.
- **Note:** This is important because subnet names aren't unique ("Tactical Spare") so we use a combination of sequence and presence of the name to identify a new group, rather than IP Logic.
- Row 1 contains column headers.
- Row 2 contains header information.
- Rows 3 -> x contain Subnets.
- Row x+1 contains value "Unit Base Info" in the 2nd column.
- Row y contains value "Detailed Info" in the 2nd column.
- Row y+1 -> z-1 contains reserved IPAddresses.
- Row z contains value "End" in the 2nd column

RPS IPSheet Parser .NET Library (DLL)

The RPS IPSheet Parser is a .NET library (DLL) written in C#. This library contains two key Powershell Cmdlets for importing or removing IPSheet data to/from the RPS CMDB database.

The PS Module defined by the RPS IPSheet Parser library assembly is imported in Powershell through the module's manifest (psd1). Subsequently, the Cmdlets can be called to import and parse IPSheet data into RPS.

Example:

```
Import-Module Rps-IpSheet
```

Import-RpsIpSheet Cmdlet

The Import-RpsIpSheet Powershell Cmdlet is used to import IPSheet data into the RPS database. This cmdlet accepts one parameter "Path" that accepts an absolute or relative path to the IPSheet Excel file location. Results will show the total number of Subnets and IpAddresses imported. Warnings will show any import errors, including invalid file paths.

NOTE

The import will not overwrite or merge with existing data, so if there is a collision on subnet or ipaddress, then a warning is displayed.

Example:

```
Import-RpsIpSheet -Path c:\ipsheet\abc-tactical-2018.xls

Import-RpsIpSheet -Path .\abc-tactical-2018.xls
```

Remove-RpsIpSheetCmdlet

The Remove-RpsIpSheet Powershell Cmdlet is used to remove previously imported IPSheet data from the RPS database. This cmdlet has no parameters. It will remove all previous imports of IPSheet data. Results will display all resource groups (subnets) and resource items (ip addresses) removed from the RPS database.

Example:

```
Remove-RpsIpSheet
```

Powershell Script

The IPSheet data can be imported into an RPS database using Powershell. The Powershell script will import PS Module from the RPS.Extension.IpSheet .NET assembly. The PS Module is distributed to the Content Store located at ContentStore\Modules\Rps-IpSheet. Then the Import-RpsIpSheet cmdlet is called with an absolute or relative path to the location of the IpSheet Excel file that contains the Subnet and IpAddress endpoint data for the CMDB.

Example:

```
Import-Module Rps-IpSheet
$start = Get-Date
Import-RpsIpSheet -Path C:\ipsheet\86ESB-A-Tactical-Template-11-17-2015.xls -Verbose
(Get-Date).Subtract($start).TotalSeconds
```

More Resources

- RPS Automation Package Guidelines
- RPS Customization Guide - High Level Overview

Instance Definition

Introduction

RPS Instance Definition are an abstraction layer on top of existing RPS Types. Types currently allow us to define a specific Type of Resource or Target Item, its expected or possible Properties and their associated value types (string/int/etc.). Instance Definition will allow us to take these generic Types and combine them together to create complex, nested, entities composed of many different Types through Parent/Child�and assignment relationships, and the known default values for each individual Types' properties.

Examples: SNEs, TCNs, TSI-Large, laptop, could be a VM (because it could be a collection of components)

Why RPS Instance Definition?

We have defined many existing Types such as Vehicle, Computer, VirtualMachine, NetworkConfiguration, and so on. The entities we are now interacting with though are complex and composed of many of these Types in Parent/Child hierarchies. Instance Definitions are collections of Type Definitions, that have the ability to create an object. Type Definitions are parts with detailed properties. They do not create objects, but they are able to define something you can create which is what an object will consist of.

Example: Virtual Machine, NIC, drive, etc.

NOTE

The word Instance Definition is formerly known as 'Templates'.

Creating an Instance Definition

Creates a new Instance Definition.

```
New-RpsInstanceDefinition -Name testName
```

Creates a new Instance Definition with Properties.

```
$hs = @{
Prop1 = "value1"
Prop2 = "value2"
}
New-RpsInstanceDefinition -Name testName -Properties $hs
```

Creates a new Instance Definition with Properties and a parent Node

```
$nodeId = "81B8272D-B49C-4350-A8F4-ABBB9CE29C68"
$hs = @{
Prop1 = "value1"
Prop2 = "value2"
}
New-RpsInstanceDefinition -Name testName -Properties $hs  -ParentNodeId $nodeId
```

Creates a new Instance Definition with Virtual Machine and Network Configuration.

```
$vmTypeDef = Get-RpsResourceItem -Name VirtualMachine -Type RpsTargetType
$nicTypeDef = Get-RpsResourceItem -Name NetworkConfiguration -Type RpsTargetType
$credentialTypeDef = Get-RpsResourceItem -Name Credential -Type RpsResourceType

$vmAdServer = New-RpsInstanceDefinitionItem -EntityName "Ad.[^DomainName]" -Name AdServer -TypeDefinition
$vmTypeDef -Properties @{
� � JoinDomain = $false
� � ComputerName = 'AD'
� � DnsZone = '[^DomainName]'
� � OSType = 'Windows'
� � OSVersion = '8.1'
� � MemoryMB = 1024
� � IsDC = $true
}
$vmAppServer = New-RpsInstanceDefinitionItem -EntityName "App.[^DomainName]" -Name AppServer -TypeDefinition
$vmTypeDef -Properties @{
� � JoinDomain = $true
� � ComputerName = 'APP'
� � DnsZone = '[^DomainName]'
� � OSType = 'Windows'
� � OSVersion = '8.1'
� � MemoryMB = 2048
� � IsCDN = $true
� � IsDB = $true
� � IsSMA = $true
}
$nicVlan996 = New-RpsInstanceDefinitionItem -EntityName "[^ParentName]-Vlan996" -Name Nic-Vlan996 -
TypeDefinition $nicTypeDef -Properties @{
� � Primary = $true
� � VlanId = 996
� � NetworkCategory = 'DomainAuthenticated'
}
$credentialRpsAdmin = New-RpsInstanceDefinitionItem -EntityName "[^DomainPrefix]_RpsAdmin" -Name
Credential_RpsAdmin -TypeDefinition $credentialTypeDef -Properties @{
� � UserName = '[^DomainPrefix]\RpsAdmin'
� � Role = 'ServerAdmin'
� � IsLocal = $false
� � CreateAccount = $true
� � PasswordNeverExpires = $false
}
$instanceDefinition = New-RpsInstanceDefinition -Name Vehicle -Properties @{DomainName = 'RequiredValue';
DomainPrefix = 'RequiredValue'}
New-RpsInstanceDefinitionReference -Name Vehicle_AppServer -InstanceDefinition $instanceDefinition -
InstanceDefinitionItem $vmAppServer
New-RpsInstanceDefinitionReference -Name Vehicle_AppServer -InstanceDefinition $instanceDefinition -
InstanceDefinitionItem $vmAdServer
New-RpsInstanceDefinitionReference -Name Vehicle_AppServer -InstanceDefinition $instanceDefinition -
InstanceDefinitionItem $nicVlan996 -ParentItem $vmAppServer
New-RpsInstanceDefinitionReference -Name Vehicle_AppServer -InstanceDefinition $instanceDefinition -
InstanceDefinitionItem $nicVlan996 -ParentItem $vmAdServer
New-RpsInstanceDefinitionReference -Name Vehicle_AppServer -InstanceDefinition $instanceDefinition -
InstanceDefinitionItem $credentialRpsAdmin
New-RpsInstanceDefinitionReference -Name Vehicle_AppServer -InstanceDefinition $instanceDefinition -
InstanceDefinitionItem $credentialRpsAdmin -ParentItem $vmAppServer
New-RpsInstanceDefinitionReference -Name Vehicle_AppServer -InstanceDefinition $instanceDefinition -
InstanceDefinitionItem $credentialRpsAdmin -ParentItem $vmAdServer
$settingsResourceItem = New-RpsResourceItem -Type Settings -Name UnitASettings -Properties @{
� � DomainName = 'Master.Rps'
� � DomainPrefix = 'Master'
}
Invoke-RpsInstanceDefinition -InstanceDefinition $instanceDefinition -settings $settingsResourceItem
```

Getting an Instance Definition

Gets an instance definition by Id.

```
Get-RpsInstanceDefinition -Id $lookupId
```

Setting an Instance Definition

Creates or Updates instance definition.

```
$instanceDef = Set-RpsInstanceDefinition -Name "MyInstanceDef" -Properties @{Prop1 = "Value1"} -ParentNodeId
$nodeId
```

Invoking an Instance Definition

Creates instances based on the Instance Definition using a resource item to hold its settings.

```
$settings = Get-RpsResourceItem -Id $guid
Invoke-RpsInstanceDefinition -settings $settings
```

Glossary

TERM	DEFINITION
Instance Definition	The abstraction layer on top of existing RPS Types.
Instance Definition Reference	The assignment of the instance definition to a root Resource Item, which results in a set of one or more Resource Items to be run.
Instance Definition Item	The an item on the abstraction layer on top of existing RPS Types.
Instance Definition Node	A node item that will result in association of a node with target items that are created when an Instance Definition is invoked

More Resources

- Instance Definition Item
- Instance Definition Node
- Instance Definition Reference

Certificate Requirements for Linux Clients

Table of Contents

Introduction

This document describes the certificate requirements to leverage Patch Management in Rapid Provisioning System (RPS) for Linux clients. Additionally, this document also provides instructions on installing the certificate required.

Document Overview

Patch Management in RPS requires communication via HTTPS. The certificate authority (CA) that signed the webserver's certificate must be trusted by the Linux client or patches will not be downloaded. This is done by installing the public certificate of the CA. This document is considered a living document and subject to change.

Installing the RPS CA Public Certificate

1. Copy the RPS CA public certificate to Linux machine.

   a. The RPS CA public certificate is located in `\ContentStore\Certificates\RpsRoot.cer`

2. Convert to .pem file with openssl tool.

   a. `openssl x509 -inform der -in certificate.cer -out certificate.pem`

   b. If you receive a 0D0680A8 and 0D07803A error, the certificate is already in the correct format. The only change needed is to change the certificate's file extension from .cer to .pem

3. Rename RpsRoot.cer to RpsRoot.pem

   a. `mv RpsRoot.cer RpsRoot.pem`

4. Once the certificate has the .pem extension copy certificate to:

   a. `/etc/pki/ca-trust/source/anchors/`

5. Import the certificate with the following command:

   a. `update-ca-trust extract`

   b. The certificate will be added to the `/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem` file.

6. Verify the certificate imported with the following command:

   a. `cat /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem | grep RPS`

How to Configure Logging for the RPS API

This guide will provide links for how to configure logging for the RPS API. RPS uses Serilog for logging and log file management. RPS reads an app settings section of the app.config file within RPS to configure Serilog. The app.config file for the RPS API is located in the Source/Rps.Api folder.

- The Serilog website is located at https://serilog.net/
- The Serilog documentation for app settings can be found at https://github.com/serilog/serilog-settings-appsettings
- RPS will honor all app settings that Serilog supports by passing the app settings section to Serilog upon construction of the logger.