ivanti | User Workspace Manager

Server Configuration Portal Scripting Guide

Version 2026.2

Contents

Introduction

This guide assists Ivanti administrators to automate the configuration of Ivanti product servers, such as Management Server and Personalization Server, using Windows PowerShell.

Ivanti product servers are shipped with Server Configuration Portal (SCP) cmdlets – a set of Windows PowerShell cmdlets that allow administrators to easily automate tasks such as:

- Initial configuration of a server
- Creating databases or exporting database scripts
- Starting or stopping a server
- Checking the operation of a server
- Enabling or disable logging

Note

When following examples in this document, enter only the part after the Windows PowerShell prompt. For example, if the example shows:

```
PS C:\> $credential = Get-Credential
```

Enter the following in the Windows PowerShell console:

```
$credential = Get-Credential
```

Limitations

The following limitations apply to the SCP cmdlets:

A new PowerShell session is required for the configuration of each Ivanti product server. Once an Import-ApsInstanceModule operation has been invoked the PowerShell session can only be used to configure that specific server instance.

Prerequisites

Installing Windows PowerShell

SCP cmdlets require Windows PowerShell 3.0 to be installed before use. To install Windows PowerShell:

- Windows Server 2012 or 2012 R2 – Windows PowerShell is installed as standard on Windows Server 2012.
- Windows Server 2008 SP2 or 2008 R2 SP1 – Download Windows Management Framework 3.0 from the Microsoft Download Center:

  http://www.microsoft.com/en-gb/download/details.aspx?id=34595

> **Note**
>
> For more information refer to *Installing Windows PowerShell* on Microsoft MSDN:
>
> https://msdn.microsoft.com/en-us/powershell/scripting/setup/installing-windows-powershell

Starting Windows PowerShell

Use either of the following methods to launch a Windows PowerShell console:

- Click **Start**, type PowerShell. Right-click Windows PowerShell and select **Run as administrator**.
- On the taskbar, right-click Windows PowerShell and select **Run as administrator**.

> **Note**
>
> For more information refer to *Starting Windows PowerShell on Windows Server* on Microsoft MSDN:
>
> https://msdn.microsoft.com/powershell/scripting/setup/starting-windows-powershell

Checking the Version of Windows PowerShell

The $PSVersionTable variable should be used to confirm that either PSVersion is 3.0 or later, or PSCompatibleVersions contains 3.0.

```
PS C:\> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      3.0
WSManStackVersion              3.0
SerializationVersion           1.1.0.1
CLRVersion                     4.0.30319.18047
BuildVersion                   6.2.9200.16481
PSCompatibleVersions           {1.0, 2.0, 3.0}
PSRemotingProtocolVersion      2.2
```

Getting Help

Help Available in Windows PowerShell

Help for all SCP cmdlets are available in Windows PowerShell using the Get-Help cmdlet:

```
PS C:\> Get-Help Get-ApsInstance

NAME
    Get-ApsInstance

SYNOPSIS
    Gets the product instances that are installed.
...
```

Examples and further details can be obtained using the -full parameter:

```
PS C:\> Get-Help Get-ApsInstance -Full
```

Listing User Workspace Manager Modules and Cmdlets

To list modules:

```
PS C:\> Get-Module -Name AppSense* | Format-List

Name              : AppSense.ServerConfiguration.PowerShell
Path              : C:\Program Files\AppSense\Environment Manager\Personalizatio
                    n Server\bin\AppSense.ServerConfiguration.PowerShell.dll
Description       :
ModuleType        : Binary
Version           : 13.0.0.0
NestedModules     : {}
ExportedFunctions :
ExportedCmdlets   : {Get-ApsDatabaseVersion, Disable-ApsLogging,
                    Export-ApsDatabaseScript, Export-ApsServerImage...}

...
```

To list cmdlets:

```
PS C:\> Get-Command -Module AppSense*

CommandType     Name                                               ModuleName
-----------     ----                                               ----------
Cmdlet          Disable-ApsLogging                                 AppSense....
Cmdlet          Enable-ApsLogging                                  AppSense....
Cmdlet          Export-ApsDatabaseScript                           AppSense....
Cmdlet          Export-ApsServerImage                              AppSense....
Cmdlet          Get-ApsCurrentInstance                             AppSense....
Cmdlet          Get-ApsDatabaseDetail                              AppSense....
Cmdlet          Get-ApsDatabaseScript                              AppSense....
Cmdlet          Get-ApsDatabaseVersion                             AppSense....
Cmdlet          Get-ApsInstance                                    AppSenseI...
Cmdlet          Get-ApsPrerequisite                                AppSense....
Cmdlet          Get-ApsServerDetail                                AppSense....
Cmdlet          Get-ApsVariance                                    AppSense....
Cmdlet          Import-ApsInstanceModule                           AppSenseI...
Cmdlet          Initialize-ApsDatabase                             AppSense....
Cmdlet          Initialize-ApsServer                               AppSense....
Cmdlet          Install-ApsPrerequisite                            AppSense....
Cmdlet          Repair-ApsVariance                                 AppSense....
Cmdlet          Reset-ApsServer                                    AppSense....
Cmdlet          Set-ApsServerDatabase                              AppSense....
Cmdlet          Set-ApsWebSite                                     AppSense....
Cmdlet          Start-ApsServer                                    AppSense....
Cmdlet          Stop-ApsServer                                     AppSense.... Cmdlet
Update-ApsServer                                  AppSense....
```

Further Resources

For more information about Windows PowerShell refer to the following resources on Microsoft MSDN and Microsoft TechNet:

Getting Started with Windows PowerShell

https://msdn.microsoft.com/powershell/scripting/getting-started/getting-started-with-windows-powershell

Windows PowerShell Core About Topics

http://technet.microsoft.com/en-us/library/hh847856.aspx

Product Instances

Multiple Ivanti product servers can be installed simultaneously on a machine. For example, a machine can run up to 17 instances of the Ivanti Management Server and up to 17 instances of the Ivanti Personalization Server. Each different product server is referred to as an instance.

To allow administrators to manage different instances on the same machine a set of cmdlets known as AppSenseInstances is installed by Ivanti product servers. Use AppSenseInstances to:

- List installed product instances.
- Load the appropriate set of SCP cmdlets for a product instance.

Windows PowerShell automatically loads the AppSenseInstances cmdlets for you when launched – you don't need to load these cmdlets manually.

Note

You can only load the SCP cmdlets for one product instance into each Windows PowerShell session. If you need to work with multiple instances in the same script refer to Scripting Multiple/Remote Servers.

Listing available Product Instances

Use the Get-ApsInstance cmdlet to list available product instances. For example:

```
PS C:\> Get-ApsInstance

Name                : DEFAULT
InstanceId          : 935bfcca-23d0-4a08-93e4-064025b91d7b
ProductCode         : 05aa1fcd-7e88-408d-992b-4c96926bdd22
OriginalUpgradeCode : 935bfcca-23d0-4a08-93e4-064025b91d7b
OriginalProductCode : 05aa1fcd-7e88-408d-992b-4c96926bdd22
Version             : 10.0.604.0
InstallPath         : C:\Program Files\AppSense\Management Center\Server\Bin\
CmdletPath          : C:\Program Files\AppSense\Server Configuration
                      Portal\Bin\AppSense.ServerConfiguration.PowerShell.dll;
                      C:\Program Files\AppSense\Management Center\Server\Bin\
                      AppSense.ManagementServer.PowerShell\AppSense.Management
                      Server.PowerShell.dll;C:\Program Files\AppSense\Management
                      Center\Server\Bin\AppSense.ManagementServer.PowerShell\
                      AppSense.ManagementServer.PowerShell.psd1
ProductName         : AppSense Management Server 10.0
IsDefault           : True
Status              : Unknown
LoggingStatus       : Undefined
DisplayName         : AppSense Management Server 10.0 (DEFAULT)
```

You can also specify filter and optional parameters including:

- -InstanceId –The GUID that uniquely identifies this product instance on the machine. For example, f321d748-8600-4cd1-beb9-8f1fc6dd0b85 would identify the above product instance.
- -Name –The full name associated with a product instance on the machine. This parameter is case insensitive. The default instance is always named “DEFAULT”.
- -ProductName – The part of the product name associated with a product instance on the machine. This parameter is case sensitive.
- -IsDefault – Returns details on the default instance.
- -AsJson – Returns the instance in json format.

Loading SCP cmdlets for a Product Instance

Before using the SCP cmdlets, you must specify the instance you want to manage using the Import-ApsInstanceModule cmdlets. For example:

```
PS C:\> Import-ApsInstanceModule -ProductName "Personalization Server" -IsDefault
```

You can specify:

- No parameters – loads the SCP cmdlets for the only product instance installed on the machine as long as there is only a single product instance installed.
- -InstanceId – loads the SCP cmdlets for the specified product instance.
- Any combination of -Name, -ProductName loads the SCP cmdlets for the matched product instance as long as there is only a single product instance installed that matches the specified parameters.

The following example loads the cmdlets for a Personalization Server named ‘Instance1’:

```
PS C:\> Import-ApsInstanceModule -ProductName "Personalization Server" –Name Instance1
```

Viewing the currently loaded Product Instance

After loading cmdlets for a product instance, you can use the Get-ApsCurrentInstance cmdlet to see which instance is loaded. For example:

```
PS C:\> Get-ApsCurrentInstance

Name                : DEFAULT
InstanceId          : 935bfcca-23d0-4a08-93e4-064025b91d7b
ProductCode         : 05aa1fcd-7e88-408d-992b-4c96926bdd22
OriginalUpgradeCode : 935bfcca-23d0-4a08-93e4-064025b91d7b
OriginalProductCode : 05aa1fcd-7e88-408d-992b-4c96926bdd22
Version             : 10.0.604.0
InstallPath         : C:\Program Files\AppSense\Management Center\Server\Bin\
CmdletPath          : C:\Program Files\AppSense\Server Configuration
                      Portal\Bin\AppSense.ServerConfiguration.PowerShell.dll;
                      C:\Program Files\AppSense\Management Center\Server\Bin\
                      AppSense.ManagementServer.PowerShell\AppSense.Management
                      Server.PowerShell.dll;C:\Program Files\AppSense\Management
                      Center\Server\Bin\AppSense.ManagementServer.PowerShell\
                      AppSense.ManagementServer.PowerShell.psd1
ProductName         : AppSense Management Server 10.0
IsDefault           : True
Status              : Unknown
LoggingStatus       : Undefined
DisplayName         : Get-ApsCurrentInstance
 Management Server 10.0 (DEFAULT)
```

Server Installation

Completing installation of an Ivanti product server involves:

- Installing any missing prerequisites
- Creating a database
- Initializing the product server and connecting to the database

Installing Prerequisites

Each Ivanti product server specifies its own list of prerequisites. This list normally includes:

- IIS
- ASP.NET

To view missing prerequisites, use the Get-ApsPrerequisite cmdlet. For example, to list missing prerequisites:

```
PS C:\> Get-ApsPrerequisite
```

Missing prerequisites can be installed using the Install-ApsPrerequisite cmdlet. For example, to install all missing prerequisites:

```
PS C:\> Install-ApsPrerequisite -All
```

You can also specify a subset/individual prerequisite using the Where-Object cmdlet. For example, to install the BITS prerequisite only:

```
PS C:\> $prerequisite = Get-ApsPrerequisite | Where-Object { $_.Name -like "*BITS*" }
PS C:\> Install-ApsPrerequisite -Prerequisite $prerequisite
```

Setting Up a New Server and Database

During the creation of product databases and configuration of product servers various credentials are required:

- -ConfigurerCredential – specifies the credentials that will be used to connect to the SQL Server during database creation and schema modification. This can be a dbo account or another user with appropriate privileges. This parameter defaults to the current logged in user.
- -ServiceCredential – specifies the credentials that will be used by the product server to connect to the database during normal operation. This parameter is required during database creation to allow the database roles for the specified credentials to be created/mapped.

Use Get-Credential to create a credential object for configurer/service credentials:

```
PS C:\> $credential = Get-Credential

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential
```

Additional parameters indicate what type of credentials are supplied:

- -ConfigurerSqlAuthentication – if present indicates that the configurer account is a SQL login. If missing, the configurer account is a Windows account. The configurer account must already exist in the database.
- -ServiceSqlAuthentication – if present indicates that the service account is a SQL login. The powershell cmdlets will create this sql login in the database if it doesn't already exist – if it does exist the password must match the one supplied.
- -ServicePassThroughAuthentication – if present indicates that the service account is a gMSA (group managed service account). This is a virtual account specified with a terminal $ and no password (e.g MyGMSA$). The cmdlets ensure that the services and web applications run under this account and connect to the database directly. The cmdlets create an appropriate login in the database.

ServiceSqlAuthentication and ServicePassThroughAuthentication cannot be specified together. If neither is present, the service account is a Windows account. The account must already exist in Active Directory, but the cmdlets will create a login for it in the database if missing.

Use the Initialize-ApsDatabase cmdlet to create a product database:

```
PS C:\> Initialize-ApsDatabase -DatabaseServer "SQLSERVER\Instance" -DatabaseNam
e ManagementServer –ConfigurerCredential $configurerAccount -ServiceCredential
$serviceAccount –DatabaseConnection "NewConnection" –Verbose

VERBOSE: Starting Initialize-ApsDatabase
VERBOSE: DatabaseServer: SQLSERVER\Instance
VERBOSE: DatabaseName: ManagementServer
VERBOSE: DatabaseConnection: NewConnection
VERBOSE: Initialize-ApsDatabase: Connecting to database 'ManagementServer'
VERBOSE: Initialize-ApsDatabase: Applying latest database schema
VERBOSE: Initialize-ApsDatabase: Applying default configuration to database
'ManagementServer'
VERBOSE: Initialize-ApsDatabase: Creating database login 'serviceAccount' of type
'Impersonate' for web directory and Windows services access
VERBOSE: Initialize-ApsDatabase: Setting web directories to use new database login
VERBOSE: Completed Initialize-ApsDatabase
```

Note If the –ConfigurerCredential is not supplied then Initialize-ApsDatabase cmdlet will default to using Windows Impersonation. On completion the DatabaseConnection.xml will not contain the configurer credentials. This can be fixed within the Server Configuration Portal.

Once a database has been created, then the server can be created using the Initialize-ApsServer cmdlet. The DatabaseServer, DatabaseName, ConfigurerCredential, and ServiceAccount parameters can all be specified again, or instead, the DatabaseConnection parameter can be used by specifying the name of the DatabaseConnection created during the Initialize-ApsDatabase step.

```
PS C:\> Initialize-ApsServer –DatabaseConnection "NewConnection" –Verbose

VERBOSE: Starting Initialize-ApsServer
VERBOSE: WebsiteName: Management
VERBOSE: WebsiteAuthentication: Windows
VERBOSE: DatabaseServer: SQLSERVER\Instance
VERBOSE: DatabaseName: ManagementServer
VERBOSE: DatabaseConnection: NewConnection
VERBOSE: ConnectionString:
VERBOSE: InstallMode:
VERBOSE: Initialize-ApsServer: Loading recommendations
VERBOSE: Initialize-ApsServer: Creating application pools
VERBOSE: Initialize-ApsServer: Applying application pool defaults
VERBOSE: Initialize-ApsServer: Creating and initializing IIS directories
VERBOSE: Initialize-ApsServer: Applying web directory defaults
VERBOSE: Initialize-ApsServer: Creating database login 'serviceAccount' of type
'Impersonate' for web directory and Windows services access
VERBOSE: Initialize-ApsServer: Setting web directories to use new database login
VERBOSE: Initialize-ApsServer: Registering and starting Windows services
VERBOSE: Initialize-ApsServer: Applying service defaults
VERBOSE: Initialize-ApsServer: Setup complete
```

```
VERBOSE: Completed Initialize-ApsServer
```

In addition to the parameters specified above, Initialize-ApsServer can be supplied with the following parameters:

- WebSiteName – the name of the website this server will be placed on. The website must already exist.
- WebsiteAuthentication – comma separated list of authentication types to be applied to the server. A website will only apply authentication types configured as allowable.
- InstallModes
- ConnectionString – the additional parameters to add to the default connection string. If you have configured SQL AlwaysOn with Availability Group Multi-Subnet Failovers, you must handle the MultiSubnetFailover value in the database connection string. See the example below.
- HostName – the custom IIS Website binding host name
- HostPort – the custom IIS Website binding port number
- Port – the local IIS Website binding port number

The following example sets MultiSubnetFailover=true on the connection string:

```
$db = "ms0425"
$sql = "gw-sql\mssql2012"
import-apsinstancemodule
$p="password"|convertto-securestring -asplaintext -force;
$c=new-object system.management.automation.pscredential("amc_service",$p)
$p2="password"|convertto-securestring -asplaintext -force;
$c2=new-object system.management.automation.pscredential("DEVELOPMENT\amc_admin",$p2)
initialize-apsdatabase -databaseconnection db1 -databaseserver $sql -databasename $db -
servicecredential $c -servicesqlauthentication -configurercredential $c2 -
connectionstring 'MultiSubnetFailover=True;'
initialize-apsserver -databaseconnection db1
```

Upgrading an existing Server and Database

To upgrade a product instance:

- Install the product server update – install the appropriate MSI or MSP file from the product server release, available from the Ivanti community website (https://community.ivanti.com/community/appsense).
- Upgrade the database – Use Initialize-ApsDatabase with the same parameters to upgrade the database to the version needed for the installed product server.

Use Get-Credential to create a credential object for configure or service credentials:

```
PS C:\> $credential = Get-Credential

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential
```

If upgrading from 8.x, pass in the same parameters used to initially create the database.

```
PS C:\> Initialize-ApsDatabase -DatabaseServer "SQLSERVER\Instance" -DatabaseNam
e ManagementServer –ConfigurerCredential $configurerAccount -ServiceCredential
$serviceAccount -Verbose
VERBOSE: Starting Initialize-ApsDatabase
VERBOSE: DatabaseServer: SQLSERVER\Instance
VERBOSE: DatabaseName: ManagementServer

Database Upgrade
Database 'ManagementServer' is out of date and requires upgrading. Clients will not be
able to connect to this server until the database has been upgraded.
Upgrade Database?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"):y
VERBOSE: Initialize-ApsDatabase: Creating database
VERBOSE: Initialize-ApsDatabase: Connecting to database
'ManagementServer'
VERBOSE: Initialize-ApsDatabase: Applying latest database schema
VERBOSE: Initialize-ApsDatabase: Applying default configuration to database
'ManagementServer'
VERBOSE: Completed Initialize-ApsDatabase
```

If upgrading from 10.x, a database connection parameter can be specified instead:

```
PS C:\> Initialize-ApsDatabase –DatabaseConnection 'NewConnection' -Verbose
```

You can use the ReportOnly option in Initialize-ApsDatabase to find out beforehand what the command is going to upgrade:

```
PS C:\> Initialize-ApsDatabase -DatabaseServer "SQLSERVER\Instance" -DatabaseNam
e ManagementServer –ConfigurerCredential $configurerAccount -ServiceCredential
$serviceAccount -ReportOnly
```

```
Existing database "ManagementServer" on server "SQLSERVER\Instance" to be upgraded from
Server Configuration version 13.2.0 to version 13.3.0

Existing database "ManagementServer" on server "SQLSERVER\Instance" to be upgraded from
version 10.0.0 to version 10.1.0

Upgrade details:

   10.1.0      Upgrade from earlier major version
   10.1.0      Create/replace [dbo].[Machine_Create]
   10.1.0      Create/replace [dbo].[DiscoveredMachines_Create]
```

Cloning an existing Server

The SCP cmdlets provide the ability to export and import server configuration settings. This allows the entire configuration of a product server to be cloned between machines. The configuration is stored as an XML file and information within the file is encrypted using a password when the configuration is exported. This password must be specified when configuring a server using the configuration file.

To export configuration settings, use the Export-ApsServerImage cmdlet:

```
Export-ApsServerImage -Path "PersonalizationServer.xml" -Password "password"
```

To configure a server using a configuration file use the Initialize-ApsServer cmdlet with the -Path and -Password parameters:

```
Initialize-ApsServer -Path "PersonalizationServer.xml" -Password "password"
```

Encrypted SQL Communication

Encrypted SQL connections are supported as of 2023.3. The following commands:

- Initialize-ApsDatabase,
- Initialize-ApsServer,
- Set-ApsServerDatabase

now have two additional switches, either can be used to encrypt the communication:

- EncryptConnection

  Specifies that all SQL communication by the server is to be encrypted.

- ValidateServerCertificate

  Specifies that all SQL communication by the server is to be encrypted, and that the SQL server certificate must be trusted.

Server Configuration Portal Cmdlets

The SCP comes with a new PowerShell module to support operations provided by the new web interface. The cmdlets defined within this module can be accessed via a standard Import-Module:

```
Import-Module AppSense.ServerConfigurationPortal.PowerShell
```

After successfully importing, the following addition cmdlets will be available within the PowerShell session:

- Get-ApsDatabaseConnection
- Get-ApsServerInstance
- Get-ApsServerInstanceDetail
- Remove-ApsDatabaseConnection

Use the Get-ApsDatabaseConnection to obtain the list of all DatabaseConnections local to the installed SCP:

```
PS C:\Windows\system32> Get-ApsDatabaseConnection


ConnectionId         : b4e1cd90-5431-41f4-821e-f033396cfaf9
ServerType           : 935bfcca-23d0-4a08-93e4-064025b91d7b
FriendlyName         : AMC-Conn
SqlInstance          : SQLSERVER\Instance
DatabaseName         : ManagementServer
ConnectionString     :
ServiceCredential    : ServerConfig.ConnectionCredential
ConfigurerCredential : ServerConfig.ConnectionCredential
UpgradeStarted       :
IsUpgrading          : False
```

The Get-ApsServerInstance cmdlet is used by the SCP to build its tree view and is a lightweight version of Get-ApsInstance.

```
PS C:\Windows\system32> Get-ApsServerInstance


ServerType   : Management
InstanceId   : 935bfcca-23d0-4a08-93e4-064025b91d7b
InstanceName : DEFAULT
Local        : True
HostName     : VMM-2012R2
```

The Get-ApsServerInstanceDetail cmdlet is used by the SCP to provide information on a specific instance.

```
PS C:\> Get-ApsServerInstanceDetail -InstanceId 935bfcca-23d0-4a08-93e4-064025b91d7b

ServerType            : Management
InstanceId            : 935bfcca-23d0-4a08-93e4-064025b91d7b
InstanceName          : DEFAULT
Local                 : True
HostName              :
WebSite               : Management
HostNameBinding       : Management.DesktopNow.AppSense
HostPort              : 80
LocalPort             : 7751
AuthTypes             : {Windows}
SupportedAuthTypes    : {Anonymous, Windows}
DatabaseSettingsName  :
DatabaseName          :
DatabaseInstance      :
BindingInfo           :
{AppSense.ServerConfiguration.PowerShell.Model.ApsServerBindingDetails,

AppSense.ServerConfiguration.PowerShell.Model.ApsServerBindingDetails}
Status                : Unconfigured
Logging               : Undefined
Variances             : {}
ConfigurationBindings :
{AppSense.ServerConfiguration.PowerShell.Model.ApsServerBindingDetails,

AppSense.ServerConfiguration.PowerShell.Model.ApsServerBindingDetails}
```

You can specify:

- -InstanceId – returns information on the specified instance
- -InstanceName – returns information on the specified instance
- -HostName – returns information for all instances on a given host

The Remove-ApsDatabaseConnection cmdlet is used by the SCP to remove DatabaseConnections local to the SCP.

```
PS C:\> Remove-ApsDatabaseConnection -DatabaseConnection "NewConn" -Type Management
```

You can specify:

- -DatabaseConnection – The name of the DatabaseConnection to remove
- -Type – The DatabaseConnection type, either Management or Personalization.

Database Management

View Database Connection details

Use the Get-ApsDatabaseDetails cmdlet to display details of the current instances DatabaseConnection:

```
PS C:\> Get-ApsDatabaseDetail


DatabaseConnection : PS-Database-Conn
ServerName         : SQLSERVER\Instance
SqlDbName          : ManagementServer
ConnectionString   :
ServiceAccount     : ServiceCredentials
ServiceAuthType    : Impersonate
ServicePassword    : System.Security.SecureString
DatabaseState      : UpToDate
CurrentVersion     : 10.0.0.0
LatestVersion      : 10.0.0.0
```

Update Database Connection details

Use the Update-ApsDatabase cmdlet to perform either a Schema upgrade and/or a Data Upgrade of an existing out of date database:

```
PS C:\> Update-ApsDatabase –DatabaseConnection "AMC-Database-Conn"
```

Viewing Product Database Scripts

Use the Get-ApsDatabaseScript cmdlet to display the database scripts for a product:

```
PS C:\> Get-ApsDatabaseScript

Name                Description                 ScriptType AppliesTo
----                -----------                 ---------- ---------
Create Database     Database creation        CreateDatabase 10.1
Get Version         Version retrieva...  GetDatabaseVersion 10.1
IsDbEmpty           Checks If Databa...     IsDatabaseEmpty 10.1
Create Login        Creates a Login ...        CreateLogins 10.1
Purge Events        Purges all event...           BatchJobs 10.1
Create Schema       Create Schema              CreateSchema 10.1
Upgrade Schema 7... Upgrade Schema 7...       UpgradeSchema 7.0
Upgrade Schema 7... Upgrade Schema 7...       UpgradeSchema 7.1
Upgrade Schema 7... Upgrade Schema 7...       UpgradeSchema 7.2
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.0
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.1
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.2
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.3
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.4
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.5
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.6
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.7
Upgrade Schema 8... Upgrade Schema 1...       UpgradeSchema 10.0
```

Exporting Database Creation Scripts

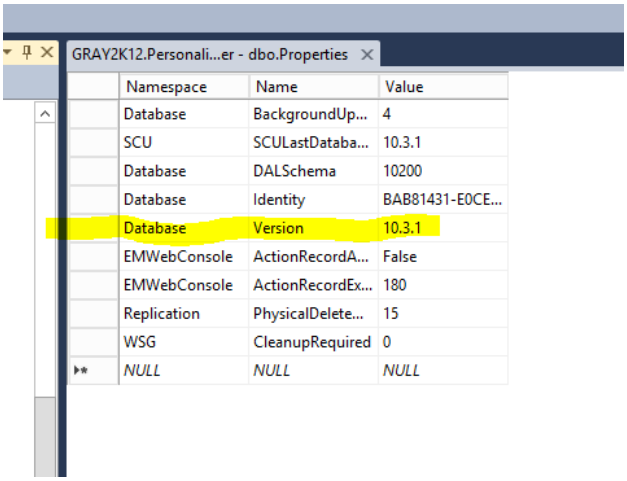
Use the Export-ApsDatabaseScript cmdlet to export database scripts.

```
PS C:\> Export-ApsDatabaseScript -ScriptType "CreateDatabase, CreateSchema,
CreateLogins" -Path C:\Scripts
PS C:\> Get-ChildItem C:\Scripts


    Directory: C:\Scripts


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---        17/07/2013     17:36       2281 Create_Database.sql
-a---        17/07/2013     17:36       4501 Create_Login.sql
-a---        17/07/2013     17:36     349344 Create_Schema.sql
```

Exporting Database Upgrade Scripts

To export scripts to upgrade a database, always use the cmdlet **Export-ApsDatabaseScript** and specify the exact database version with the **-AppliesTo** parameter. The **Get-ApsDatabaseScript** cmdlet is **deprecated** and should no longer be used - it does not correctly process the AppliesTo parameter and will be removed or updated in a future release.

It is necessary to provide the exact version of the database to be upgraded (including all components, e.g. 10.1.33) for the AppliesTo parameter. The cmdlet will then generate the correct SQL script for the upgrade. The version can be found on the landing page of the personalization side of the EM Console (before upgrading), or in the database table dbo.Properties, in the row with Namespace 'Database' and Name 'Version':



An example is:

```
Import-ApsInstanceModule -ProductName PersonalizationServer -IsDefault

Export-ApsDatabaseScript -ScriptType UpgradeSchema -AppliesTo 10.0.56 | out-file Upgrade.sql
```

The exported script will have a check that it is being applied to the correct database version.

Connecting a Server to a Database

Use the Set-ApsServerDatabase cmdlet to quickly switch a server to use a different database. You must have previously used Initialize-ApsServer to ensure the server has completed initial configuration.

To switch a server to use a database for which a DatabaseConnection exists, use the –DatabaseConnection parameter.

For example:

```
PS C:\> Set-ApsServerDatabase -DatabaseConnection "PS-Database-Conn" -Verbose
```

To switch a server to use a database where no existing DatabaseConnection exists, specify the required parameters.

For example:

```
PS C:\> Set-ApsServerDatabase -DatabaseServer "SQLSERVER\Instance" -DatabaseName
 "PersonalizationServerNew" -ServiceCredential $credential -Verbose
```

Server Management

View Server details

Use the Get-ApsServerDetail cmdlet to display details of the current instances website:

```
PS C:\> Get-ApsServerDetail


WebSite                  : Management
HostName                 : Management.DesktopNow.AppSense
HostPort                 : 7751
LocalPort                : 80
AuthTypes                : {Windows}
SupportedAuthTypes       : {Anonymous, Windows}
DatabaseSettingsName     : NewConn
DatabaseName             : ManagementServer
DatabaseInstance         : SQLSERVER\Instance
Status                   : Started
Logging                  : Disabled
Variances                : {}
BindingInfo              :
{AppSense.ServerConfiguration.PowerShell.Model.ApsServerBindingDetails,

AppSense.ServerConfiguration.PowerShell.Model.ApsServerBindingDetails}
ConfigurationBindingInfo :
{AppSense.ServerConfiguration.PowerShell.Model.ApsServerBindingDetails,

AppSense.ServerConfiguration.PowerShell.Model.ApsServerBindingDetails}
OriginalUpgradeCode      : 935bfcca-23d0-4a08-93e4-064025b91d7b
InstanceId               : 935bfcca-23d0-4a08-93e4-064025b91d7b
InstanceName             : DEFAULT
```

The full set of binding information for the website is contained within the BindingInfo array. To access this array, use the following commands:

```
PS C:\> $serverDetails = Get-ApsServerDetail
PS C:\> $serverDetails.BindingInfo.Count
2

PS C:\> $serverDetails.BindingInfo[0] | Format-List

IpAddress : *
Host      :
Port      : 8081
Protocol  : http


PS C:\> $serverDetails.BindingInfo[1] | Format-List
```

```
IpAddress : *
Host      : MyHostHeader
Port      : 9091
Protocol  : http
```

Resetting a Server to an out of box state

The Reset-ApsServer cmdlet can be used to reset a server to an out of box state. During a reset:

- All services and application pools are stopped/removed.
- All web directories are removed.
- The product configuration is deleted.
- The server stops running.

For example:

```
PS C:\> Reset-ApsServer
```

Starting or Stopping a Server

The Start-ApsServer and Stop-ApsServer cmdlets can be used to start or stop a product server. Starting or stopping a product server starts/stops all services and application pools associated with the product server. Product consoles may display an error message to users if they attempt to connect or perform any actions whilst a server is stopped.

To stop a server, use the Stop-ApsServer cmdlet. For example:

```
PS C:\> Stop-ApsServer
```

To start a server, use the Start-ApsServer cmdlet. For example:

```
PS C:\> Start-ApsServer
```

The current state of a server can be determined using the Get-ApsCurrentInstance cmdlet. For example:

```
PS C:\> Get-ApsCurrentInstance
```

Review the status field to determine if a server is started, stopped or unconfigured.

Changing Website

The Set-ApsWebSite cmdlet allows a configured server instance to migrate from its current website to a new one. For example:

```
PS C:\> Set-ApsWebSite –WebSiteName "NewWebSite"
```

If the website name specified by the –WebSiteName parameter does not exist, it will be created. If it does exist, the website will be used. The old website will be stopped.

Viewing and Fixing Variances

If a server is encountering problems, use the Get-ApsVariance cmdlet to highlight known issues:

```
PS C:\> Get-ApsVariance
```

Many variances can be fixed automatically. Use the Repair-ApsVariance cmdlet to attempt to repair variances:

```
PS C:\> Repair-ApsVariance -All
```

You can also limit the variances that will be repaired using the Where-Object cmdlet. For example:

```
PS C:\> $variance = Get-ApsVariance | Where-Object { $_.Name -eq "" }
PS C:\> Repair-ApsVariance -Variance $variance
```

Enabling or Disabling Logging

Logging can be enabled/disabled using the Enable-ApsLogging and Disable-ApsLogging cmdlets. Enabling logging creates log files in product server installation directory:

```
PS C:\> Enable-ApsLogging
```

Once an issue requiring logging has been resolved, use the Disable-ApsLogging cmdlets to disable logging:

```
PS C:\> Disable-ApsLogging
```

Additional Configuration

Changing the HTTP listener port for WinRM

The Server Configuration Portal (SCP) can now be configured to communicate with remote personalization and management servers using non-default WinRM ports, non-default UrlPrefixes and either http or https transports.

By default, the SCP will continue to use http port 5985 and UrlPrefix ‘wsman’ to communicate with remote servers. If a different configuration is required, the SCP web.config file must be modified. This can typically be found here:

C:\Program Files\AppSense\Server Configuration Portal\web.config

Open the file and insert a line matching your environment into the <WinRMPorts> section. For example:

<Server Name=”MyServer” ServerURL=”MyServer.MyCompany.com” Transport=”MyTransport” UrlPrefix=”MyUrlPrefix” Port=”MyPort” />

If required separate <Server> elements can be added for each Management and Personalization server in your environment. If no server element is found for a server, the default <Server> element with name * is used.

The attributes that can be used are:

Attribute	Description
Name	This attribute is required and can be either the NetBIOS name as shown in the SCP or *.
ServerUrl	This is an optional attribute and is used to provide the full URL of the server. This is required for https connections where the full server name must match the SSL certificate. If omitted the SCP will use the Name attribute to connect.
Transport	Http or Https
UrlPrefix	This is an optional attribute and is used by the WinRM listener. If omitted the default PowerShell value of ‘wsman’ is used.
Port	The TCP port used by the server.

Scripting Multiple/Remote Servers

Enabling Remoting

Windows PowerShell remoting allows scripts to communicate with and run Windows PowerShell cmdlets on remote machines. No configuration is required to enable a computer to send remote commands. However, to receive remote commands, Windows PowerShell remoting must be enabled on the machine.

Windows PowerShell remoting is enabled on Windows Server 2012 by default. On all other systems, use the Enable-PSRemoting cmdlet to enable it. You can also use the Enable-PSRemoting cmdlet to re-enable remoting on Windows Server 2012 if it is disabled.

To enable remoting, open a Windows PowerShell console as an administrator and enter:

```
PS C:\> Enable-PSRemoting
```

Note

For more information refer to *Enable-PSRemoting* on Microsoft TechNet:

http://technet.microsoft.com/en-us/library/hh849694.aspx

Enabling CredSSP

CredSSP allows credentials to be passed to a secondary machine on the network. This allows the SCP cmdlets to:

- Connect to SQL server databases using the current user.
- Read server images (used for cloning servers) from a network share.

Whilst CredSSP is not required, if it is not used you must specify the –ConfigurerCredential parameter wherever it is used.

Enabling CredSSP requires changes on both server and client machines:

- On a server machine, to allow CredSSP connections, use:

  ```
  Enable-WSManCredSSP Server
  ```

  Select Yes when prompted to allow CredSSP connections.

- On client machines, to allow outbound CredSSP connections to specified servers, use:

  ```
  Enable-WSManCredSSP Client -DelegateComputer @("server1.domain", "server2.domain")
  ```

  Select Yes when prompted to allow CredSSP connections.

Note

For more information refer to *Enable-WSManCredSS* on Microsoft TechNet:

http://technet.microsoft.com/en-us/library/hh849872.aspx.

Working with PowerShell Sessions

Windows PowerShell allows for the concept of sessions, each of which may load in a separate copy of the SCP cmdlets. You can use Invoke-Command to create a temporary session and run cmdlets or create a session that can be re-used using New-PSSession.

Note

For more information refer to *about_PSSessions* on Microsoft Technet:

http://technet.microsoft.com/en-us/library/hh847839.aspx.

Script Examples

Complete Server Configuration

```
$ProductName = "Personalization Server"
$ConfigurerUsername = "DOMAIN\AppSensePersonalizationServerConfigurer"
$ConfigurerPassword = "password"
$ServiceUsername = "DOMAIN\AppSensePersonalizationServerService"
$ServicePassword = "password"
$DatabaseServer = "SQLSERVER\Instance"
$DatabaseName = "AppSensePersonalizationServer"
$DatabaseConnection = "PS-Database-Conn"


# Load cmdlets for appropriate product
Import-Module AppSenseInstances
Import-ApsInstanceModule -ProductName $ProductName -IsDefault

# Create configurer credential object
$SecurePassword = ConvertTo-SecureString -AsPlainText $ConfigurerPassword -Force
$ConfigurerCredential = New-Object System.Management.Automation.PSCredential
($ConfigurerUsername, $SecurePassword)

# Create service credential object
$SecurePassword = ConvertTo-SecureString -AsPlainText $ServicePassword -Force
$ServiceCredential = New-Object System.Management.Automation.PSCredential
($ServiceUsername, $SecurePassword)

# Install/fix all prerequisites
Get-ApsPrerequisite | Install-ApsPrerequisite

# Create database
Initialize-ApsDatabase -DatabaseServer $DatabaseServer -DatabaseName $DatabaseName -
ConfigurerCredential $ConfigurerCredential -ServiceCredential $ServiceCredential -
DatabaseConnection $DatabaseConnection

# Configure server
Initialize-ApsServer -DatabaseConnection $DatabaseConnection
```

Clone Multiple Servers from the Current Server

```
$Credential = Get-Credential
$ProductName = "Personalization Server"
$Path = "\\server\share\ServerImage.xml"
$Password = "password"
$ComputerName = @("server1.domain", "server2.domain")

Import-ApsInstanceModule -ProductName $ProductName -IsDefault
Export-ApsServerImage -Path $Path -Password $Password

Invoke-Command -ComputerName $ComputerName -ScriptBlock {
    Param($ProductName, $Path, $Password)
    Import-ApsInstanceModule -ProductName $ProductName -IsDefault
    Initialize-ApsServer -Path $Path -Password $Password
} -ArgumentList ($ProductName, $Path, $Password) -Authentication Credssp -Credential
$Credential
```

Return Variances from Multiple Servers

```
$Credential = Get-Credential
$ProductName = "Personalization Server"
$ComputerName = @("server1.domain", "server2.domain")

Invoke-Command -ComputerName $ComputerName -ScriptBlock {
    Param($ProductName, $Path, $Password)
    Import-ApsInstanceModule -ProductName $ProductName -IsDefault
    Get-ApsVariance
} -ArgumentList ($ProductName, $Path, $Password) -Authentication Credssp -Credential
$Credential
```