Technical Note: Microsoft Windows PowerShell™/ SA Integration

IN THIS CHAPTER

This chapter contains the following topics:

Introduction to Microsoft Windows PowerShell™

Windows PowerShell Version 1.0 is a new command-shell for System Administrators and Programmers. It is deeply integrated with Microsoft's .Net 2.0 Framework Class Library (FCL), highly extensible, and quite intuitive. It is available for Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008.

Windows PowerShell uses the .NET common language runtime (CLR) and the .NET Framework, and accepts and returns .NET objects. This enhances the tools and methods available to manage and configure of Windows.

Windows PowerShell provides numerous *cmdlets* (pronounced "command-let"), which are built into the shell and provide a wide range of functionality. Cmdlets can be used individually or in combination to perform more complex tasks.

Windows PowerShell not only enables access to a computer's file system, PowerShell *Providers* allow you to access data stores like the registry and digital signature certificate stores. A *Provider* is a software module that provides a uniform interface between a service and a data source.

Before you attempt to use the Windows PowerShell/SA integration feature, it is assumed that you are familiar with and comfortable using Microsoft Windows PowerShell. If you need background or instruction in using PowerShell, see *http://www.microsoft.com*.

Because the included cmdlets can modify data on your managed servers, it is important that you have a solid understanding of Windows PowerShell and its use.

Windows PowerShell Integration with SA

In SA 6.61 and later, Microsoft Windows PowerShell/SA integration provides initial integration between SA and Microsoft Windows PowerShell on managed servers running Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 (Longhorn).

PowerShell is available from SA user interfaces and SA data and features are available from within the standard PowerShell command-shell environment or from within any PowerShell Runspace. A *PowerShell Runspace* is a hosting environment for the PowerShell runtime system.

As of SA 6.61 and later, the following PowerShell cmdlets are available:

- `Get-SASServer`
- `Set-SASServer`
- `Get-SASJob`

SA 6.61 and later also includes a PowerShell *SAS Provider* (a component that provides access to the objects in a SA core in a PowerShell environment).

Integrated PowerShell/SA Cmdlets

Table 1-1 lists and describes the integrated PowerShell/SA cmdlets included with SA 6.61 and later.

Table 1-1:

CMDLET	DESCRIPTION	ARGUMENTS

`Get-SASServer`	Retrieves server data from specified server(s)	`-Credential <PSCredential>`
		`-Core <Hostname|IPAddress>`
		`-Name < ListOfHostnameFragments> |`
		`-Id <ListOfServerIDs>`

`Get-SASJob`	Retrieves data for specified jobs	`-Credential <PSCredential>`
		`-Core <Hostname|IPAddress>`
		`-JobFilter <ListOfJobIDs>`

`Set-SASServer`	Retrieves a list of managed servers	`-Credential <PSCredential>`
		`-Core <Hostname|IPAddress>`
		`-Server <ServerVO>`

Installation Requirements

An MSI installer package containing the cmdlets and PowerShell SAS Provider assemblies, configuration and setup files for installation on a System Administrator's Windows desktop.

Operating System Support

- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008

Installation

To implement Microsoft Windows PowerShell/SA integration, you must perform the following tasks:

1. Locate the Microsoft Windows PowerShell/SA Connector MSI package in the OCC **Library ➤ Software Policies**.

2 Run the MSI to install the assemblies that define the SA-specific cmdlets and SAS Provider. The file `readme.rtf` provides last minute information. The Microsoft Windows PowerShell inititialization script, `profile.ps1` (similar to `.bashrc`) and a set of sample PowerShell scripts that show how to use PowerShell in an SA environment are also installed.

By default, the MSI installs the connector into `C:\Program Files\Opsware\PsSas`.

The file, `SAS-WSAPI.ps1`, describes accessing the WS-API directly from PowerShell, without the need for cmdlets.

Microsoft PowerShell Integration with SA Features

As of SA 6.61 and later, you will find the Microsoft PowerShell is available as an option in the following areas:

- Remote access to Managed Servers
- Audit and Snapshots Rules
- DSE Script Integration

Remote access to Managed Servers

From the SA Client, you can open a remote PowerShell session for any managed server (not available for a group of servers). As you would when opening a remote terminal

1 Launch the SA Client.

2 From the Navigation pane, select **Devices ➤ All Managed Servers**.

3 Select a Managed Server and open it.

In the Device Explorer window, from the **Actions** menu, select **Launch Remote PowerShell**.

You cannot run a script that contains *WMI calls* while logged in to a remote PowerShell session. If you try to run a script containing WMI call, you will get an `Access Denied` error, even if you are a member of a group with the necessary permissions to run that script.

Audit and Snapshots Rules

Microsoft PowerShell is integrated with the SA Audit and Remediation feature. While configuring a custom script rule, Microsoft PowerShell scripts are now an option along with batch, Python 1.5.2, and Visual Basic. For details about Audit and Remediation custom script configuration, see the Audit and Remediation chapter in the *SA User's Guide: Application Automation*.

DSE Script Integration

For Managed Servers, you can set up PowerShell scripts that call SA APIs using Pytwist so that end users can invoke the scripts as DSEs or ISM controls. For more information about writing scripts that invoke Pytwist APIs, see the *SA Platform Developer's Guide*.

Sample Sessions

This section provides four scenarios that demonstrate using Windows PowerShell/ SA integration.

- Scenario 1 demonstrates extracting managed server data from an SA Core, modifying it, and writing it back to the core.
- Scenario 2 demonstrates exporting SA managed server data to an Excel spreadsheet using Windows PowerShell/SA integration.
- Scenario 3 demonstrates mounting the SA core as a Windows PowerShell PSdrive and navigating around the virtual file system.
- Scenario 4 demonstrates listing all the types of SA objects available to a Windows PowerShell environment.

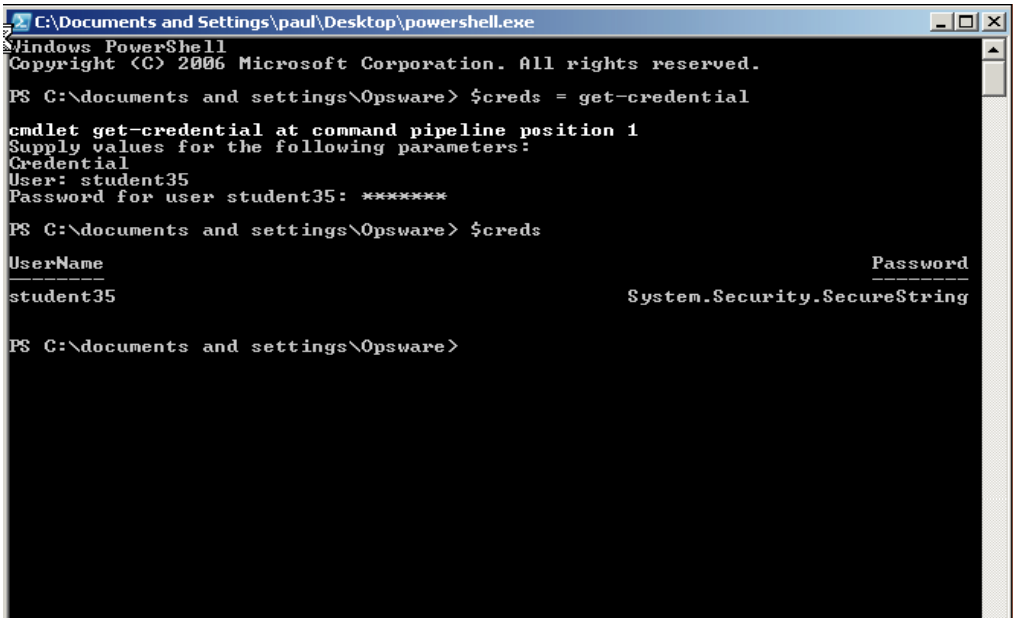
Scenario 1

Authenticating to an SA Core, obtaining data about a managed server, modifying the data, and writing the data back to the SA Core.

1. Open a PowerShell prompt from the desktop icon.

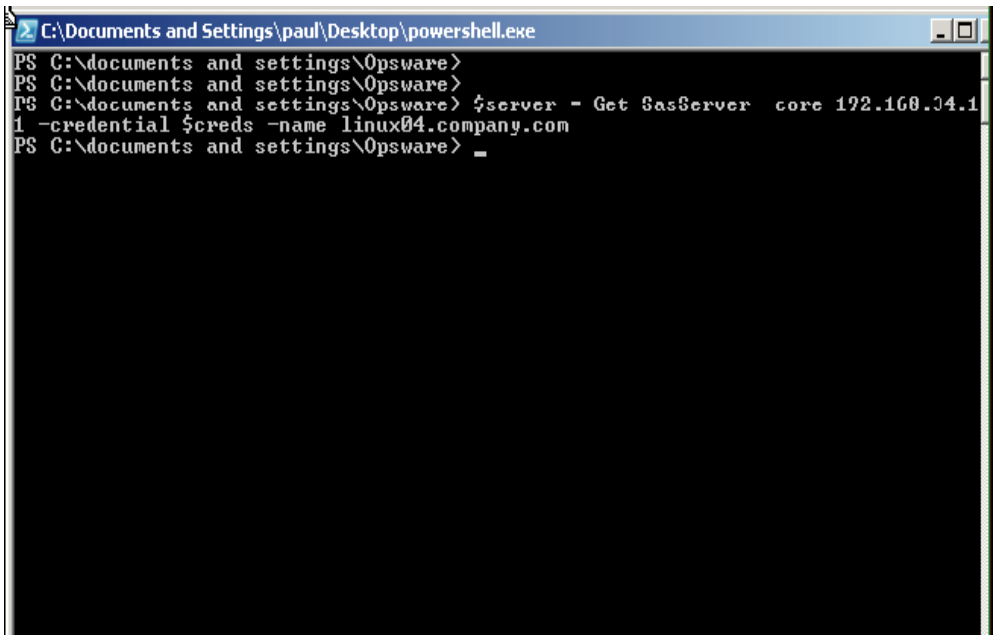
2 Store the SA Core credentials securely in a PowerShell shell variable. See Figure 1-1.

Figure 1-1:



3 Using the `Get-SasServer` cmdlet, you can retrieve the SA record representing a server as shown in Figure 1-2.

Figure 1-2:



The returned object is stored in a shell variable.

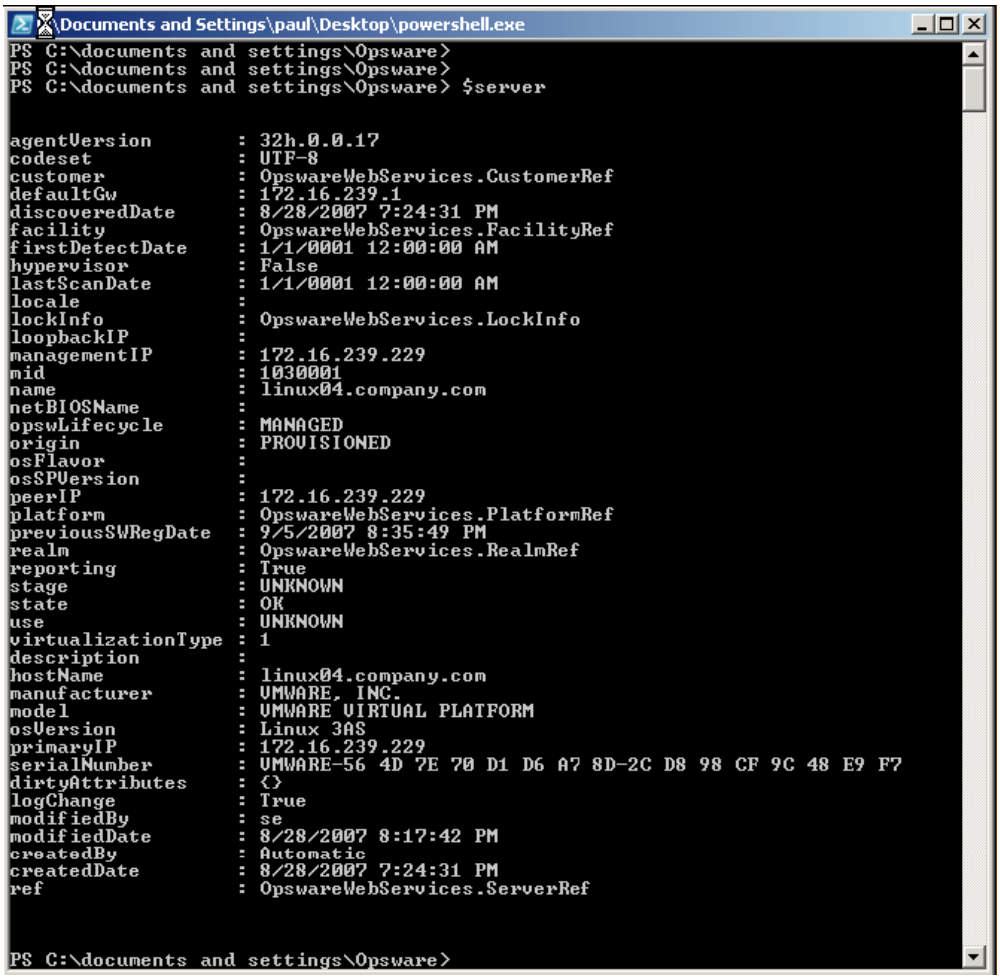
The `Get-SasServer` cmdlet takes a parameter to identify the SA Core from which the server data is to be retrieved, a parameter to supply credentials to the SA core for the operation, identifying and authenticating the SA user account in whose identity the operation is to be attempted, and a parameter to identify the server being requested.

More information on the `Get-SasServer` cmdlet arguments or the arguments for any cmdlet can be obtained by using the PowerShell `Get-Help` base cmdlet, for example:

```
Get-Help Get-SasServer -detailed
```

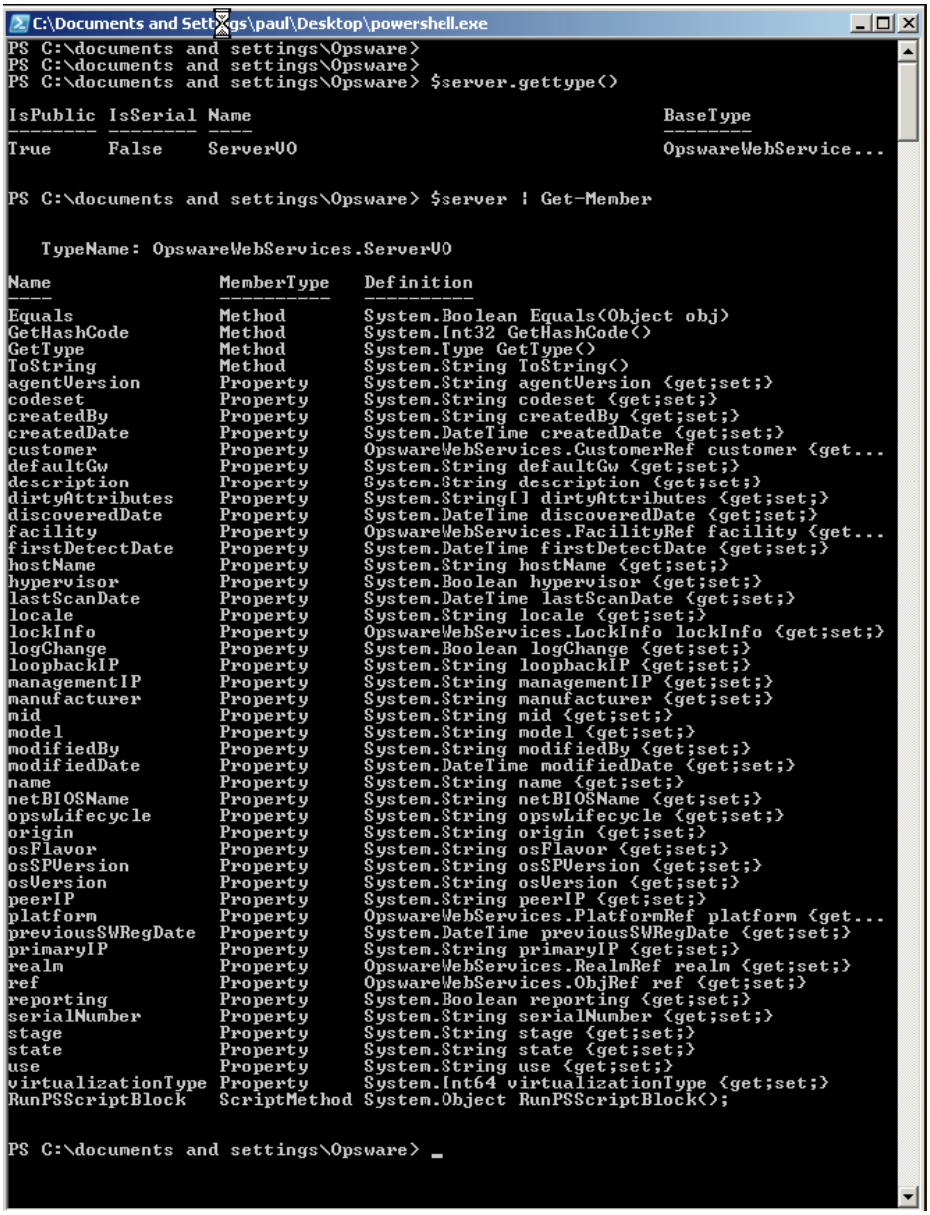
4 You can now examine the properties of the returned object by entering the name of the shell variable. See Figure 1-3

Figure 1-3:



5 List the object's properties, the types of the properties and the methods that can be called on the object from a PowerShell script as shown inFigure 1-4.

Figure 1-4:



6 You can modify the object's **Description** attribute in Windows PowerShell, then call the `Set-SasServer` cmdlet and pass the modified `ServerVO` object to the cmdlet. This cmdlet will take the `ServerVO` object and update the managed server record in the SA Core. The `Set-SasServer` cmdlet takes parameters that identify the SA Core to which the updated data is to be written and credentials identifying the SA user account under whose identity the operation is executed.

At the end of the update operation, the updated `ServerVO` is returned to Windows PowerShell and the properties are displayed at the prompt as shown in Figure 1-5.

Figure 1-5:

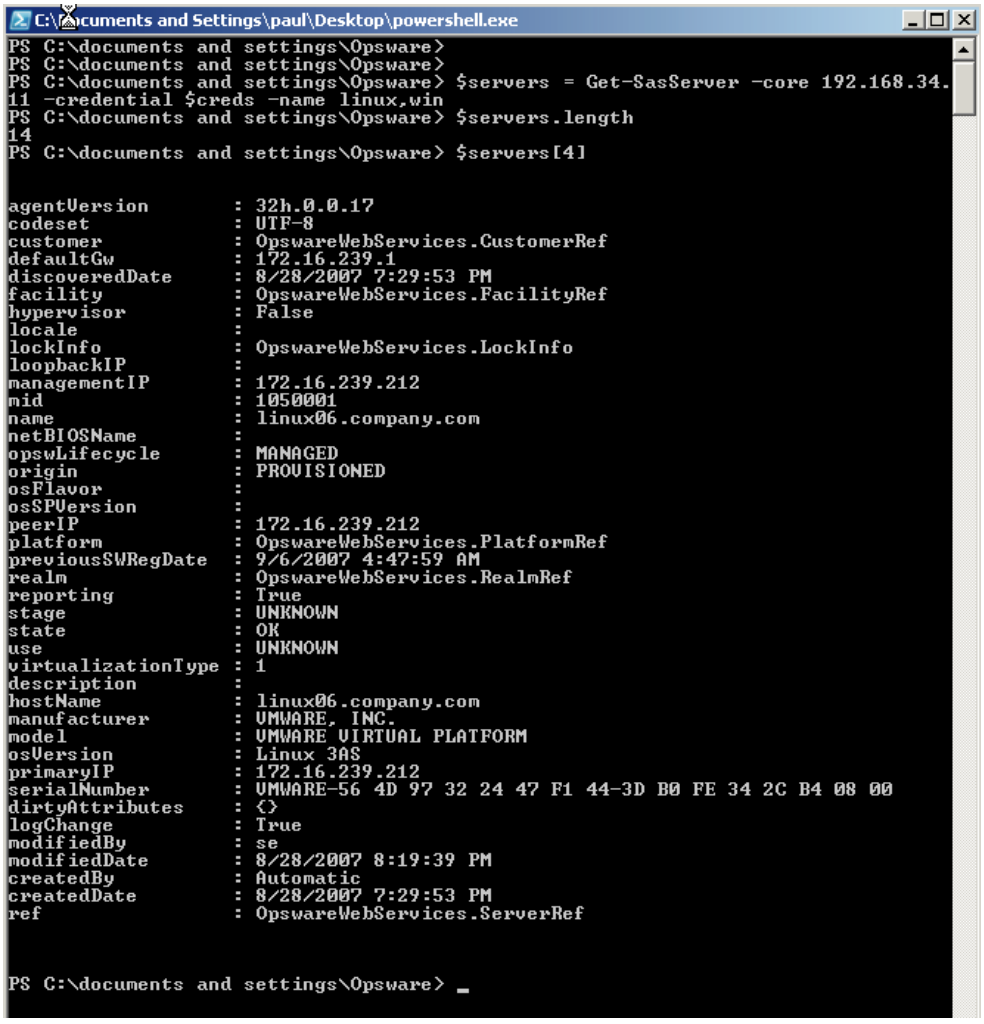
Scenario 2

This scenario demonstrates retrieving all managed server data from the SA Core and displaying it in Microsoft Excel.

1. Use the `Get-SasServer` cmdlet to retrieve `ServerVO`s for each Linux and Windows managed server from the SA Core. In the session below, the `-name` parameter is used to supply a list of name matching filters, for example, `-name linux,win`, to the SA Core.

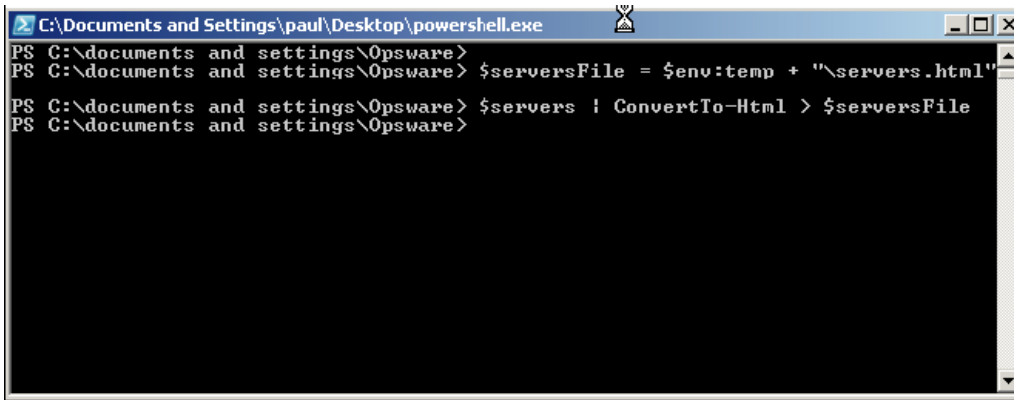
   The `Get-SasServer` cmdlet returns an array of `ServerVO`s that is, in this example, 14 items in length. You can index into this array to examine any one of the `ServerVO` objects. See Figure 1-6.

Figure 1-6:



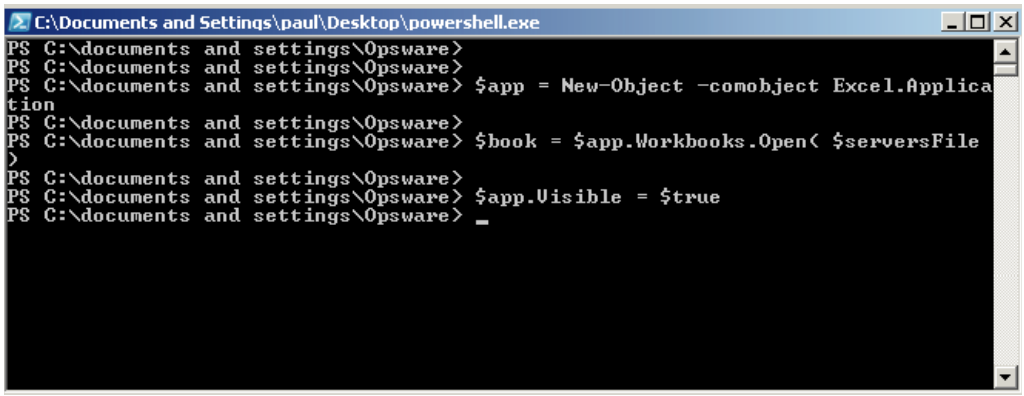
2 No you can format the `ServerVO` data as HTML and save to a temporary file. The temporary file is created in the TEMP directory. In a PowerShell session, to get the value of the `%TEMP%` environment variable, enter `$env:temp`. See Figure 1-7.

Figure 1-7:



3 Using the `New-Object` base Windows PowerShell cmdlet you can launch Microsoft Excel, then create a new workbook inside this instance of Excel, and populate the workbook from the contents of the temporary file. Finally, set the running Excel instance to be visible. This will cause Excel to come to the foreground. Now you can sort the data by date, column value, etc., to determine, for example, the date on which each server came under management in the SA Core. SeeFigure 1-8.

Figure 1-8:

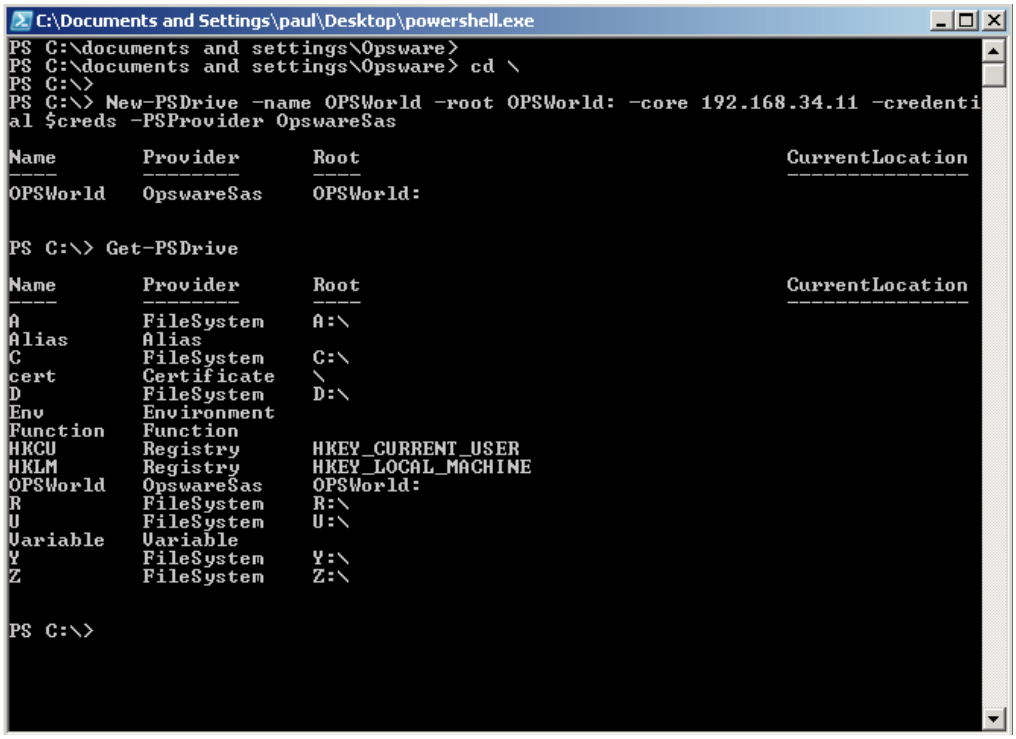


Scenario 3

This scenario demonstrates mounting the SA Core as a Windows PowerShell PSDrive, navigating to the SA **Jobs** folder and retrieving its contents.

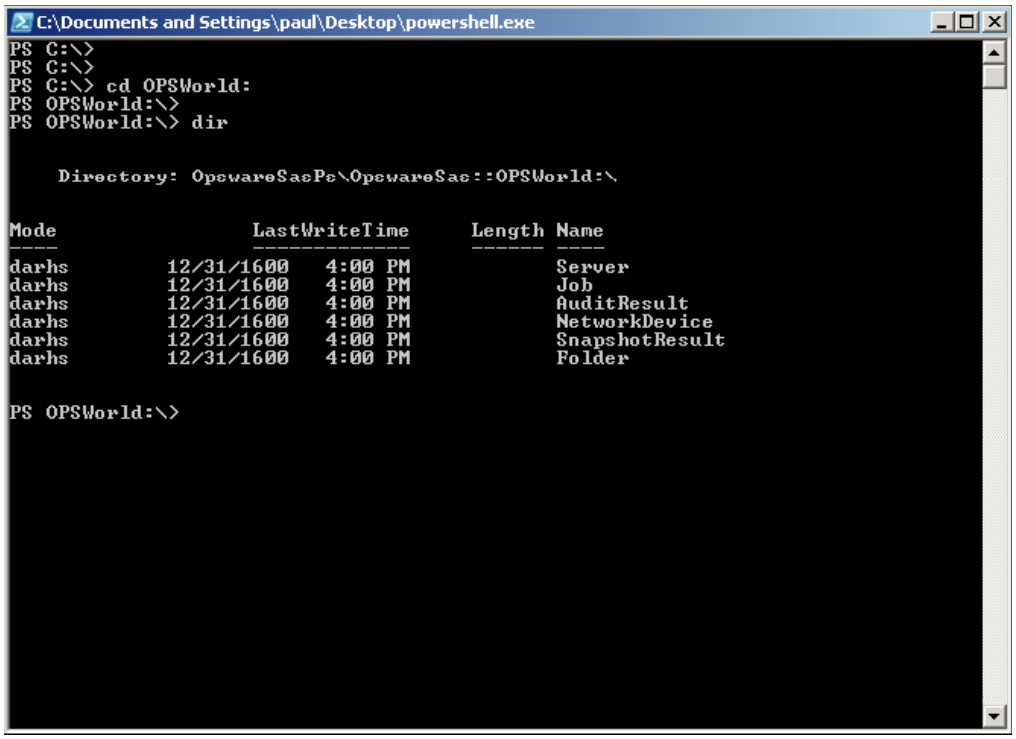
1 Mount the SA core as a Windows PowerShell PSDrive. PowerShell allows different data stores or repositories to be navigated as if they were a file system. In this scenario, you *mount* the SA Core, specifically the managed environment data store, as if it were a drive named `OPSWorld`. The windows PowerShell base system then calls the PowerShell SAS Provider, `-PSProvider OpswareSas`, whenever data is read from or written to this virtual file system – or when the file system is navigated by a client. See Figure 1-9.

Figure 1-9:



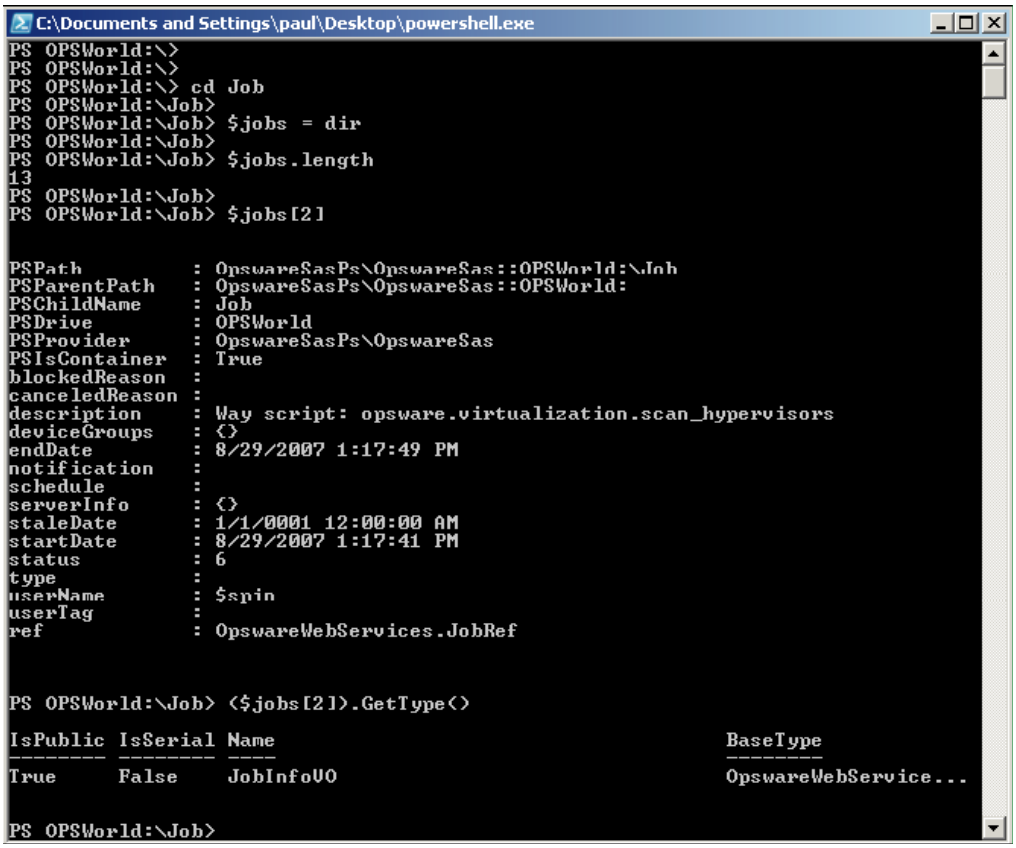
2 Change directory to the newly mounted drive and obtain a directory listing. `dir` is a PowerShell alias for the `Get-ChildItem` cmdlet. See Figure 1-10.

Figure 1-10:



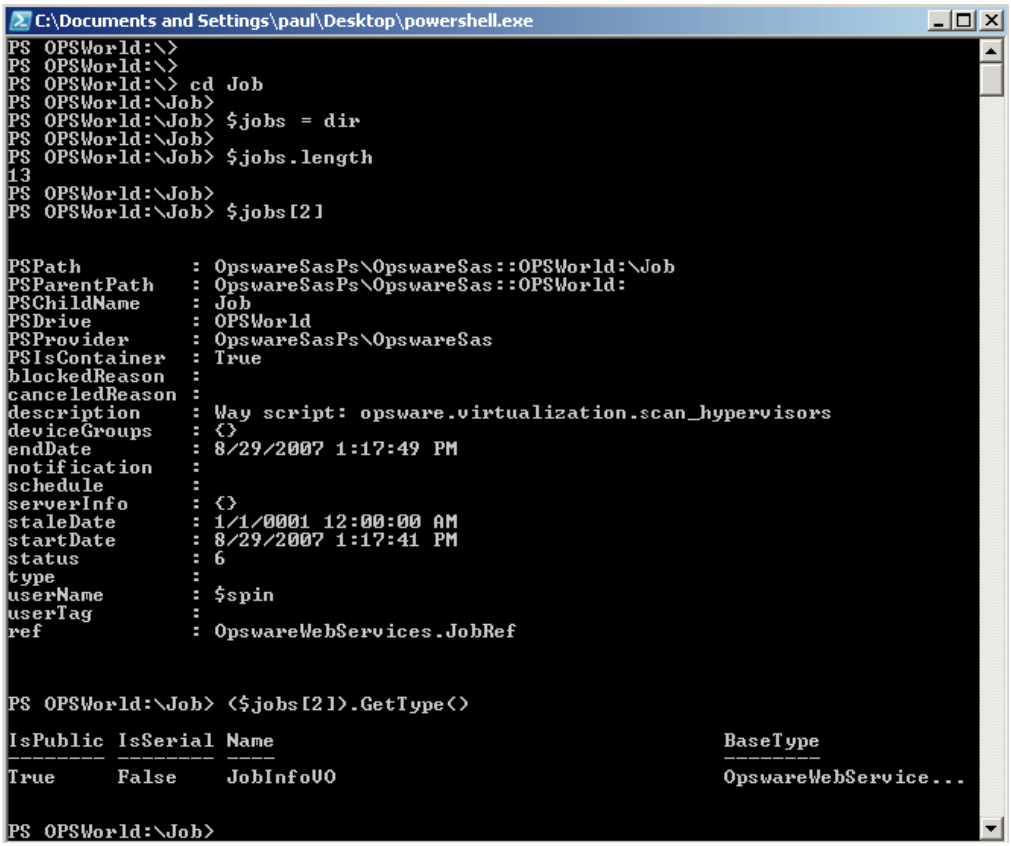
3 Change directory to the **Jobs** folder, get a directory listing, and save the directory listing as a shell variable. This shell variable will contain an array of `JobInfoVO` objects from the SA Core into which you can index. See Figure 1-11.

Figure 1-11:



4 Change directory to the C: drive and remove the `OPSWorld` PSDrive. See Figure 1-12.

Figure 1-12:

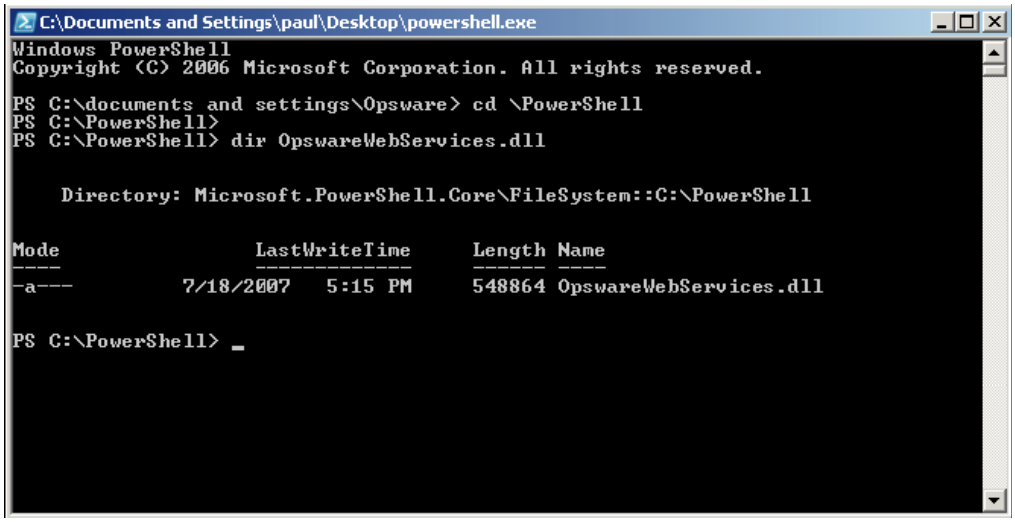


Scenario 4

This scenario describes examining all the types of SA objects available inside the Windows PowerShell environment.

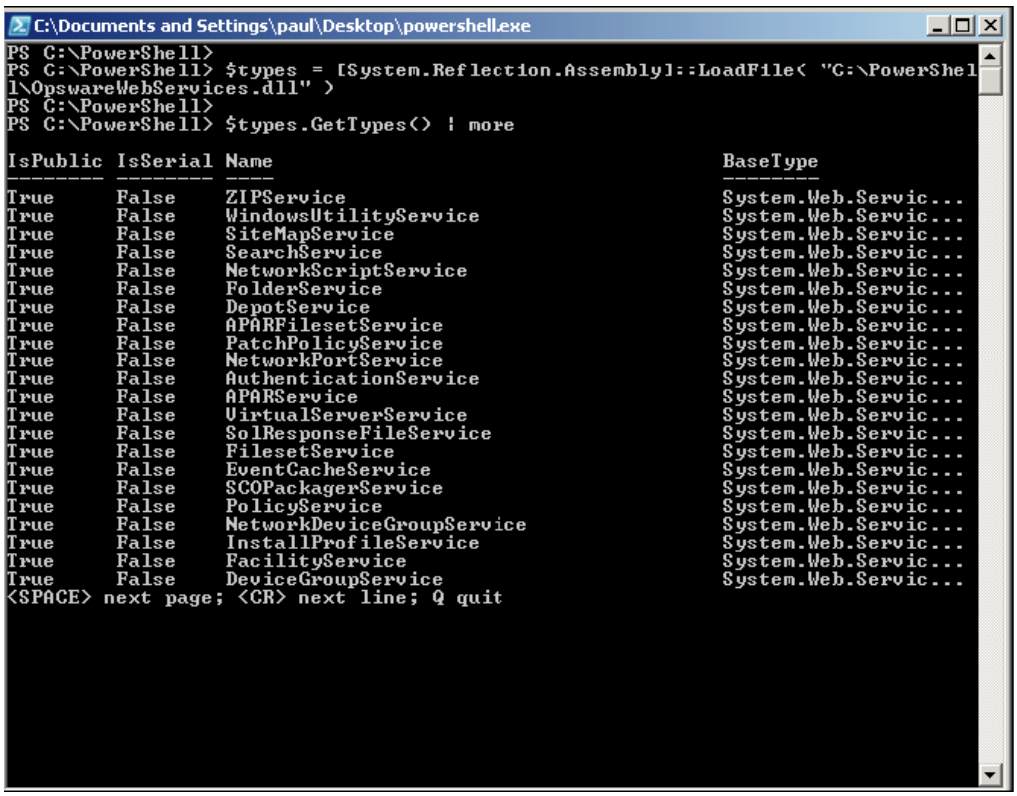
1. Locate the .NET assembly containing the PowerShell SAS Provider and cmdlets. See Figure 1-13.

Figure 1-13:



2 Using .NET Reflection, load the .NET assembly and examine the loaded types. This displays all the SA types that are available for use in the Windows PowerShell environment. See Figure 1-14

Figure 1-14:



3 Create an instance of a `NetworkDeviceVO`. This is a nascent `NetworkDeviceVO`, showing all of the attributes of a network device available for scripting, reporting etc. in the PowerShell environment. See Figure 1-15.

Figure 1-15: