

Microsoft System Center

Introduction to Microsoft Automation Solutions

Rob Costello • Richard Maunsell
Mitch Tulloch, Series Editor

Microsoft System Center

Introduction to Microsoft Automation Solutions

Rob Costello • Richard Maunsell
Mitch Tulloch, Series Editor

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Copyright © 2014 by Microsoft Corporation. All rights reserved.

No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Control Number: 2014956173
ISBN: 978-0-7356-9581-8

Printed and bound in the United States of America.

First Printing

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at <http://aka.ms/tellpress>.

This book is provided “as-is” and expresses the author’s views and opinions. The views, opinions and information expressed in this book, including URL and other Internet Web site references, may change without notice.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

Microsoft and the trademarks listed at <http://www.microsoft.com> on the “Trademarks” Web page are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

Acquisitions Editor: Karen Szall
Developmental Editor: Karen Szall
Copyeditor: Megan Smith-Creed
Cover Illustration: Twist Creative, Seattle

Contents

	Introduction	vii
Chapter 1	Why automation?	1
	Microsoft automation history	1
	What are Microsoft automation solutions?	2
	Where do Microsoft automation solutions fit?	6
	What do Microsoft automation solutions consist of?	7
Chapter 2	Understanding automation: Architectures	11
	On-premises architectures	11
	Database role	11
	Web service role	12
	Runbook worker role	13
	Service Management Automation PowerShell module	14
	Windows Azure Pack Service Management Portal	14
	Azure-based architectures	15
	Considerations for Azure Automation accounts	16
	Creating Azure Automation accounts	18
Chapter 3	Understanding automation: Interfaces	19
	Web services interface	19
	Using the web service in an application	22
	Using the web service in Windows PowerShell	24
	Windows PowerShell interfaces	26
	Service Management Automation PowerShell module	26
	Azure Automation PowerShell cmdlets	28
	Web portal interfaces	31

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

<http://aka.ms/tellpress>

Chapter 4	Implementing automation	37
	Using assets	37
	Connections	38
	Credentials	39
	Variables	39
	Schedules	40
	Integration Modules	41
	Using runbooks.....	41
	Authoring runbooks	42
	Exploring the runbook workflow structure	43
	Differences between standard Windows PowerShell	
	and workflows	44
	Defining parent and child runbook relationships	47
	Authoring Integration Modules.....	48
	Creating an Integration Module file	49
	Creating an Integration Module Manifest file	51
	Creating an Integration Module Metadata file	51
Chapter 5	Managing runbooks	53
	Runbook authoring tools.....	53
	Management portals	53
	Windows PowerShell ISE	54
	Visual Studio IDE	56
	Using source control.....	57
	Team Foundation Server	57
	Visual Studio Online	58
	Using logging	59
	Using error handling	61
	Troubleshooting runbooks	63
	Logs	63
	Configuration files	64
	Certificates	65
	Groups and service accounts	65

Chapter 6	Examples of automation scenarios	67
	Logging to a local file	67
	Logging to Table storage	69
	Creating an automation databus	78
	Continuous Integration Model	87

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

<http://aka.ms/tellpress>

This page intentionally left blank

Introduction

This book provides you with an introduction to the Microsoft automation solutions: Azure Automation and Service Management Automation. Throughout the chapters, the text explores these tools and how they can be used to meet the automation needs of your Microsoft Azure cloud solutions or your enterprise datacenter environments.

We provide considerations on the features of each solution, and how they can be architected to fit your needs. Next, the text explores the interfaces you will use to interact with the solutions, including the web-based portals, Windows PowerShell command-line interaction, and programmatic access via the web services. The text then covers how you implement and manage automation using runbooks, assets, and Integration Modules, along with how you can use a source control system to manage runbook content. Finally, some examples of automation scenarios are discussed, providing you with samples that can be used to speed development in your own solution.

The target audience for this book is IT pros charged with planning, building, or managing Microsoft Azure cloud solutions or enterprise datacenter environments, or anyone who is curious about taking their own steps toward using the Microsoft automation solutions. The information contained in this book has been gathered from field experience of the authors, working with customers to automate their solutions.

The content covered in this book is just the beginning for the Microsoft automation solutions, and the intention is to help you quickly learn the tools and lower any barrier for adoption of the Microsoft automation solutions.

To continue learning about Microsoft automation solutions, explore the following:

- System Center Orchestrator Engineering Blog:
<http://blogs.technet.com/b/orchestrator/>
- Microsoft Azure Automation Forum:
<http://social.msdn.microsoft.com/forums/azure/en-US/home?forum=azureautomation>
- Runbook Writing Wiki:
<http://social.technet.microsoft.com/wiki/contents/articles/26616.quick-tips-and-tricks-for-runbook-writing.aspx>
- TechNet Script Center Automation content:
http://gallery.technet.microsoft.com/scriptcenter/site/search?f%5B0%5D.Type=RootCategory&f%5B0%5D.Value=WindowsAzure&f%5B0%5D.Text=Windows%20Azure&f%5B1%5D.Type=SubCategory&f%5B1%5D.Value=WindowsAzure_automation&f%5B1%5D.Text=Automation

About the companion content

The companion content for this book can be downloaded from the following page:

<http://aka.ms/automation/files>

The companion content includes the following:

- The examples from Chapters 3 and 4
- The code listings and scenarios from Chapters 5 and 6

Acknowledgments

The authors would like to thank Sheldon Boxall and Yen Chiu Chin of Microsoft Services in Australia for their support in working with us to automate solutions for our customers and grow our collective knowledge of the Microsoft automation solutions, and for their reviews and input into the contents of this book.

The Series Editor would like to thank the following individuals at Microsoft who reviewed the outlines for the proposed titles in this series and provided helpful feedback to our authors:

- David Ziembicki
- Adam Fazio
- Robert Larson
- David Stoker
- Joel Yoker

Free ebooks from Microsoft Press

From technical overviews to in-depth information on special topics, the free ebooks from Microsoft Press cover a wide range of topics. These ebooks are available in PDF, EPUB, and Mobi for Kindle formats, ready for you to download at:

<http://aka.ms/mspressfree>

Check back often to see what is new!

Microsoft Virtual Academy

Build your knowledge of Microsoft technologies with free expert-led online training from Microsoft Virtual Academy (MVA). MVA offers a comprehensive library of videos, live events, and more to help you learn the latest technologies and prepare for certification exams. You'll find what you need here:

<http://www.microsoftvirtualacademy.com>

Errata, updates, & book support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

<http://aka.ms/automation/errata>

If you discover an error that is not already listed, please submit it to us at the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *<http://support.microsoft.com>*.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

<http://aka.ms/tellpress>

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *<http://twitter.com/MicrosoftPress>*.

This page intentionally left blank

Why automation?

This chapter introduces the Microsoft automation solutions by reviewing the history of automation and describing the new automation tools available, where they are used, and what they include. This introduction will provide you with the background necessary to dig deeper into the technology through the remainder of the book.

Microsoft automation history

If you have been around the IT industry for more than a few years, you remember when automation went by the generic name of “scripting.” Scripting has evolved over the years, starting in the Windows arena with languages such as DOS batch files, Perl, and Kix, moving on to VBScript and Jscript, and now including Windows PowerShell and System Center Orchestrator. Each advance has been a generational leap forward in capability and complexity. Now with the push to the cloud, scripting is again taking a leap forward, having become the center of operations and management of the hyper-scale datacenter.

Not to understate the usefulness of the early tools, but as the needs of IT pros have grown, the tools have advanced too. The need to perform complex, long-running, branched, and repeatable tasks is becoming common place. Windows PowerShell and System Center Orchestrator have made it possible to perform these functions without excessive scripting logic.

The limits of the technologies of old have forced the development of the technologies of today. With DOS scripting, IT professionals could control logic flow, manipulate text, and call out to perform tasks. Some impressive feats were achieved with DOS scripting, and even to this day some IT professionals fall back to this method for certain tasks. However, limitations came, reliance on external programs became cumbersome, and a sequentially processed scripting model constrained how a task could be done.

The inclusion of technology such as Windows Management Instrumentation (WMI), Active Directory Services Interface (ADSI), and direct object access into the Windows operating system in the early 1990s drove the shift to VBScript, Perl, Kix, and Jscript. The ability of VBScript and these other languages to directly access information, perform tasks, and manipulate external systems was a boon for scripters, who no longer needed to call out to external executables or code, with their inconsistent calling and returning routines. The ability to manage objects in scripts, rather than basic text alone, meant less time spent manipulating strings into the right formats from one output so that it could fit into another input. If a

function returned an integer, it could be stored in an object as an integer, and future access to the object could be treated as such.

With the onslaught of object-oriented programming (OOP) models (such as Microsoft .NET) during the 1990s, it was only a matter of time before the concepts and corresponding benefits of these models made their way into the scripting world.

Windows PowerShell set IT professionals on the path to a single unified model for automation. The Microsoft vision was grand: Windows PowerShell would be used as the command-line interface for administering the entire stack of Microsoft products. Some people scoffed at the idea. But before long the vision became clear to everyone, and now it is a reality. Everything IT professionals do to administer their Windows-based systems can be done through Windows PowerShell these days. It is ubiquitous across Microsoft products, from Windows to Exchange, Lync, SharePoint, System Center, and SQL. The early days of Windows PowerShell had rough moments, but gradually, features such as Remoting and Workflow trickled in, and IT professionals can now use Windows PowerShell to accomplish tasks that were impossible five years ago.

And then there was System Center Orchestrator. Microsoft purchased Opalis Software in December 2009, and with that purchase, the shift to a new age began to build momentum. As is usually the case with any technology purchase, it took Microsoft some time to truly transform the purchase into the cog that they wanted in the grand Microsoft technology machine. The Opalis name was changed to System Center Orchestrator, and through a couple of releases, its features grew and its integration with the broader ecosystem strengthened.

What are Microsoft automation solutions?

Microsoft automation solutions enable the IT professional to leverage a centralized automation engine to interact with any target system. Automation solutions function as a flexible and extensible service that can be leveraged for almost any task that can be imagined (see Figure 1-1).

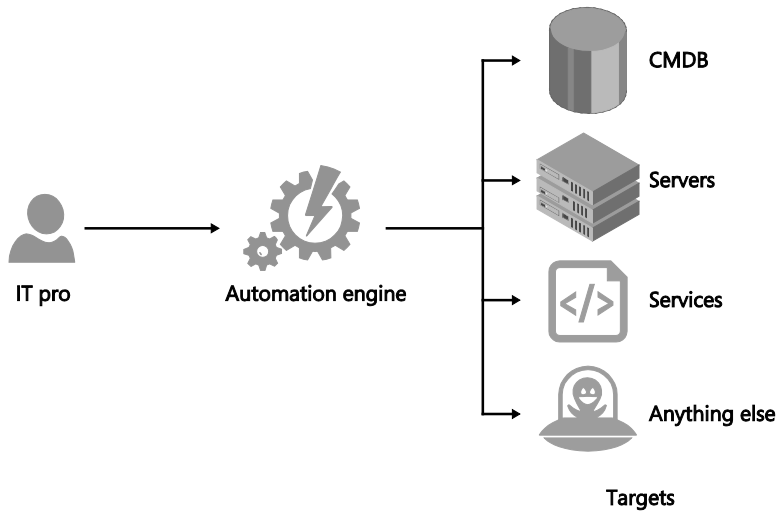


FIGURE 1-1 Automation overview

In this book, the term “Microsoft automation solutions” references both the on-premises tool System Center Service Management Automation and the cloud-based tool Microsoft Azure Automation. These are essentially the same, the only difference being the location from which they are delivered (see Figure 1-2).

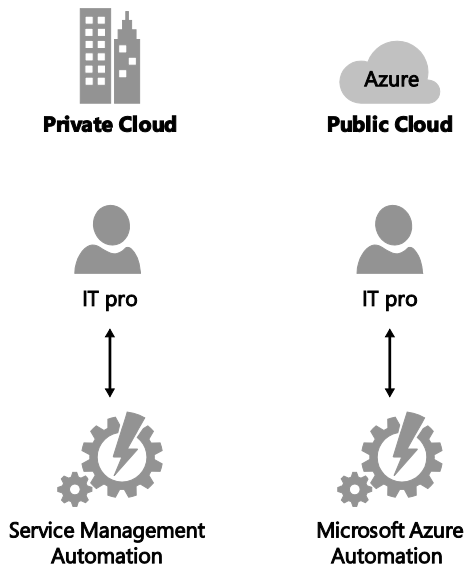


FIGURE 1-2 Automation deployment models

Microsoft automation solutions are the next evolution of orchestration capability, built on top of Windows PowerShell version 3 and its Workflow engine. The purchase and integration of Opalis into the Microsoft stack provided a step into the orchestration arena; however, Microsoft investments in Windows PowerShell garnered a more powerful and flexible alternative. Beginning with the release of System Center 2012 R2, Service Management Automation was released as a standalone product included in the System Center 2012 R2 Orchestrator media, subsequently followed up by the preview release of Microsoft Azure Automation.

These solutions provide a consistent interface and set of functionality to work with both private and public cloud infrastructures and let you build on your existing knowledge of Windows PowerShell to help you rapidly learn and adopt these tools.

As illustrated in Figure 1-3, Service Management Automation consists of the following pieces:

- **Database** A SQL database storing the Service Management Automation activities and objects.
- **Web service** The core entry point for administration of Service Management Automation, from which all other administrative interfaces are derived. It is based on the industry standard representational state transfer (REST) and open data protocol (OData) application programming interfaces (API), which allows third-party systems to interact with Service Management Automation via a well-known programming interface.
- **Runbook worker** The execution environment for activities defined in Service Management Automation. A runbook worker hosts one or more isolated execution environments to run and control runbooks, which are referred to as *sandboxes*. A sandbox is a managed, isolated environment where runbook activities can be activated without affecting other processes running on the same computer. This allows Service Management Automation to manage and monitor compute resources for runbooks and control or terminate runbooks as required.
- **Service Management Automation PowerShell module** A portable PowerShell module that can be deployed to endpoints to enable IT professionals to administer Service Management Automation remotely. The PowerShell module interacts with the Service Management Automation web service for access to the Service Management Automation database.
- **(Optional) Windows Azure Pack Service Management Portal** A web-based graphical user interface (GUI) for authoring, managing, monitoring, and running Service Management Automation activities. The Service Management Portal interacts with the Service Management Automation web service for access to the Service Management Automation database.

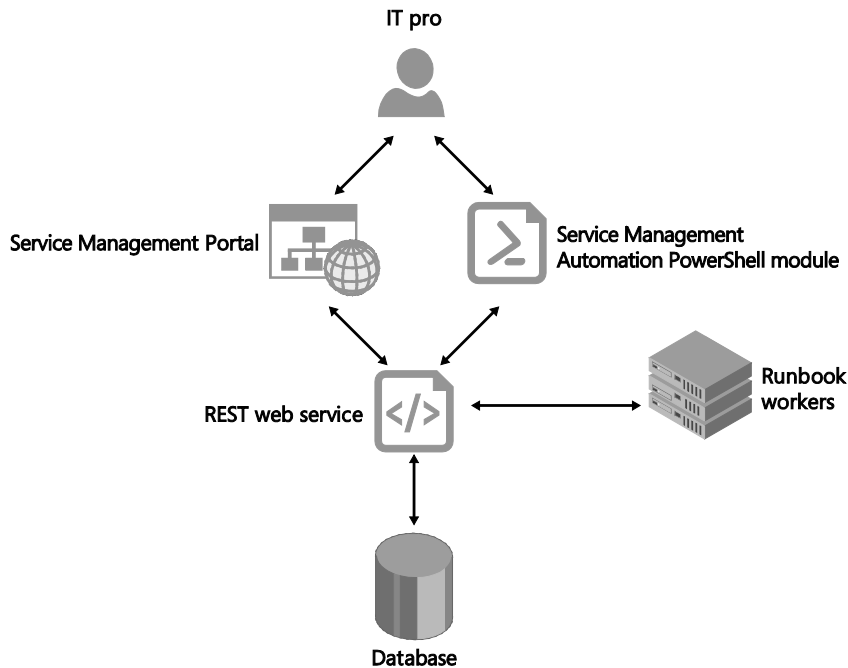


FIGURE 1-3 Service Management Automation model

Each of these pieces can be combined on a single server for a basic deployment or distributed to provide greater scalability and availability. For details, refer to Chapter 2, “Understanding automation: architectures.”

As illustrated in Figure 1-4, the Microsoft Azure Automation model is much simpler because it is provided as a service. Deployment of the service in Microsoft Azure is as easy as creating an automation account under an existing Microsoft Azure subscription. The instance is created and ready to use in minutes. This makes automation accessible to anyone using the Microsoft Azure public cloud services, including those with an MSDN subscription. If you have none of these, you can sign up for a free trial of Microsoft Azure at <http://azure.microsoft.com/>.

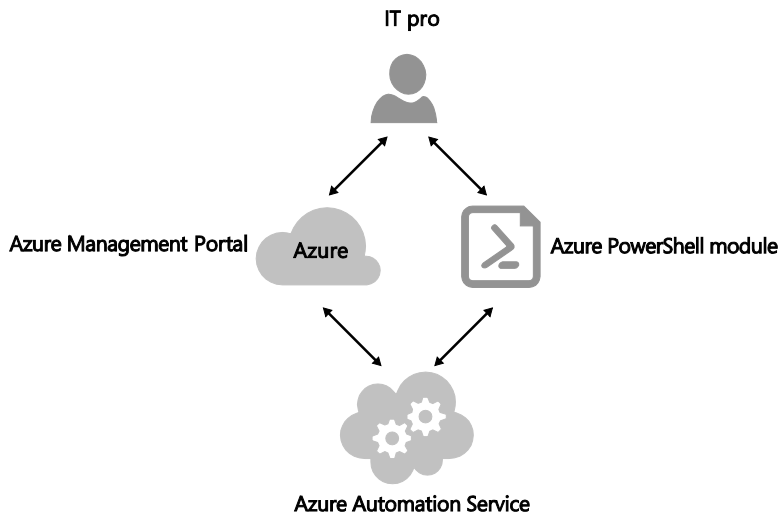


FIGURE 1-4 Microsoft Azure Automation model

Where do Microsoft automation solutions fit?

Service Management Automation provides the engine for powering automation activities for a private cloud environment. It provides a bridge between the System Center suite and Windows Azure Pack. This bridge makes it possible for you to use scheduled, ad-hoc, or event-based triggers for performing automated administrative tasks. Service Management Automation can be used in conjunction with other tools, such as those in the System Center suite or any product that can be interacted with via Windows PowerShell or a custom Windows PowerShell module. For example, you can use Service Management Automation to do the following:

- Perform an action upon creation of a virtual machine, such as updating a configuration management database (CMDB) or ticketing system (such as System Center Service Manager) or sending a notification email to the virtual machine owner on successful completion of the activity.
- Do scheduled maintenance, such as coordinated patch management of hosting infrastructure or workload balancing across host clusters.
- Manually trigger a discovery or auditing activity as needed.

Likewise, Microsoft Azure Automation powers automation activities for the Microsoft Azure public cloud environment. Under the public cloud models, your interaction is not with the underlying infrastructure; instead, it is with the cloud assets and workloads themselves. For example, you can use Microsoft Azure Automation to do the following:

- Create and tear down development and test environments.
- Monitor and manage workload usage, such as shutting down unused virtual machines to conserve spending and scaling out workloads and assets as required to meet demand.
- Manage deployment of application workloads through various lifecycles in conjunction with developer tools and processes.

These tools by no means negate the need for you to continue to use Windows PowerShell or the native tools provided for administering systems. Automation gives you the ability to predefine workflows for common activities and have them handled automatically so you can focus your attention on adding value in other areas.

Functionally, the Microsoft automation tools overlap with the existing capability of System Center Orchestrator, and you may wonder which one you should use. It depends. You must consider the solution that is being built or the problem being solved, any investments already made in automation, and the skills of those involved in building a solution. Consider the following guidelines when deciding which tool to use:

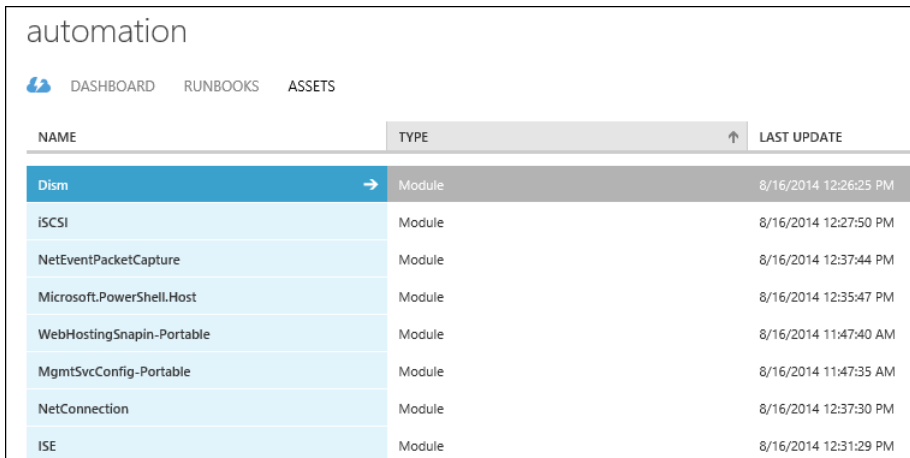
- If you are targeting traditional datacenter automation activities and have an existing investment in System Center Orchestrator and the graphical authoring environment, continue to use this path.
- If you are building out a private cloud solution, have skilled resources, and are comfortable with Windows PowerShell, Service Management Automation is the path you should take.
- If you are building out a solution in the Microsoft Azure public cloud, Microsoft Azure Automation is a more logical path for starting your automation journey.

What do Microsoft automation solutions consist of?


The Microsoft automation solutions consist of several object types that collectively provide the automation service, including the following:

- **Assets** The static objects in the service that provide configurations to automation activities, allowing globally defined settings to be leveraged across the service. Asset types used in Microsoft automation solutions include:
 - **Connections** Used to store details of a service that is used by multiple automation activities; include settings such as a server name, user name, and password. Connections are useful for connecting to services such as System Center Virtual Machine Manager (VMM) or a Microsoft Azure subscription.

- **Credentials** Used to store two types of credentials: Windows PowerShell credentials or certificate credentials. Credentials allow automation activities to refer to a repository of securely stored credentials that can be used for accessing systems and services as required, without the need for user input. Because each activity runs under the context of the runbook worker services, credentials or connections are required for privilege escalation or any interaction happening off the runbook worker server.
- **Variables** Used to store key-value pair variables to be consumed in automation activities. Variables can be defined with the following types: string, integer, boolean, datetime, and not defined. Variables can be optionally encrypted if the values are determined to be sensitive, for example, if storing license keys or passwords.
- **Schedules** Used to define a regular schedule to which an activity can be linked, for example, to initiate a runbook at the start or end of business each day.
- **Modules** Used to store Windows PowerShell modules for use by automation activities. If a module is stored as an asset (see Figure 1-5) and leveraged in a runbook workflow, it will be delivered to the runbook worker. This ensures that modules can be centrally managed rather than manually deployed to each runbook worker server in a deployment.
- **Runbooks** Windows PowerShell workflow objects used to define a single or logical set of activities to be run (see Figure 1-6). On initiation, a runbook starts as a job on a runbook worker in a sandbox environment. Each runbook represents a Windows PowerShell workflow script, which inherits all Windows PowerShell workflow capabilities, including input and out parameters, checkpoint and suspend/resume, parent-child relationships, and extensibility through Windows PowerShell modules.



automation

 DASHBOARD RUNBOOKS ASSETS

NAME	TYPE	LAST UPDATE
Dism →	Module	8/16/2014 12:26:25 PM
iSCSI	Module	8/16/2014 12:27:50 PM
NetEventPacketCapture	Module	8/16/2014 12:37:44 PM
Microsoft.PowerShell.Host	Module	8/16/2014 12:35:47 PM
WebHostingSnapiin-Portable	Module	8/16/2014 11:47:40 AM
MgmtSvcConfig-Portable	Module	8/16/2014 11:47:35 AM
NetConnection	Module	8/16/2014 12:37:30 PM
ISE	Module	8/16/2014 12:31:29 PM

FIGURE 1-5 Asset user experience

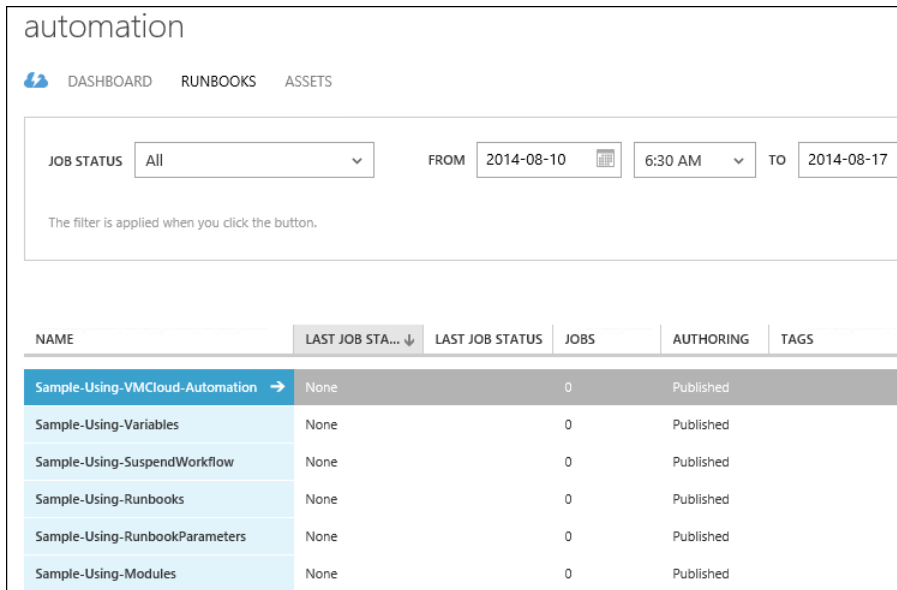


FIGURE 1-6 Runbook user experience

Each of the objects defined in this section are represented in the Service Management Automation database or the Microsoft Azure Automation database. For more details of managing assets and runbooks, refer to Chapter 4, “Implementing automation.”

This page intentionally left blank

Understanding automation: Architectures

This chapter provides an overview of the architectural options for deploying Service Management Automation on-premises and for using Microsoft Azure Automation in the Microsoft Azure public cloud.

On-premises architectures

The on-premises Microsoft automation solution, Service Management Automation, has a quite flexible architecture, allowing for consolidated or scaled-out deployment scenarios. This section explores roles, scaling options, and deployment considerations for each Service Management Automation role and how these roles can be deployed in your organization:

The recommended architecture for a Service Management Automation deployment includes the following:

- A dedicated SQL server instance for the database role
- Three servers co-hosting the web service and runbook worker roles
- Load balancing of the web service role
- Virtual machine deployment for ease of scaling and management

For hardware and software requirements, refer to the Service Management Automation TechNet article at <http://technet.microsoft.com/en-us/library/dn469256.aspx>.

Database role

The role of the Service Management Automation database is to provide the central configuration store for all objects, including runbooks, assets, jobs, and logging data. No data is stored on other Service Management Automation roles, with the exception of Windows PowerShell modules on runbook workers if they are not imported as assets in Service Management Automation (which is explored in Chapter 4, “Implementing automation”).

The database must run on Microsoft SQL Server 2012 at minimum and can be deployed to a standalone SQL Server server, a SQL Server failover cluster, or a SQL Server AlwaysOn Availability Group (see Figure 2-1). It is worth noting the Service Management Automation

database is a *contained database*, meaning it can be included in an availability group without the need to manually replicate additional objects.

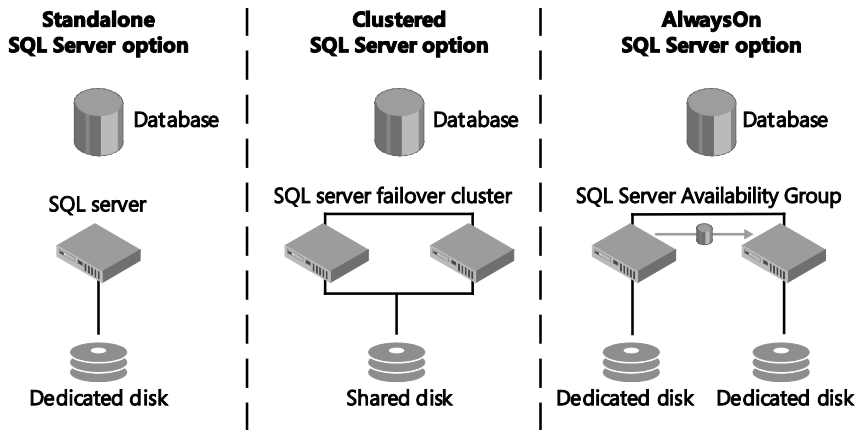


FIGURE 2-1 Service Management Automation database server options

Generally, the use of a dedicated SQL server instance is recommended for the Service Management Automation database to ensure it can be allocated appropriate memory and CPU resources and can be managed independently of other databases.

The Service Management Automation database role can exist only once for a single deployment; therefore, its scaling ability is limited to scaling-up resources (memory and CPU cores). As with any SQL database, it is best to separate the application database, tempdb, and log files onto disks to optimize performance.

The Service Management Automation database is created during the installation of the Service Management Automation web service. Prior to this installation, the target SQL server instance must be installed and configured. When installing the database, a service account is used to grant the web service access to the database, which is also used later for integration with the Windows Azure Pack Service Management Portal and for the addition of web service instances when scaling out. Be sure to follow your standards for creating and managing this service account since you will need it to configure your Service Management Automation deployment later.

Web service role

The role of the web service is to provide a control point for interaction with Service Management Automation from user and system interfaces when accessing the database. The web service is essential for the operation of your Service Management Automation deployment; therefore, planning how it is deployed with scalability and resiliency in mind is critically important.

The Service Management Automation web service is stateless, so it can be scaled up with additional compute resources (such as CPU and memory), as well as scaled out with additional

server instances and the use of a load balancing service, such as Windows Network Load Balancing or a third-party load balancer.

Product guidance for Service Management Automation web service roles specifies at least two cores and 4 gigabytes (GB) of RAM, even when combined with the runbook worker role. However, this will not suffice for most deployments, so it is better to start with 8 GB of RAM.

By default, the web service creates a self-signed certificate and binds the HTTPS protocol to port 9090. When registering Service Management Automation with the Service Management Portal, the URL of the web service endpoint is required, such as *https://mysmaserver:9090/*. For a lab or small-scale deployment that does not require high availability, self-signed certificates and a standard server hostname URL may suffice. If, however, you are starting small and intend to scale out the web service role at some point, careful planning early will help you avoid heavy reconfiguration later. Consider the following when planning the deployment of the web service:

- Use a generic DNS hostname for your web service URL, such as *sma.contoso.com*.
- Provision a web server certificate for your web service role using your generic DNS hostname. Be sure to mark the certificate's private key as exportable so you can install it on additional web service role servers. As a general best practice with certificates, be sure to keep the certificate's private key safe.
- If you plan to use a third-party load balancer, use it to front your single server deployment initially so scaling out will not impact the service.

The web service is based on industry standard representational state transfer (REST) and open data protocol (OData) application programming interfaces (API), is required for each deployment, and must be deployed on Internet Information Services (IIS) 7.5. This makes it easy to consume the web service from external systems, such as when integrating with service desk or monitoring systems. Web services capabilities are covered in more detail in Chapter 3, "Understanding automation: Interfaces."

Runbook worker role

The role of the Service Management Automation runbook worker is to host the runtime environment of runbooks. From the runbook worker server, a runbook can run the defined workflows with access to any required assets and modules that are stored in the Service Management Automation database.

Unlike the web service role, the runbook worker role is not stateless. In a Service Management Automation deployment, a *runbook worker deployment* is created. A runbook worker deployment defines a static list of runbook worker servers, which are allocated a portion of the runbook job queue for processing. The runbook job queue is divided by the number of available runbook workers in a deployment and each is given a range of jobs it can process between a low and high key value.

For example, in a single server deployment, the runbook worker job queue is allocated to one server, and its key range is 0 to 2147483647. However, if two runbook workers are available in a deployment, the job queue is split, giving the first runbook worker a key range of

0 to 1073741823 and the second runbook worker a key range of 1073741824 to 2147483647. In the static job queue model, if a runbook worker becomes unavailable, any jobs destined for that server do not process until that server (or a new server with the same name) is brought online. For this reason, any change to runbook workers in your Service Management Automation deployment, including scaling up for capacity or losing a runbook worker due to failure, must be handled manually.

See also For more details of a runbook work deployment, including guidance for changing an existing deployment, refer to [http://technet.microsoft.com/en-us/library/dn530618\(v=sc.20\).aspx](http://technet.microsoft.com/en-us/library/dn530618(v=sc.20).aspx).

As introduced in Chapter 1, “Why automation?” the runbook worker uses a sandbox runtime model, which contains multiple discrete units of transactions running in isolated containers. Each sandbox is assigned resources and, by default, is configured to handle 30 concurrent jobs. A runbook worker can instantiate up to four sandboxes by default (since the default maximum number of running jobs per runbook worker is 120). However, undesirable performance characteristics may be exhibited well before reaching these maximums, such as compute resource exhaustion and an increase in transactional processing times.

Service Management Automation PowerShell module

The Service Management Automation PowerShell module can be deployed to clients and servers to allow remote interaction with the web service. By default, the PowerShell module is installed as part of the web service, but it can also be installed on any computer by using the installation media.

To view the available cmdlets in the Microsoft.SystemCenter.ServiceManagementAutomation module, run the following command from a Windows PowerShell session on your web service server:

```
Get-Command -Module Microsoft.SystemCenter.ServiceManagementAutomation | select  
Name, Parameters
```

The PowerShell module is explored further in Chapter 3.

If you intend to explore the PowerShell module before reading on, note that each cmdlet will require you to provide a `-WebServiceEndpoint` parameter, for which you should provide the web service URL without the port number. For example:

```
Get-SmaRunbook -WebServiceEndpoint "https://sma.contoso.com"
```

Windows Azure Pack Service Management Portal

The Windows Azure Pack Service Management Portal is the primary interface to a Microsoft private cloud deployment, providing you with a web-based interface to manage your resources. The Service Management Portal also functions as the primary interface for Service Management Automation, allowing you to use the web-based management and authoring experience rather than relying on the PowerShell module alone.

The deployment and scaling options for Windows Azure Pack are outside the scope of this book; however, for detailed guidance, refer to the published documentation at <http://technet.microsoft.com/en-us/library/dn296432.aspx>.

When a Service Management Portal deployment is available, integrating Service Management Automation is straightforward. Log in to the Service Management Portal and perform the following steps:

1. Scroll down in the menu pane and click Automation.
2. In the Automation Quick Start workspace, click the Register Service Management Automation Endpoint link.
3. Enter the service URL of the Service Management Automation web service and the user name and password of the account used when installing the Service Management Automation web service and database, and then click OK.

When the Service Management Automation endpoint is successfully registered, you can navigate to the Automation workspace to view the Dashboard, Runbooks, and Assets views (see Figure 2-2).

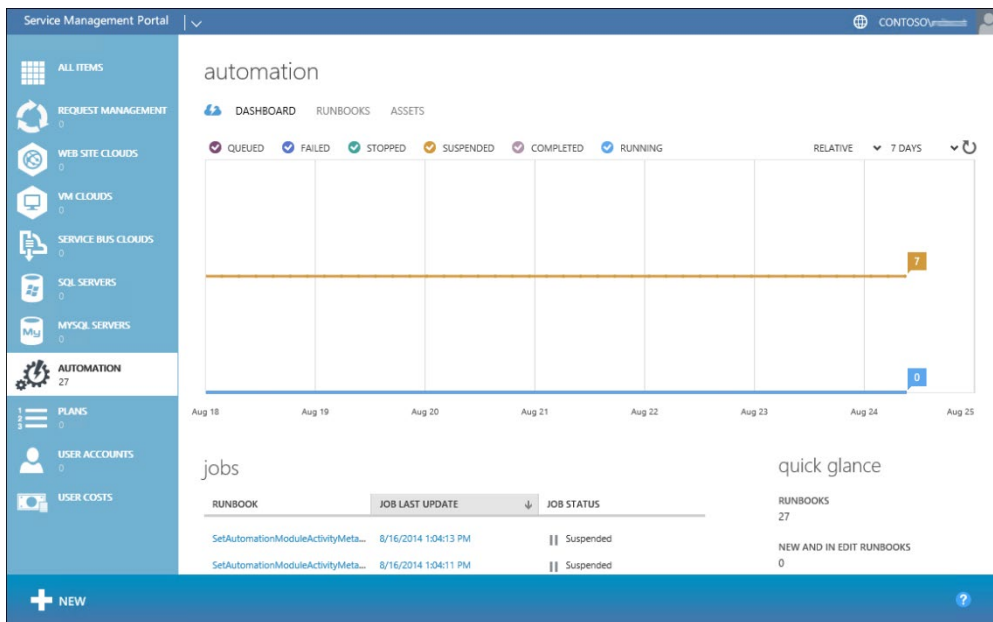


FIGURE 2-2 Service Management Portal Automation workspace

Azure-based architectures

The Azure-based architecture for Microsoft Azure Automation is provided as a service, so you do not need to plan and deploy the underlying infrastructure to use the service.

NOTE If you are not familiar with Microsoft Azure, you can sign up for a free trial of the software at <http://azure.microsoft.com/>.

Before you use Azure Automation, it is useful to understand how Azure services and features are organized (see Figure 2-3). When you sign up for Microsoft Azure, you establish an Azure account that acts as a management and billing container. Within an Azure account, you can create and manage one or more Azure subscriptions, which can be used to separate administrator access or manage multiple lifecycles (e.g., development, test, and production). Each Azure subscription can host any Azure service, including the Azure Automation service.

When using the Azure Automation service, you create one or more Automation accounts under your Azure subscription (up to 30, in fact). Each Automation account represents a separate instance of Azure Automation under your subscription, allowing you to create and manage runbooks under multiple Automation accounts for development and testing, for load distribution, or for distributing services across Microsoft Azure regions.

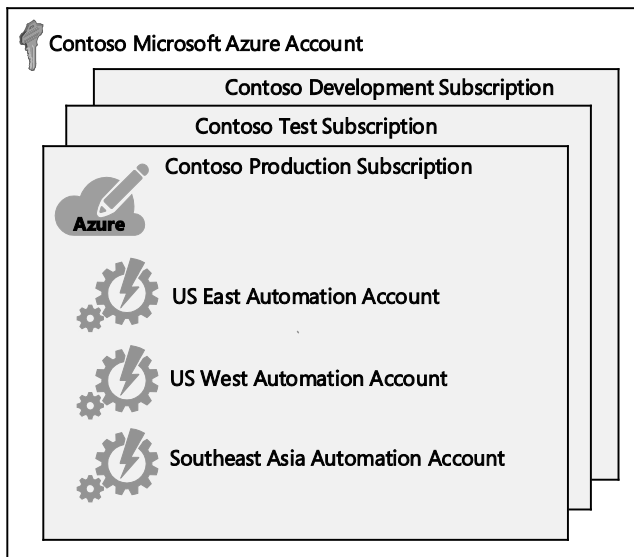


FIGURE 2-3 Microsoft Azure services and features model

Considerations for Azure Automation accounts

When planning your Azure Automation deployment, you should consider the following concepts to ensure your deployment is flexible enough to service your needs but simple to manage.

Administrative control

In the current Microsoft Azure management model, an Azure subscription is the lowest level of administrative control. Any co-administrator of an Azure subscription can fully manage the services within that subscription. If multiple Azure Automation accounts are created in a subscription, all co-administrators will have full access to each of them. For this reason, if you need to limit access for administrators to a subset of Microsoft Azure resources, including Azure Automation services, it may be necessary to create multiple Azure subscriptions for an Azure account.

Load distribution

When you are familiar with Azure Automation and you're using it extensively to automate your Azure services, you may consider scaling out your Azure Automation accounts to distribute load. Unlike Service Management Automation where the runbook worker roles configuration can be tuned (e.g., job runtime and sandbox resource limits settings), the Azure Automation service provides statically configured runbook worker services that cannot be tuned. To overcome this limitation, you can create additional Azure Automation accounts and distribute your runbooks and jobs across them.

For example, this may be useful when you need to use a very long-running runbook. A runbook that runs for more than 30 minutes will be suspended to allow other runbook jobs on the same runbook worker to process. After all other runbooks have processed, the original runbook will resume from its last checkpoint. This is known as the *Fair Use* feature in Azure Automation and ensures one or more runbooks do not dominate job time on a runbook worker. In this scenario, using multiple Azure Automation accounts allows more long-running runbooks to run in parallel. They will still suspend and resume, but the time between will be minimized (see Figure 2-4). Given this feature, it is important to understand how runbooks use checkpointing capabilities in the Windows PowerShell workflow to handle suspend and resume actions. Chapter 4 explores checkpointing in more depth.

NOTE The Azure Automation service billing model is based on runbook processing time in minutes, so long-running runbooks that have little actual activity should be avoided in favor of smaller, more modular runbooks that run on schedules.

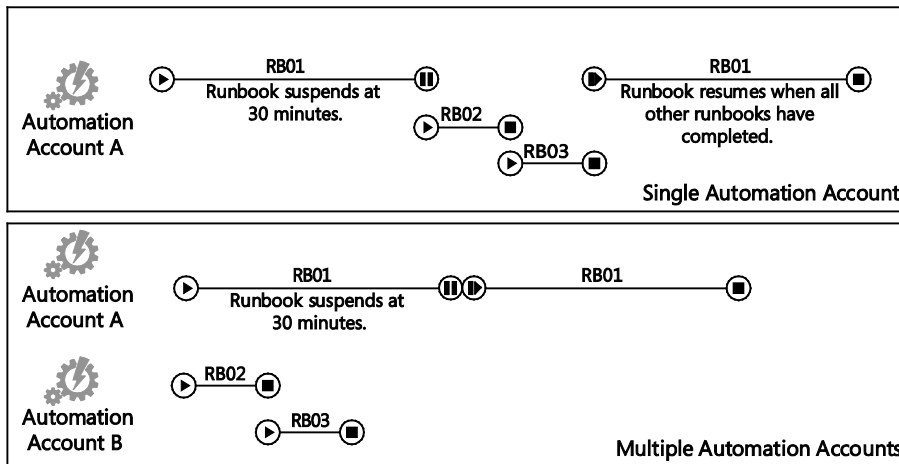


FIGURE 2-4 Comparison of load distribution models

Regional distribution

In solutions built with Microsoft Azure, deployed services, such as websites, virtual machines, and SQL databases, are often associated with a common Azure region (e.g., U.S. West or Southeast Asia). Regional deployment ensures these services are optimized for performance due to the short physical distance between the infrastructures they are hosted on.

This consideration applies to Azure Automation as well. When creating an Azure Automation account, you are required to provide an Azure region. You should place the new account in the region where your runbooks will interact with other Azure services. If you have Azure services distributed across multiple regions, consider creating multiple Azure Automation accounts.

NOTE At the time of writing this book, the Azure Automation service is in preview and is provided only in the East U.S. region. On release, the service will be available in more regions.

Creating Azure Automation accounts

To create an Azure Automation account in your Azure subscription, you will need an account name and the target region.

Log in to the Microsoft Azure Management Portal and perform the following steps:

1. Scroll down in the menu pane and click Automation.
2. In the bottom menu pane, click Create.
3. In the Create An Automation Account dialog box, provide the account name, select the region for the account, and click OK.

When you complete the procedure, the Azure Automation account is created, and an instance is provisioned in the region you specified. In a few minutes, the instance is available for use.

Understanding automation: Interfaces

This chapter introduces you to the interfaces through which you can interact with the Microsoft automation solutions. This chapter will cover:

- How the web services work
- How to administer automation solutions using Windows PowerShell
- How to navigate the web portals for the best authoring and management experience

Web services interface

As introduced in Chapter 2, "Understanding automation: Architectures," the web service plays a critical role in managing the Microsoft automation solutions. The web service is exposed in a Service Management Automation system, allowing you to interact with the system and build extended solutions using it. When the Azure Automation solution is provided as a service, however, the web service is not exposed, allowing management only through the Azure PowerShell module and the Microsoft Azure Management Portal interfaces. For this reason, the remainder of this section on the web services interface will focus on the Service Management Automation web service.

The Service Management Automation web service is based on the open data protocol (OData) web service model, meaning requests delivered to the web service are returned with XML responses. This allows external services to interact with the web service and receive standardized responses that can be easily interpreted.

See also For details of the OData standard, refer to <http://msdn.microsoft.com/en-us/library/ff478141.aspx>.

A web service request requires the following information:

- Request type (Get, Put, and so on)
- Uniform resource identifier (URI)

- Authentication type (Basic or Windows)
- HTTP headers
- Request body when posting data through the web service

Request types sent to the web service include:

- **Get** Retrieves a collection or single entry record as requested in the URI
- **Post** Creates a new entry based on the body of the request
- **Put** Edits an existing entry based on the body of the request
- **Delete** Deletes an entry as requested in the URI

The hierarchy of the web service is broken into collections and entries. A collection is similar to a folder and contains one or more entries below it. An entry is a single object similar to a file. The URI that you use navigates the hierarchical structure of the web service. In the case of the Service Management Automation web service, the base of the hierarchy (or service root URI) is represented by the following URI:

<https://sma.contoso.com:9090/00000000-0000-0000-0000-000000000000/>

From the service root URI, you can navigate through the hierarchy by extending the URI with the appropriate path based on the following structure:

`[Service Root URI]/[Resource Path]?[Query Options]`

See also For more details of URI syntax for the Service Management Automation web service, refer to <http://msdn.microsoft.com/en-us/library/dn720250.aspx>.

For example, you can use basic navigation to do the following:

- To navigate into the runbooks collection, use this URI (see Figure 3-1):
<https://sma.contoso.com:9090/00000000-0000-0000-0000-000000000000/Runbooks>
- To navigate to a module entry directly, use this URI:
[https://sma.contoso.com:9090/00000000-0000-0000-0000-000000000000/Modules\(guid'e83b4d53-4028-4ccc-9063-72a1047969fb'\)](https://sma.contoso.com:9090/00000000-0000-0000-0000-000000000000/Modules(guid'e83b4d53-4028-4ccc-9063-72a1047969fb'))

```
<?xml version="1.0" encoding="utf-8" ?>  
- <service xml:base="https://sma01.contoso.com:9090/000000"  
- <workspace>  
    <atom:title>Default</atom:title>  
+ <collection href="#Jobs">  
+ <collection href="#Runbooks">  
+ <collection href="#RunbookVersions">  
+ <collection href="#JobContexts">  
+ <collection href="#JobParameters">  
+ <collection href="#Schedules">  
+ <collection href="#Modules">  
+ <collection href="#ConnectionFields">  
+ <collection href="#ConnectionFactoryValues">  
+ <collection href="#Connections">  
+ <collection href="#ConnectionTypes">  
+ <collection href="#Variables">  
+ <collection href="#Credentials">  
+ <collection href="#Certificates">  
+ <collection href="#Activities">  
+ <collection href="#ActivityParameterSets">  
+ <collection href="#ActivityParameters">  
+ <collection href="#ActivityOutputTypes">  
+ <collection href="#RunbookParameters">  
+ <collection href="#Statistics">  
+ <collection href="#AdminConfigurations">  
+ <collection href="#Deployment">  
</workspace>  
</service>
```

FIGURE 3-1 Example web service response for the service root URI

The OData web service framework allows for complex request types, such as using queries or paging. For example, to find a module by name, use this URI:

```
https://sma.contoso.com:9090/00000000-0000-0000-0000-000000000000/  
Modules?filter=Name eq 'OperationsManager-Portable'
```

The Authentication type used when making a request to the web service will be either Basic or Windows, depending on where your request is coming from and the authentication mechanisms it supports, along with your security preferences. By default, the Service Management Automation web service is enabled for both Basic and Windows authentication; however, the requirements of your solution or organization may dictate that a specific mechanism is used. Whichever authentication method is used, the authenticating user must then be authorized to use the web service through membership in the smaAdminGroup local group on the web service servers.

When a request is sent to the web service, a standard set of HTTP headers is used for requests and responses.

See also For details of these headers, refer to <http://msdn.microsoft.com/en-us/library/dn688268.aspx>.

Finally, when a Post or Put request type is sent to the web service, additional data is delivered in the request via the body of the request. The body of the request is an XML document.

See also For details of collection and entry body syntax, refer to the Service Management Automation Web Service Reference at <http://msdn.microsoft.com/en-us/library/dn688271.aspx>.

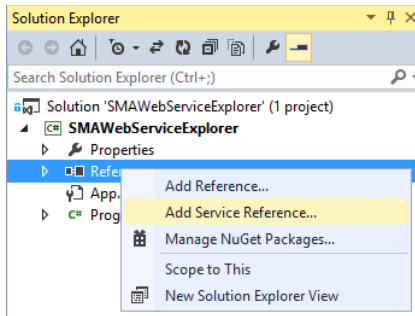
Bringing this all together, you can construct a request to the web service and receive a response. Following are two examples of using the web service, the first using a basic command-line tool created in Visual Studio using C#, and the second using Windows PowerShell.

Using the web service in an application

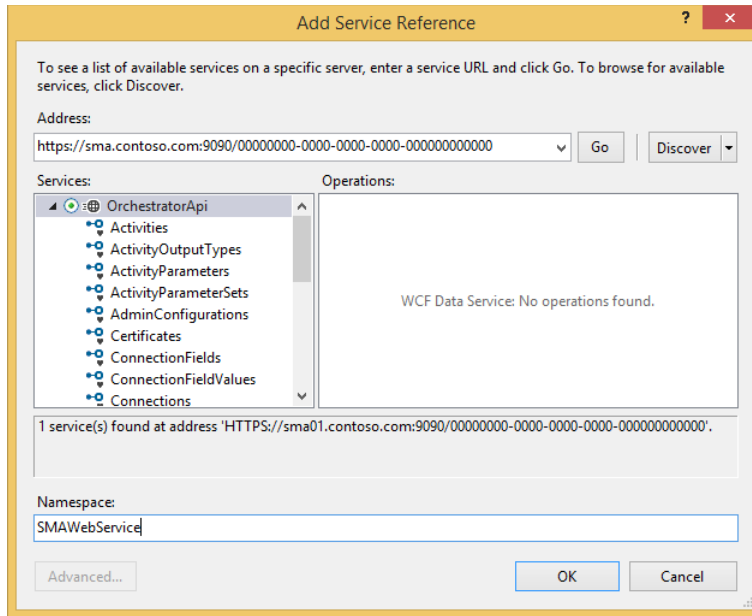
This example shows how to create a basic command-line application that uses the web service to explore the collections and entries available through the web service.

From a computer running Visual Studio, perform the following steps:

1. Open Visual Studio, and, from the File menu, click New, and then click Project.
2. In the New Project dialog box, using the tree-view, expand the Visual C# view. Locate and select the Console Application template in the center list pane.
3. Enter the name of the application, for example type **SMAWebServiceExplorer**.
4. From Solution Explorer, right-click References, and select Add Service Reference.



5. In the Add Service Reference dialog box, type the URL of the web service, for example type **<https://sma.contoso.com:9090/00000000-0000-0000-0000-000000000000>**, and then provide a name for the namespace, for example type **SMAWebService**.



6. The Program.cs file should already be open in the main workspace. If it is not, locate and open it from Solution Explorer.

7. Add additional resources to the project by appending the following lines after the default using references:

```
using System.Net;
using System.Data.Services.Client;
using SMAWebServiceExplorer.SMAWebService;
```

8. In the Main code block, insert the following:

```
// Create connection to the web service
OrchestratorApi smaws = new OrchestratorApi(new Uri(@"https://sma01.contoso.com:
9090/00000000-0000-0000-0000-000000000000/"));

// Authenticate with currently logged on credentials
((DataServiceContext)smaws).Credentials = CredentialCache.DefaultCredentials;

try
{
    // Request all runbooks
    var rbs = smaws.Runbooks;

    // Enumerate runbooks and display the name of each
    foreach (Runbook rb in rbs)
        Console.WriteLine("---{0}", rb.RunbookName);
}
```

```

// Prompt user to exit
Console.WriteLine("Press any key to exit...");
Console.ReadKey(true);
}
catch (NullReferenceException ex)
{
    // Catch exception and notify user
    throw new ApplicationException("An error occurred during execution.", ex);
}

```

9. To test the program, click Start in the Visual Studio toolbar or press F5. The program compiles and runs, and you are presented with a list of the available runbooks on your Service Management Automation deployment (see Figure 3-2).

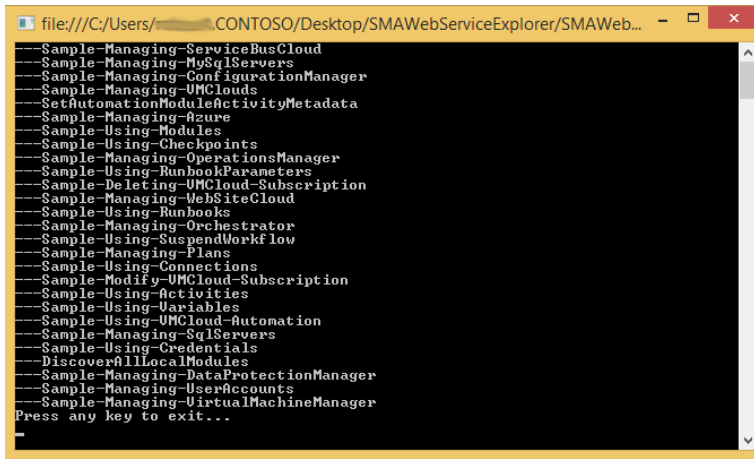


FIGURE 3-2 Output of the sample command-line program

Using the web service in Windows PowerShell

Since the web service is a representational state transfer (REST) Odata web service, you can leverage it using the built-in `Invoke-RestMethod` cmdlet in Windows PowerShell. This may be useful when you want to interact with Service Management Automation from a client or server but are unsure if the Windows PowerShell module is available on the end point, or if you are unwilling to install it. This example browses the available collections via the web service and explores a runbook and a variable asset.

Browsing the web service service root URI

```
PS> $smaRoot = Invoke-RestMethod -Method Get -Uri https://sma.contoso.com:9090/00000000-0000-0000-0000-000000000000 -UseDefaultCredentials
```

```
PS> $smaRoot.service.workspace.collection
```

href	title
----	-----
Jobs	Jobs
Runbooks	Runbooks
RunbookVersions	RunbookVersions
JobContexts	JobContexts
JobParameters	JobParameters
Schedules	Schedules
Modules	Modules
ConnectionFields	ConnectionFields
ConnectionFieldValues	ConnectionFieldValues
Connections	Connections
ConnectionTypes	ConnectionTypes
Variables	Variables
Credentials	Credentials
Certificates	Certificates
Activities	Activities
ActivityParameterSets	ActivityParameterSets
ActivityParameters	ActivityParameters
ActivityOutputTypes	ActivityOutputTypes
RunbookParameters	RunbookParameters
Statistics	Statistics
AdminConfigurations	AdminConfigurations
Deployment	Deployment

Exploring a runbook

```
PS> $smaRunbooks = Invoke-RestMethod -Method Get -Uri https://sma.contoso.com:9090/00000000-0000-0000-0000-000000000000/Runbooks -UseDefaultCredentials
```

```
PS> $smaRunbooks.[0].content.properties
```

TenantID	: TenantID
RunbookID	: RunbookID
RunbookName	: Sample-Managing-ServiceBusCloud
CreationTime	: CreationTime
LastModifiedTime	: LastModifiedTime
LastModifiedBy	: LastModifiedBy
Description	: Description
IsApiOnly	: IsApiOnly

```

IsGlobal                : IsGlobal
PublishedRunbookVersionID : PublishedRunbookVersionID
DraftRunbookVersionID    : DraftRunbookVersionID
Tags                    : Tags

LogDebug                : LogDebug
LogVerbose              : LogVerbose
LogProgress             : LogProgress

```

Exploring a variable asset

```

PS> $smaVariable = Invoke-RestMethod -Method Get -Uri https://sma.contoso.com:9090/
00000000-0000-0000-0000-000000000000/Variables -UseDefaultCredentials

```

```

PS> $smaVariable.[0].content.properties

```

```

VariableID      : VariableID
TenantID        : TenantID
Name            : Company
Value           : "Contoso"
Description     : Description
IsEncrypted     : IsEncrypted
CreationTime    : CreationTime
LastModifiedTime : LastModifiedTime

```

Windows PowerShell interfaces

The Windows PowerShell interfaces for Service Management Automation and Azure Automation, while similar in some ways, are provided separately and have different levels of interaction. In this section you will learn about the Windows PowerShell interface for each solution.

Service Management Automation PowerShell module

The Service Management Automation Powershell module can be deployed on any system with Windows PowerShell version 4.0. This module allows you to manage and interact with the Service Management Automation service by leveraging the web service described previously. The module includes the following cmdlets.

```
PS > get-command -Module *ServiceManagementAutomation*
```

CommandType	Name
-----	----
Cmdlet	Edit-SmaRunbook
Cmdlet	Get-SmaAdminConfiguration
Cmdlet	Get-SmaCertificate
Cmdlet	Get-SmaConnection
Cmdlet	Get-SmaConnectionField
Cmdlet	Get-SmaConnectionType
Cmdlet	Get-SmaCredential
Cmdlet	Get-SmaJob
Cmdlet	Get-SmaJobOutput
Cmdlet	Get-SmaLicense
Cmdlet	Get-SmaModule
Cmdlet	Get-SmaRunbook
Cmdlet	Get-SmaRunbookDefinition
Cmdlet	Get-SmaRunbookWorkerDeployment
Cmdlet	Get-SmaSchedule
Cmdlet	Get-SmaVariable
Cmdlet	Import-SmaModule
Cmdlet	Import-SmaRunbook
Cmdlet	New-SmaConnection
Cmdlet	New-SmaRunbookWorkerDeployment
Cmdlet	Publish-SmaRunbook
Cmdlet	Remove-SmaCertificate
Cmdlet	Remove-SmaConnection
Cmdlet	Remove-SmaCredential
Cmdlet	Remove-SmaModule
Cmdlet	Remove-SmaRunbook
Cmdlet	Remove-SmaSchedule
Cmdlet	Remove-SmaVariable
Cmdlet	Resume-SmaJob
Cmdlet	Set-SmaAdminConfiguration
Cmdlet	Set-SmaCertificate
Cmdlet	Set-SmaConnectionFieldValue
Cmdlet	Set-SmaCredential
Cmdlet	Set-SmaLicense
Cmdlet	Set-SmaRunbookConfiguration
Cmdlet	Set-SmaSchedule
Cmdlet	Set-SmaVariable
Cmdlet	Start-SmaRunbook
Cmdlet	Stop-SmaJob
Cmdlet	Suspend-SmaJob

See also For more details of these cmdlets, refer to the complete command reference at http://download.microsoft.com/download/A/0/B/A0BB6495-3010-43A8-B06C-6B0F0E8F92D9/SC2012R2_SMA_Cmdlets.pdf.

This array of cmdlets greatly extends the possible scenarios for operations to be performed using the Service Management Automation solution. Leveraging the module allows for standard Windows Powershell scripts to interact directly with service.

Running a runbook using the PowerShell module

This example launches a runbook remotely, with a set of parameters supplied directly to the runbook, and then, when the runbook is complete, uses the Job ID to display the runbook output stream.

```
$myJob = Start-SmaRunbook -WebServiceEndpoint https://sma.contoso.com -Port 9090 -Name
"sample-using-runbookparameters" -Parameters @{"Name"="World";"Number"="73"}

$jobID = $myJob.JobID

$loop = $true

While ($loop)
{
    $job = Get-SmaJob -id $jobID -WebServiceEndpoint "https://sma.contoso.com"
    $status = $job.JobStatus
    $loop = (($status -ne "Completed") -and ($status -ne "Failed") -and ($status -ne
        "Suspended") -and ($status -ne "Stopped"))
}

$out = Get-SmaJobOutput -Id $jobID -Stream Output -WebServiceEndpoint
"https://sma.contoso.com"
```

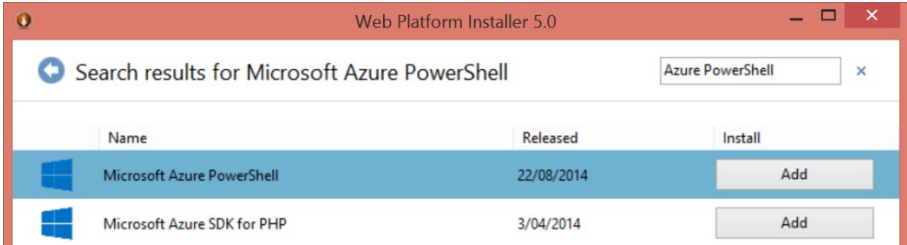
Azure Automation PowerShell cmdlets

When administering Azure Automation from Windows PowerShell, you use a separate set of cmdlets included in the Azure PowerShell module. Since the Azure Automation service handles the configuration and management of your automation account, you only need to manage runbooks, assets, and jobs. The set of cmdlets includes a limited set of commands that you can use to manage your runbooks, schedules, and jobs.

To access the Azure Automation cmdlets, install the Azure PowerShell module using the Microsoft Web Platform Installer as follows:

1. Browse to <http://www.microsoft.com/web/downloads/platform.aspx>, and click Free Download.
2. When prompted, run the wpilauncher.exe file.
3. After it has installed, launch the Web Platform Installer application.

4. On the Products tab at the top of the Web Platform Installer application, in the search field, type **Azure PowerShell**.
5. Click Add next to the Microsoft Azure PowerShell item, and then click Install.



6. When installation is complete, launch the Microsoft Azure PowerShell link from the Start menu.

NOTE Before you can use the Azure PowerShell module, you will need to establish trust with your Microsoft Azure subscription from your client computer. For details on using Azure Active Directory for authenticating to your Microsoft Azure subscription, refer to <http://azure.microsoft.com/en-us/documentation/articles/install-configure-powershell/#Connect>.

To use the Azure PowerShell cmdlets from a script or from a standard Windows PowerShell session, load the module using the following command:

```
Import-Module -Name Azure
```

The Azure PowerShell module currently includes the following Azure Automation cmdlets:

```
PS C:\> get-command *azureautomation*
```

CommandType	Name	ModuleName
-----	----	-----
Cmdlet	Get-AzureAutomationAccount	Azure
Cmdlet	Get-AzureAutomationJob	Azure
Cmdlet	Get-AzureAutomationJobOutput	Azure
Cmdlet	Get-AzureAutomationRunbook	Azure
Cmdlet	Get-AzureAutomationRunbookDefinition	Azure
Cmdlet	Get-AzureAutomationSchedule	Azure
Cmdlet	New-AzureAutomationRunbook	Azure
Cmdlet	New-AzureAutomationSchedule	Azure
Cmdlet	Publish-AzureAutomationRunbook	Azure
Cmdlet	Register-AzureAutomationScheduledRunbook	Azure
Cmdlet	Remove-AzureAutomationRunbook	Azure
Cmdlet	Remove-AzureAutomationSchedule	Azure
Cmdlet	Resume-AzureAutomationJob	Azure

Cmdlet	Set-AzureAutomationRunbook	Azure
Cmdlet	Set-AzureAutomationRunbookDefinition	Azure
Cmdlet	Set-AzureAutomationSchedule	Azure
Cmdlet	Start-AzureAutomationRunbook	Azure
Cmdlet	Stop-AzureAutomationJob	Azure
Cmdlet	Suspend-AzureAutomationJob	Azure
Cmdlet	Unregister-AzureAutomationScheduledRunbook	Azure

Note that as with all Microsoft Azure products and services, changes and enhancements are continuously added; therefore, the list above may change over time.

When using the Azure Automation cmdlets in the Azure PowerShell module, all actions are targeted against an Azure Automation account and therefore have a required parameter called `AutomationAccountName`. Rather than repetitively typing this parameter name and value, you can use a default parameter value in your Windows PowerShell session to set the value automatically. For example, in the Azure PowerShell console, use the following command to set the default value for the `AutomationAccountName` parameter for all related Azure Automation cmdlets:

```
$PSDefaultParameterValues = @{"*AzureAutomation*:AutomationAccountName"="<>AccountName">"}
```

See also For more details of using default parameter values, refer to *about_Parameters_Default_Values* at <http://technet.microsoft.com/en-us/library/hh847819.aspx>.

Creating and running a runbook using the Azure Automation PowerShell cmdlets

Follow this example to create a basic runbook in Azure Automation and run it using the Azure Automation cmdlets.

1. Create a Windows PowerShell script file called `Hello-World.ps1` and include the following code:

```
workflow Hello-World
{
    Write-Output "Hello World!!!"
}
```

2. Open an Azure PowerShell console from the Start menu.
3. Set the default value for your Automation account name:

```
$PSDefaultParameterValues =
@{"*AzureAutomation*:AutomationAccountName"="<>AccountName">"}
```

4. Use the following command to create a new runbook:

```
$runbook = New-AzureAutomationRunbook -Name "Hello-World" -Description "Example
runbook." -Tags "Example"
```

5. Update the runbook workflow with the script you created in step 1:

```
$runbook | Set-AzureAutomationRunbookDefinition -Path <Path to script>\Hello-World.ps1 -Overwrite
```

6. Publish the runbook:

```
$runbook | Publish-AzureAutomationRunbook
```

7. Run the runbook and view the results:

```
$runbook | Publish-AzureAutomationRunbook  
$job = $runbook | Start-AzureAutomationRunbook  
$job | Get-AzureAutomationJobOutput -Stream Output
```

Web portal interfaces

The Microsoft Azure Management Portal and the Windows Azure Pack Service Management Portal provide the web-based interface for the Microsoft automation solutions. Each portal provides the following:

- An overview dashboard of the system
- The ability to create and manage assets used in the system
- A set of runbook authoring and management capabilities
- The ability to launch runbooks, supply user input, and view job status and outputs

The Dashboard view (see Figure 3-3) provides an operational overview of the Service Management Automation deployment, including a graph and list view of current runbook job activity. From this page you can view the health and performance of the system.

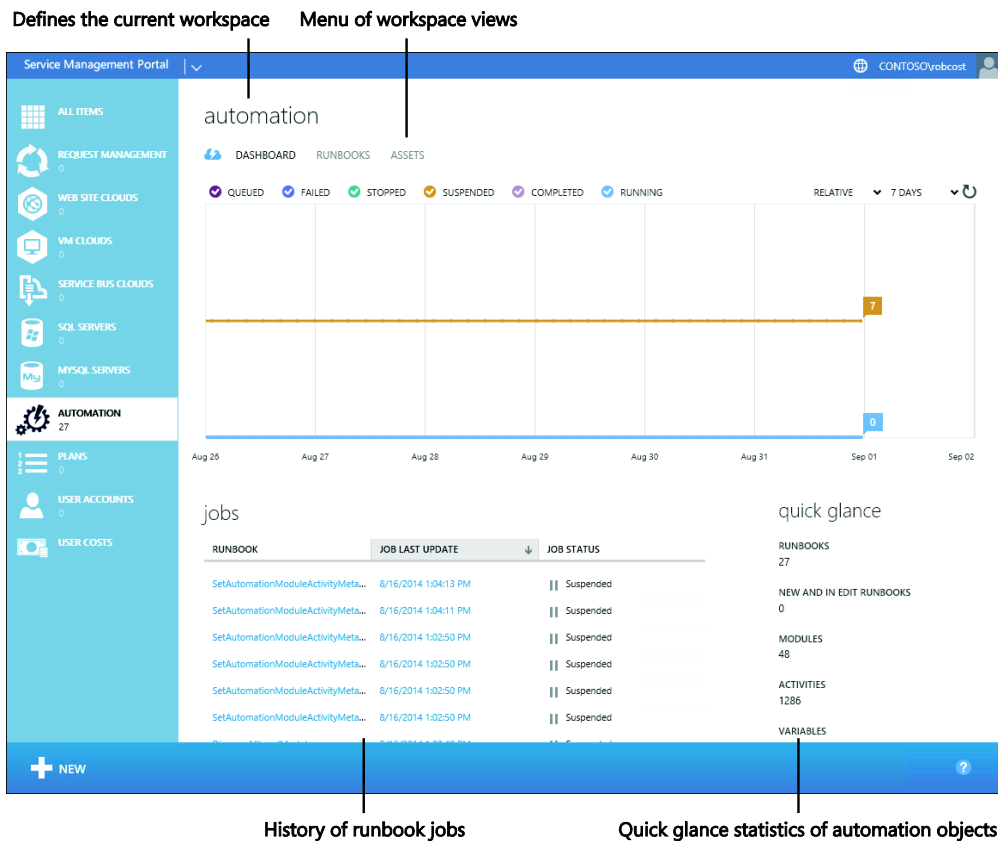


FIGURE 3-3 Elements of the automation Dashboard view in the Service Management Portal

The Runbooks view (see Figure 3-4) displays a filterable list of runbooks. From this page, you can take the following actions:

- Start a runbook
- Import a runbook from a .ps1 file
- Export a runbook to a .ps1 file
- Delete a runbook

Clicking the linked name of a runbook (in dark blue) opens the runbook authoring workspace, which will be covered in more detail in Chapter 4, "Implementing automation."

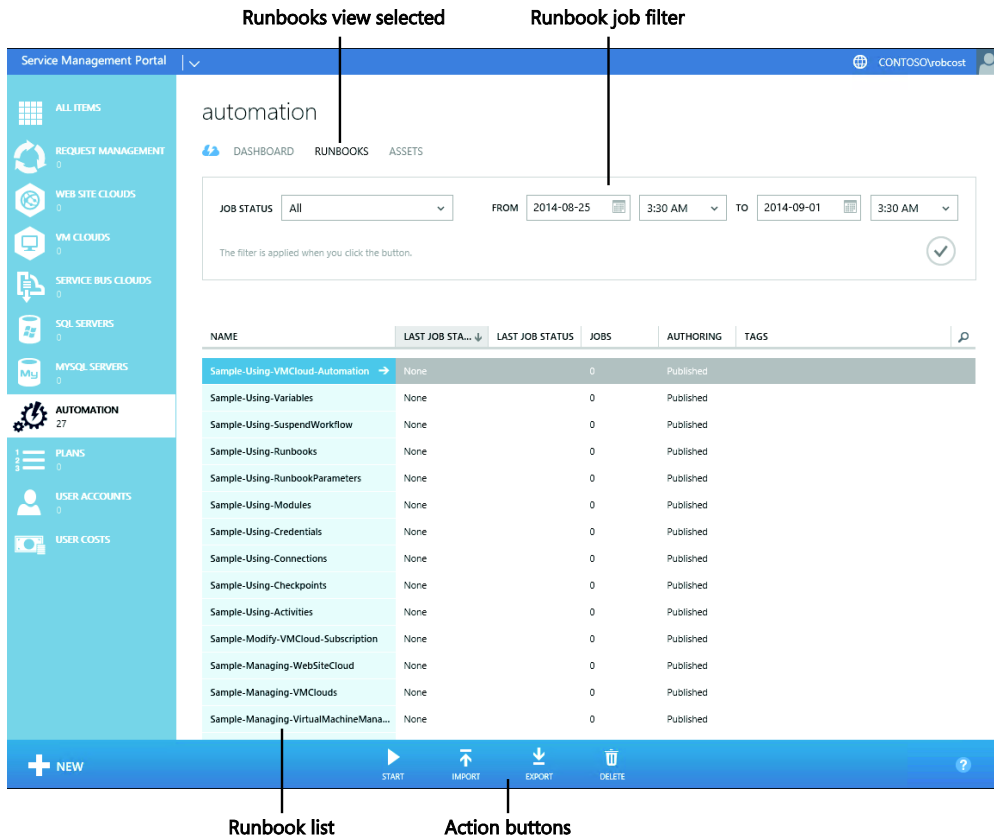


FIGURE 3-4 Elements of the automation Runbook view in the Service Management Portal

The Assets view (see Figure 3-5) displays all information for registered modules, connections, credentials, variables, and schedules. Chapter 4 covers managing assets.



FIGURE 3-5 Elements of the automation Assets view in the Service Management Portal

When using the management portals, you will be navigating a collection of workspaces and views. Figure 3-6 provides a map of the available workspaces and views in the portals, which are explored further throughout this book. As shown in the figure, the initial view you will use is the Automation workspace, which provides access to the following views:

- Dashboard view
- Runbooks view
- Assets view

From the Runbooks view, you can access any runbook to reach the Runbook workspace. This workspace provides access to additional views for managing a single runbook, including:

- Runbook Dashboard view
- Jobs view
- Author view
- Schedule view
- Configure view

From the Jobs view, you can review a runbook's job history, and accessing a particular job takes you to the Job workspace, which provides additional views for reviewing the details of a job.

Finally, the Asset view under the Automation workspace provides views for managing your assets.

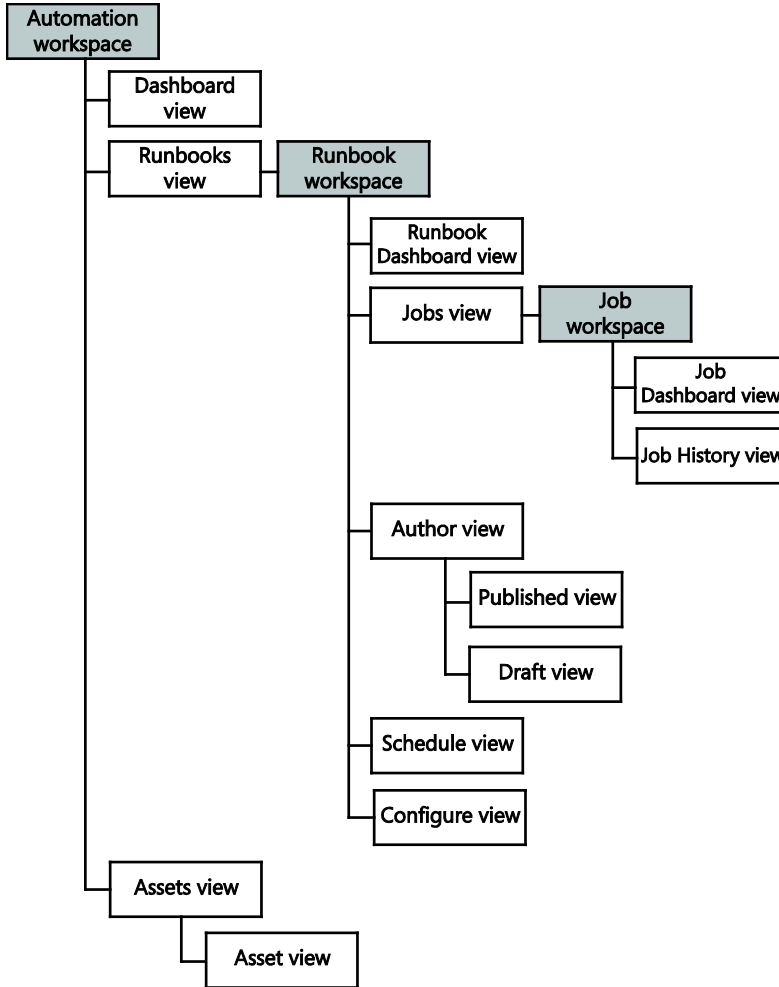


FIGURE 3-6 Tree view of workspaces and views in the management portals

This page intentionally left blank

Implementing automation

This chapter explores how to implement Microsoft automation solutions. You will learn:

- How to use asset objects in your solution
- How to use runbooks in the management portals
- How a runbook is structured
- How to author and publish runbooks
- How to author Integration Modules to use in your runbooks

Using assets

As previously identified in Chapter 1, "Why automation?" assets provide your runbooks with static configuration details commonly used in your environment. All assets can be managed from the Windows Azure Pack Service Management Portal and the Microsoft Azure Management Portal, and are accessed using the Assets view of the Automation workspace. From this workspace, you can click Add Setting to launch the Add Setting dialog box to create new assets (see Figure 4-1).

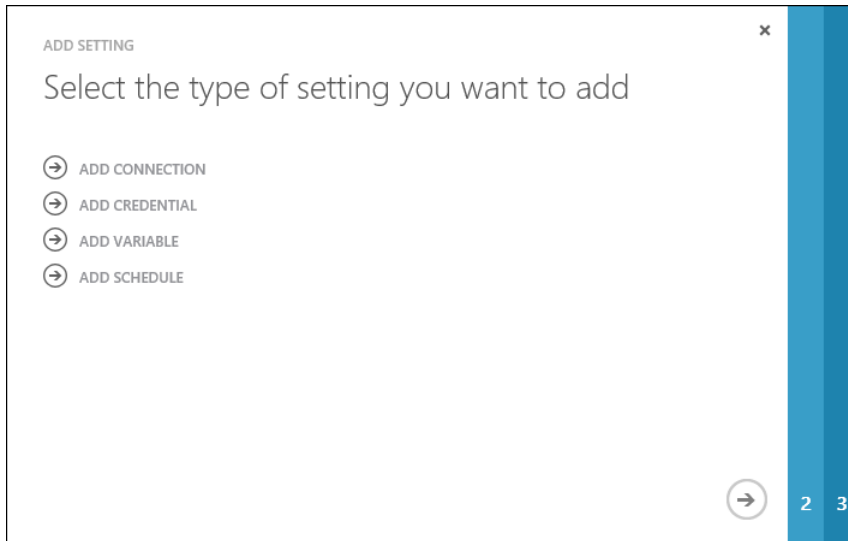


FIGURE 4-1 Add Settings dialog box for adding assets

Connections

Connections store details of a service that is used by multiple automation activities and include variables such as a server name, user name, and passwords. Connections are useful for connecting to services such as Microsoft System Center Virtual Machine Manager (VMM) or a Microsoft Azure subscription.

By default, a Service Management Automation deployment includes the following connection types:

- Azure
- ConfigurationManager
- DataProtectionManager
- MgmtSvcAdmin
- MgmtSvcConfig
- MgmtSvcMySql
- MgmtSvcSqlServer
- OperationsManager
- OrchestratorService
- ServiceBus
- ServiceManagementAutomation
- SpfAdmin
- VirtualMachineManager
- WebHostingSnapin

Each connection type is defined in the Windows PowerShell modules that are loaded by the service. In the "Authoring Integration Modules" later in this chapter, you'll learn how to author your own Windows PowerShell modules for the Microsoft automation solutions, including how to define connection types, enabling you to specify how connections are made to your own systems or products.

The configuration settings requested will depend on the connection type you use. For example, the Azure connection type requests settings such as subscription ID and certificate, whereas the Virtual Machine Manager connection type requests a server name, user name, and password. The following example describes how to create a Virtual Machine Manager connection:

1. From the Assets view in the Automation workspace, click Add Settings.
2. Click Add Connection.
3. In the Connection Type drop-down list, select VirtualMachineManager.
4. Provide a name for the connection, such as **myVMMConnection**.
5. Click the Next arrow.

6. Provide values for the Computername, Username, and Password fields.
7. Click Complete to finish adding the connection.

After you create a connection by using Add Setting, it can be consumed in a runbook as follows:

```
# Get connection asset object
$vmcConnection = Get-AutomationConnection -Name "myVMMConnection"

# Access connection properties
$vmcServerName = $vmcConnection.ComputerName
$vmcUsername = $vmcConnection.Username
$vmcEncryptedPassword = $vmcConnection.Password
```

Credentials

Credential assets store two types of credentials: PowerShell credentials and certificate credentials. Credentials allow automation activities to refer to a repository of securely stored credentials that can be used to access systems and services as required without user input. A PowerShell credential asset stores a standard user name and encrypted password combination. A certificate credential asset stores a certificate in either .cer or .pfx format, depending on whether the certificate requires its private key to be available.

The following example shows how to create a credential asset:

1. From the Assets view in the Automation workspace, click Add Settings.
2. Click Add Credential.
3. In the Credential Type drop-down list, select the required type.
4. Provide a name for the credential, such as **myCredential**.
5. Click the Next arrow.
6. If creating a PowerShell credential, provide values for the user name and password fields. If creating a certificate credential, click Browse to locate your certificate file.
7. Click Complete to finish adding the credential.

After you create a credential by using Add Setting, it can be consumed in a runbook as follows:

```
# Get PowerShell Credential asset object
$myCred = Get-AutomationPSCredential -Name "myCredential"

# Get Certificate Credential asset object
$myCert = Get-AutomationCertificate -Name "myCertificate"
```

Variables

Variable assets store name-value pair variables, which are consumed in runbooks. Variables can be defined with the following types: string, integer, boolean, datetime, and not defined.

Variables can be optionally encrypted if the values are sensitive, for example, if they are storing license keys or passwords.

The following example shows how to create a variable asset:

1. From the Assets view in the Automation workspace, click Add Settings.
2. Click Add Variable.
3. In the Variable Type drop-down list, select the required type.
4. Provide a name for the variable, such as **myVar**.
5. Click the Next arrow to continue.
6. Provide a value for the variable.
7. Click Complete to finish adding the variable.

After you create a variable by using Add Setting, it can be consumed in a runbook as follows:

```
# Get Variable asset object
$myVar = Get-AutomationVariable -Name "myVar"
```

Schedules

Schedule assets define a regular schedule to which an activity can be linked, for example, to initiate a runbook at the start or end of business each day.

The following example shows how to create a schedule asset:

1. From the Assets view in the Automation workspace, click Add Settings.
2. Click Add Schedule.
3. Provide a name for the schedule, such as mySched.
4. Click the Next arrow to continue.
5. Select the type of schedule: One Time to set the schedule to run one time only or Daily to set the schedule to run at the same time each day.
6. Provide a value for the data and a time to process.
7. Click Complete to finish adding the schedule.

After you create a schedule by using Add Setting, it can be attached to a runbook as follows:

1. Locate the runbook you are attaching the schedule to.
2. Click Use An Existing Schedule.
3. Select the schedule created previously, such as mySched.
4. Click the Next arrow.
5. Provide any parameter values expected by the runbook.
6. Click Complete to finish attaching the schedule to the runbook.

Integration Modules

Integration Modules store Windows PowerShell modules for use by automation activities. These allow you to extend the capability of the Microsoft automation solutions by leveraging Microsoft and third-party developed modules or even modules you create. Creation and use of integration modules is covered in the section "Authoring integration modules" later in this chapter.

Using runbooks

Runbooks provide the process automation definition for the Microsoft automation solutions. All runbooks can be managed from the portals and are accessed in the Runbooks view of the Automation workspace. In the Runbooks view, you can:

- Create new runbooks by clicking New at the bottom left of the view (see Figure 4-2). You'll be prompted for the name of the runbook and optional values for the description and tags for the runbook.
- Start a runbook by clicking Start . You'll be prompted for any parameters required by the runbook and for confirmation to start.
- Import a runbook by clicking Import. This allows you to import a pre-existing runbook, usually in a .ps1 file.
- Export a runbook by clicking Export. This allows you to export a single runbook as a .ps1 file.
- Delete a runbook by clicking Delete.

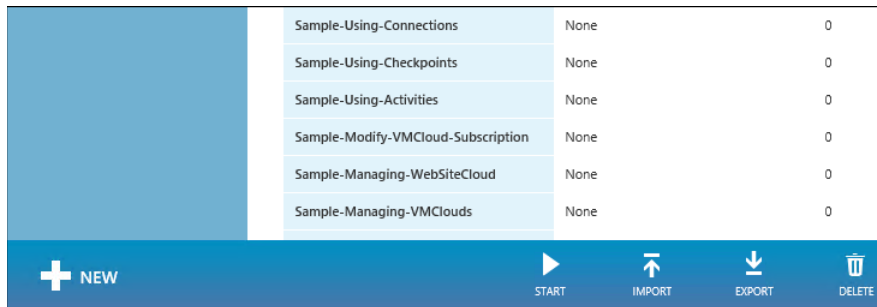


FIGURE 4-2 Action buttons in the portal user interfaces

After creating your runbook, click its name in the Runbooks view to open the Runbook workspace (see Figure 4-3), where you can configure that runbook directly.

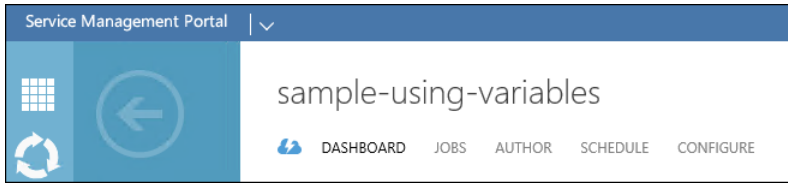


FIGURE 4-3 Runbook workspace views in the Service Management Portal

The Runbook workspace provides the following views:

- Dashboard view, similar to the Automation workspace dashboard, which provides a summary of the runbooks jobs in graph and list formats
- Jobs view, which provides a full, filterable list of the runbook jobs
- Author view for viewing the published version of the runbook and for editing a draft version of the runbook
- Schedule view for viewing or attaching schedules to the runbook
- Configuration view to edit general settings of the runbook, such as the optional description and tags values, and to configure logging levels for the runbook (explored in Chapter 5, "Managing runbooks")

Authoring runbooks

You author runbooks in the Runbook workspace, in the Author view. Runbooks can be either published or draft runbooks. A published runbook represents a complete version that is ready to run or to be nested in other runbooks, and it will log activity to the database. A draft runbook, in contrast, can be edited and tested before being published. However, be aware that testing a runbook runs it live on a runbook worker server, so be sure to develop in a development environment before running your runbooks in production. If you decide not to publish a runbook draft, you can dispose of your changes by using the Discard Draft action.

The following example shows how to create a new runbook. This runbook will serve as the basis for the rest of the examples in this section:

1. From the Service Management Portal, click New.
2. Click Runbook, then click Quick Create.
3. Provide a name for the runbook, in this case, **Hello-World**.
4. Click Create to create the runbook.
5. From the Runbooks view in the Service Management Portal, click the name of the newly created runbook.
6. Click the Author view of the runbook, and then click the Draft view.
7. Click Edit Runbook to enter edit mode.

Exploring the runbook workflow structure

A runbook contains a Windows PowerShell workflow, which is processed by the Microsoft automation solutions infrastructure. The runbook workflow structure, exactly like a standard Windows PowerShell workflow, begins with the workflow block, which defines the parent workflow script block. The runbook created in the previous section shows the default structure in the Hello-World runbook. Note that both the workflow and runbook name are the same:

```
workflow Hello-World
{
}
```

You define the steps that your runbook will follow within the curly braces. If your runbook will output data, define the output type first. This allows you to control the type of data that other runbooks will receive if they call this runbook and also enables future iterations of Microsoft automation solutions to use this field for discovery. For example:

```
workflow Hello-World
{
    [OutputType([string])]
}
```

Generally, a runbook contains one or more input parameters that allow you to feed data into the runbook at runtime, either manually, passed from another runbook, or via a scheduled or automated method. To define parameters in your runbook, extend the workflow as follows:

```
workflow Hello-World
{
    [OutputType([string])]
    param
    (
        [parameter(Mandatory=$true)]
        [string]$firstName,
        [parameter(Mandatory=$true)]
        [string]$lastName
    )
}
```

Parameters in runbooks have the following constraints:

- Parameter definitions support only the following attributes: Name, Type, Default Value (Optional), and Mandatory (Optional).
- Only the following types are permissible: string, string array, object, integer, datetime, boolean, and PSCredential.
- Parameters of string array or object types must be in a JSON string format.

Variable definitions in a runbook follow standard Windows PowerShell syntax. In the following example, the input parameters of the runbook are joined to create a variable:

```
workflow Hello-World
{
    [OutputType([string])]
    param
    (
        [parameter(Mandatory=$true)]
        [string]$firstName,
        [parameter(Mandatory=$true)]
        [string]$lastName
    )

    $fullName = "$firstName $lastName"
}
```

To complete the basic example, output the value of the new variable by using the write-output activity or by simply using the variable name:

```
workflow Hello-World
{
    [OutputType([string])]
    param
    (
        [parameter(Mandatory=$true)]
        [string]$firstName,
        [parameter(Mandatory=$true)]
        [string]$lastName
    )

    $fullName = "$firstName $lastName"

    write-output $fullName
    # or
    # $fullName
}
```

Differences between standard Windows PowerShell and workflows

When you write runbooks, it's important to understand the differences between writing standard Windows PowerShell scripts and Windows PowerShell workflows. The main considerations are how activities are used inside a workflow and how cmdlets are consumed.

An activity in a workflow is any task processed directly in the workflow script block. This can take the form of a cmdlet, a workflow-specific activity, or an activity defined by the Microsoft automation solutions.

A standard activity in a workflow can be any cmdlet defined in the Microsoft.PowerShell.* modules or the Microsoft.Wsman.Management module. These activities are processed natively in a workflow. For example, the Get-Date cmdlet exists in the Microsoft.PowerShell.Commands.Utility module and, therefore, can be used directly in a workflow.

A cmdlet activity is any cmdlet that is not in one of the modules mentioned above. When a cmdlet activity runs, Windows Workflow Foundation (which hosts Windows PowerShell Workflow) processes the cmdlet inside an InlineScript activity behind the scenes and returns any output of the cmdlet to the workflow session. For example, the Get-Disk cmdlet exists in the Storage module and runs inside an InlineScript when processed in a workflow.

The Microsoft automation solutions also include additional activities to provide enhanced features when running tasks in runbooks. These activities, starting with either Get-Automation* or Set-Automation*, enable the Microsoft automation solutions to interact with assets in the database directly so they are available in the workflow. For example, the Get-AutomationCredential activity provides access to credential assets and is capable of decrypting the password, in contrast to using the Get-SmaCredential cmdlet, which can access a credential asset, but not its unencrypted password.

See also For more details of using activities in script workflows, please refer to <http://technet.microsoft.com/en-us/library/jj574194.aspx>.

Another key feature of Windows PowerShell Workflow that you can leverage in runbooks is the parallel and sequence activity constructs. These provide you the ability to run multiple activities in a defined manner. For example, to process a set of tasks in parallel, use the following script block construct:

```
workflow Hello-World
{
    Parallel
    {
        Get-Date
        Get-Disk
        Get-SmaCredential
    }
}
```


The sequence script block allows sequential processing of tasks. At first this might not seem useful; however, when this construct is nested in a Parallel block, it can control the order of activities to ensure they run in a specific way, for example:

```
workflow Hello-World
{
    Parallel
    {
        Get-Date
        Get-Disk
        Get-SmaCredential
        Sequence
        {
            Get-NetAdapter
            Get-NetIpAddress
        }
    }
}
```

When you use cmdlet activities, rather than running them from inside the Workflow script block and having Windows Workflow Foundation push them into an InlineScript, it is often easier to pre-define the InlineScript block. Doing so allows you to control when a block of script runs in a Windows PowerShell session and gives you the ability to target that session against a remote host. The InlineScript script block supports processing against remote hosts by providing parameter values for PSComputerName and PSCredential, as in the following example:

```
workflow Hello-World
{
    InlineScript
    {
        Get-Disk
    } -PSComputerName $computerName -PSCredential $credential
}
```

Note that data is passed into and out of the script block when you use InlineScript script blocks. Because Windows Workflow Foundation runs the script block in a new Windows PowerShell session, any variables defined or used in an InlineScript do not naturally inherit values from the parent workflow. To pass values into an InlineScript script block, use the \$Using: scope modifier, as in the following example.

```
workflow Hello-World
{
    $outerVar = "test"
    InlineScript
    {
        $innerVar = $Using:outerVar
    }
}
```

To output data from an InlineScript script block, assign it to a variable and write out your required output in the script block, as in the following example:

```
workflow Hello-World
{
    $output = InlineScript
    {
        Write-Output "Hello world!!!"
    }
    Write-Output $output
}
```

Defining parent and child runbook relationships

The Microsoft automation solutions enable the development of runbooks that can be linked together using parent-child relationships. This allows you to author re-usable small modular runbooks that achieve defined tasks and group them together in a runbook set. You should use a parent runbook that controls process logic and leverages many child runbooks for runbook development. This avoids having overly complex runbooks with potentially thousands of lines of Windows PowerShell code.

Establishing a parent-child relationship for runbooks requires two things:

- Calling a child runbook from a parent runbook
- Returning data to the parent runbook

There are two methods for calling other runbooks and passing them parameters inside a runbook:

- **Inline call** Synchronously invokes a runbook directly and waits for its completion; optionally retrieves data from the child runbook

```
workflow Hello-World
{
    Example-Runbook -Param1 "value1" -Param2 " value2" -Param3 "value3"
}
```

- **Start-SmaRunbook** Asynchronously invokes a runbook and does not wait for completion; optionally returns the initiated runbook's Job ID

```
workflow Hello-World
{
    Start-SmaRunbook -WebServiceEndpoint "https://sma.contoso.com"
                    -port 9090 -name "Sample-Using-Modules" -Parameters
                    @{ "Modulepath"="C:\Modules"; "Servername"="Server01"}
}
```

For an effective parent-child relationship, you also need to return data from the child runbook so the parent runbook can manage the success or failure of the child or perform actions based on the data returned. To return data from the child, use the `OutputType` defined at the start of the workflow and output the data required at the end of the runbook, as in the following example:

```
workflow Child-Runbook
{
    [OutputType([string])]
    Write-Output "Output data."
}
```

This can then be consumed by the parent runbook, as shown in the following example:

```
workflow Parent-Runbook
{
    $result = Child-Runbook
    Write-Output $result
}
```

These methods of presenting and returning data between runbooks provides a flexible solution for automation of long-running processes.

Authoring Integration Modules

Authoring Integration Modules for Microsoft automation solutions is essentially the same as authoring standard Windows PowerShell modules, with some simple additions to enable them to expose extra capability. If you're familiar with Windows PowerShell modules, you have used .psm1 files to store common script code which can be leveraged across multiple scripts.

Integration Modules use a standard .psm1 file, with the addition of the following:

- A Module Manifest file to define metadata about the module itself, such as the version, author, and description, and any prerequisites for the module
- An Integration Module Metadata file to define any custom connection types used by the module

You can zip these three files (module .psm1 file, manifest .psd1 file, and metadata .json file) and import them into the Microsoft automation solutions (see Figure 4-4). The names of these files and the zipped module is very important. A common name must be applied to each file to ensure the import process completes successfully.

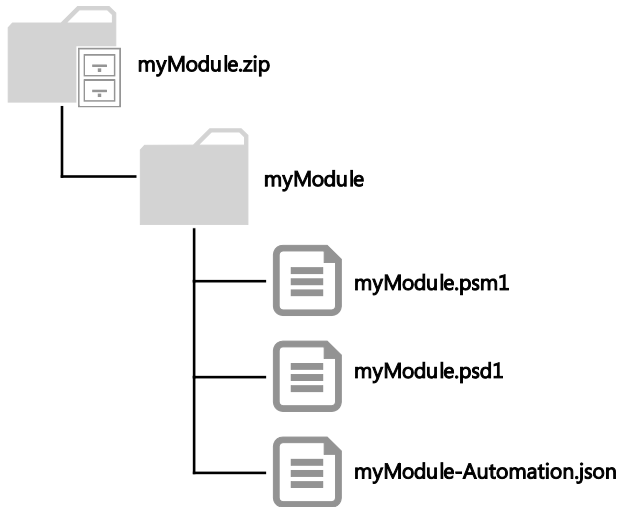


FIGURE 4-4 Structure of an Integration Module zip file

Creating an Integration Module file

As already noted, an Integration Module is essentially just a Windows PowerShell module file. A module file consists of the following:

- **Cmdlets** A function that is exported as a cmdlet, inheriting capabilities similar to compiled cmdlets, such as parameter sets, help URIs, and so on
- **Functions** A standard function used internally in the module or optionally exported
- **Variables** A standard variable object used internally in the module or optionally exported.

The following example shows a skeleton of a Windows PowerShell module using these object types.

```

$privateVariable = "0"
$publicVariable = "1"

<#
    .SYNOPSIS
    This cmdlet provides.... (Short description)

    .DESCRIPTION
    This cmdlet provides... (Long description)
  
```

```

.NOTES
Module Name : myModule
Created by   : Author Name
Date Coded   : 02/09/2014

.LINK
http://aka.ms/automate

#>
function Get-ExampleOutput
{
    [CmdletBinding(HelpURI='http://aka.ms/automate')]
    [OutputType([System.String])]
    param
    (
        [parameter(Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [string]$ParamName
    )

    # your code here...

    return $result
}

function PrivateExampleFunction
{
    [OutputType([System.String])]
    param
    (
        [parameter(Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [string]$ParamName
    )

    # your code here...

    return $result
}

Export-ModuleMember -Cmdlet "Get-ExampleOutput"
Export-ModuleMember -Variable $publicVariable

```

Creating an Integration Module Manifest file

An Integration Module Manifest file can be generated manually using any text editor or the New-ModuleManifest cmdlet. For details of the contents of a manifest file, please refer to [http://msdn.microsoft.com/en-us/library/dd878337\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/dd878337(v=vs.85).aspx).

Creating an Integration Module Metadata file

An Integration Module Metadata file provides the Microsoft automation solutions with details of a new connection type to expose through the service. This is useful for systems that may require a different set of connection variables, such as a password and an application key or ID, or that require multiple settings you would prefer to bundle into a single object.

The example below shows how to define a new connection type to be included in your Integration Module.

```
{
  "ConnectionFields": [
    {
      "IsEncrypted": false;
      "IsOptional": false;
      "Name": "StorageAccountName";
      "TypeName": "System.String";
    },
    {
      "IsEncrypted": true;
      "IsOptional": false;
      "Name": "StorageAccountKey";
      "TypeName": "System.String";
    }
  ],
  "ConnectionTypeName": "AzureBlobStorage",
  "IntegrationModuleName": "AzureBlobStorage"
}
```

This page intentionally left blank

Managing runbooks

This chapter explores the tools you can use to manage runbooks in the Microsoft automation solutions. You will learn:

- Which authoring tools you can use to create and edit runbooks and the strengths and weaknesses of each
- How to manage source control of runbooks
- How to manage runbook output and logs
- How to leverage error-handling techniques
- How to troubleshoot server issues

Runbook authoring tools

As you become more familiar with the Microsoft automation solutions, knowing how you can use various tools to create and maintain your library of runbooks becomes important. When you have authored a number of runbooks and generated thousands of lines of runbook workflows, challenges will arise, especially when you have multiple authors. The runbook authoring tools available to you provide a wide range of capabilities, each having its own strengths and weaknesses.

Management portals

The Microsoft Azure Management Portal and the Windows Azure Pack Service Management Portal, introduced in Chapter 3, "Understanding automation: Interfaces," enable you to author and manage runbooks directly. The management portals provide a web-based user interface to all aspects of the Microsoft automation solutions (see Figure 5-1) and will likely be your primary interface for managing runbooks and assets as you become familiar with the solutions.

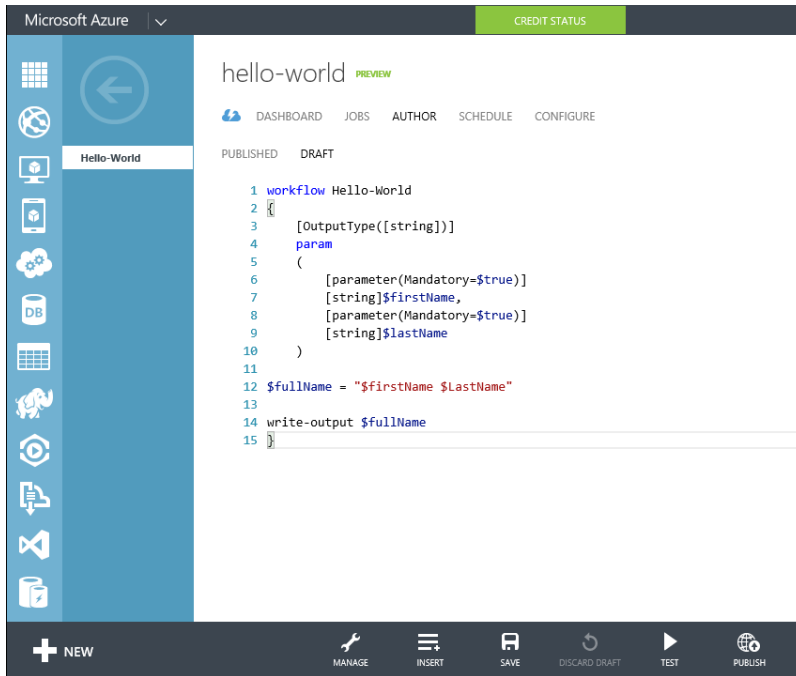


FIGURE 5-1 The Azure Management Portal interface

The management portals provide the following capabilities related to authoring runbooks:

- A live view of runbooks, assets and jobs in the Azure Automation or Service Management Automation system
- The ability to edit and test draft versions of your runbooks with full access to all assets
- A web-based authoring experience that provides basic intellisense auto-completion, text searching, and syntax highlighting
- The ability to create assets during runbook editing
- The ability to view output of a runbook in real time while it runs

The management portals do have some limitations, however, for runbook authoring. For example, the lack of integrated source control and the basic Windows PowerShell authoring user interface can be limiting.

Windows PowerShell ISE

The Windows PowerShell Integrated Scripting Environment (ISE) was introduced in Windows 7, along with Windows PowerShell 1.0, and is now included with all current versions of Windows (see Figure 5-2). Since its initial release, the ISE capabilities have expanded to include the ability to author Windows PowerShell workflows. Given this capability, it is possible to use the ISE to author runbooks, with some constraints.

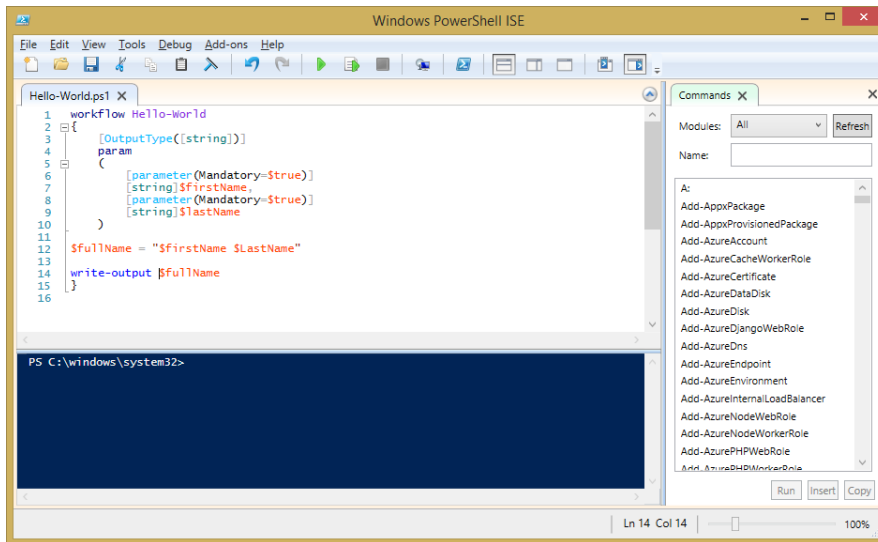


FIGURE 5-2 The Windows PowerShell ISE user interface

The ISE provides the following capabilities related to authoring runbooks:

- The ability to create and edit Windows PowerShell workflows
- A client-based authoring experience that provides full intellisense auto-completion, script outlining, text searching, and syntax highlighting
- Script snippet management
- Extensibility through add-ons

The ISE does not include direct integration with the Microsoft automation solutions; therefore, you cannot natively test your runbooks on runbook workers during authoring nor access assets stored in your Azure Automation or Service Management Automation system.

NOTE You can use the Emulated Automation Activities Windows PowerShell module to access assets in a Service Management Automation system when authoring runbooks in the ISE. For details of this solution, please refer to <http://blogs.technet.com/b/orchestrator/archive/2014/03/27/authoring-sma-runbooks-in-the-powershell-ise.aspx>. Note that at the time of this writing this solution does not work for Azure Automation.

Like the management portals, the ISE lacks an integrated source control capability, leaving source control as the responsibility of the runbook authors. There are possible solutions to this, however, which are explored in the following section.

Visual Studio IDE

Microsoft Visual Studio represents a complete Integrated Development Environment (IDE) with a wide range of capabilities and features (see Figure 5-3). Out of the box, Visual Studio is not a native Windows PowerShell authoring tool and requires add-ons to support full intellisense in Windows PowerShell scripts. Like the ISE, Visual Studio also is not directly integrated with the Microsoft automation solutions.

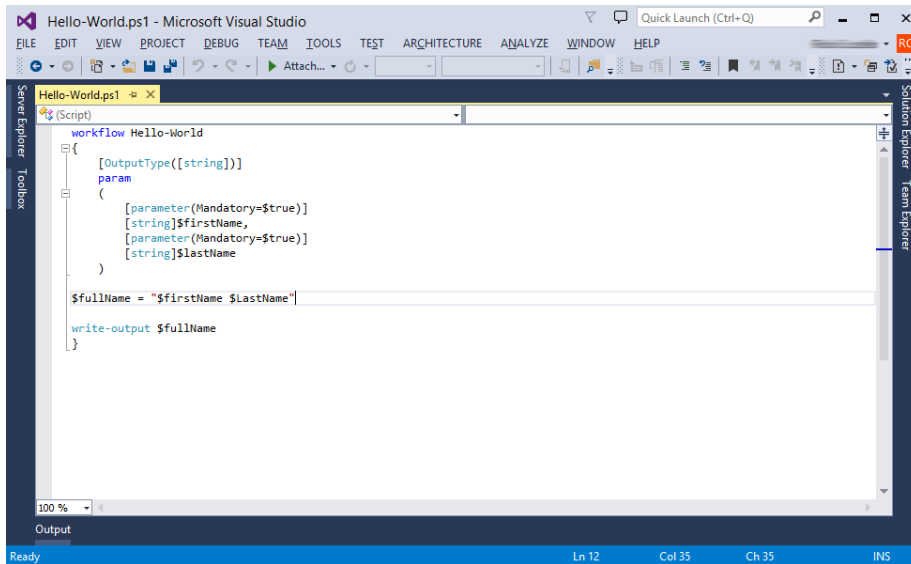
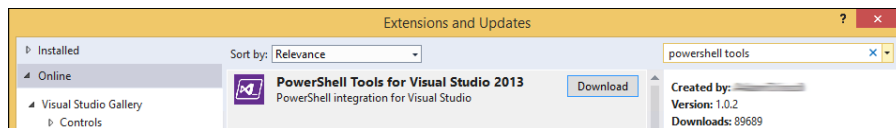


FIGURE 5-3 The Visual Studio IDE user interface

To enable extended Windows PowerShell authoring features in Visual Studio, you can use the PowerShell Tools for Visual Studio 2013 extension. To install this extension, follow these steps in the Visual Studio interface:

1. On the Tools menu, click Extensions And Updates.
2. In the Extensions And Updates dialog box, click Online in the left tree view.
3. Type **PowerShell Tools** in the search text box at the top of the dialog box.
4. From the list of search results, select PowerShell Tools for Visual Studio 2013, and then click Download.



5. After the extension downloads and runs, accept the license, and then install the extension.
6. Restart Visual Studio to complete the extension installation.

Visual Studio has some distinct advantages when integrated with a managed source control system, such as Microsoft Team Foundation Server or Visual Studio Online, which makes managing the runbook development lifecycle easier, as described in the following section.

Using source control

Traditionally, Windows PowerShell script authoring version control has been handled by a version number either inside the script file or as part of the script file name. The script author must decide what type of version numbering scheme is used and what types of changes trigger a major, minor, or incremental version numbering change. When multiple authors are involved with edits in a script repository, issues with untracked changes and data loss inevitably arise.

The concept of source control is generally associated with application development. However, as an IT professional, you are likely increasingly finding the need to manage growing libraries of scripts and configuration files. The Microsoft automation solutions are no different in this aspect: as you start to build a collection of runbooks, custom modules, and associated configuration files, the need to manage these objects is critical. Because the Microsoft automation solutions provide a centralized, multi-user authoring experience, leveraging source control tools for managing versions will help you avoid data loss by providing rollback capabilities.

The Microsoft automation solutions provide some built-in capabilities for managing runbook contents, including:

- Published versions of runbooks
- Draft versions of runbooks
- The `Get-SMARunbookDefinition` and `Set-SMARunbookDefinition` Windows PowerShell cmdlets
- Stored runbook versions linked to existing jobs

However, these provide only very basic version control, which, like traditional Windows PowerShell scripting, is at the runbook author's discretion.

A managed source control solution can resolve some of these issues and enhance your runbook authoring experience. Tools such as Team Foundation Server or Visual Studio Online offer enterprise-level capabilities.

Team Foundation Server

Team Foundation Server for version control is useful for an on-premises Service Management Automation system. Team Foundation Server provides a centralized version control system, along with other very useful capabilities for managing development activities as part of a team, such as work item management and bug tracking.

You can use Team Foundation Server version control features to check in and check out runbook script files for editing. Native integration between Team Foundation Server and Service Management Automation is not currently available out of the box, but solutions exist to provide more seamless integration between the tools. Using Team Foundation Server to manage your Service Management Automation runbooks will help you build a historical view of what has changed, when it changed, and who performed the change.

NOTE An example solution for integrating Team Foundation Server and Service Management Automation can be found at <http://blogs.technet.com/b/privatecloud/archive/2014/05/09/automation-mvp-spotlight-series-tfs-and-service-management-automation-better-together.aspx>.

See Also For more details on Team Foundation Server, please refer to <http://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx>.

Visual Studio Online

Visual Studio Online provides the same functionality as Team Foundation Server, but it is hosted and managed by Microsoft as a cloud service. With Visual Studio Online as your version control solution, you can offload the build and management of additional infrastructure and access your runbook scripts from anywhere.

Team Foundation Server uses a centralized version control system called Team Foundation Version Control (TFVC). Visual Studio Online also uses this system, but can also use the Git distributed version control system, which has gained popularity in development communities. Using Git, your team members can store a local copy (called a *clone*) of your repositories (called *repos*) on client computers and synchronize them with the Visual Studio Online repository.

While Visual Studio Online could be used for version management of a Server Management Automation system, it is more suited for use as part of an Azure Automation solution. Like Team Foundation Server, no native integration between these tools is currently available, although solutions exist to provide integration between them.

NOTE Find an example solution for Visual Studio Online and Azure Automation at <http://azure.microsoft.com/blog/2014/07/24/azure-automation-integrating-runbook-source-control-using-visual-studio-online/>.

See Also For more details on Visual Studio Online, please refer to <http://www.visualstudio.com/products/what-is-visual-studio-online-vs>.

Using logging

When using runbooks to automate tasks in your environment, gathering feedback from your runbooks is important to ensure you have a record of the activities performed and to provide diagnostic information. The Microsoft automation solutions leverage native Windows PowerShell workflows output capabilities and surface their output either through the output pane in the management portal runbook authoring view or through the job output history view. The available types of output streams include the following:

- **Output** Used to output objects in a runbook using the Write-Output activity. Should not be used for general output.
- **Warning** Used to capture warning output from activities in a runbook.
- **Error** Used to capture error output from activities in a runbook.
- **Progress** Generally not used with runbooks unless you need to track the start and finish of each activity for troubleshooting. Enable logging of progress records using the runbook configuration view of an individual runbook.
- **Verbose** Used for writing general output of a runbook using the Write-Verbose activity. Enable logging of verbose records using the runbook configuration view of an individual runbook.
- **Debug** Used to output debugging details. Enable logging of debug records using the runbook configuration view of an individual runbook. Note that debug logging is not available in Azure Automation.

NOTE More details of output types and their use can be found at <http://blogs.technet.com/b/orchestrator/archive/2014/01/16/sma-capabilities-in-depth-controlling-runbook-streams-for-testing-and-troubleshooting.aspx>.

As you author runbooks using the management portals, you can test draft versions in the runbook authoring view. When testing a runbook from this view, an output pane appears in the interface to provide near real-time output of the runbook as it runs (see Figure 5-4). In the case of a published runbook, this output is displayed on the History tab of the runbook job (see Figure 5-5).



FIGURE 5-4 Output pane of a runbook

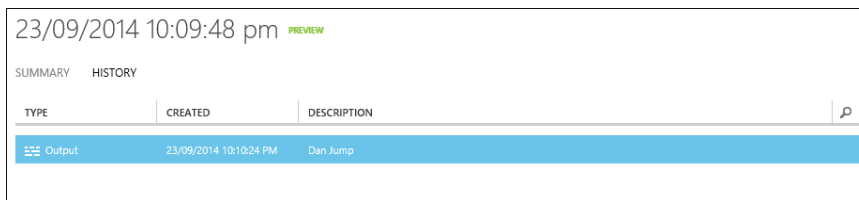


FIGURE 5-5 Job history view of a runbook

Output streams are stored in the Microsoft automation solutions databases as part of the runbook job history. Although extremely useful for troubleshooting and validation, excessive output can result in excessive database disk write IO and have a detrimental impact on the performance of the system. Service Management Automation provides a built-in grooming process that relies on the SQL Server Agent to purge job history, and when this is enabled, the process will run every 15 minutes. By default, data older than 30 days is purged from the database.

NOTE For more details on database purging process, see <http://technet.microsoft.com/en-us/library/dn469633.aspx>.

In some scenarios, retaining only 30 days of log data may not be sufficient for auditing or compliance purposes. Also, the data contained in the output logs may not meet your desired level of detail of the actions performed. A solution for this is to author your own stream output in the desired format and implement a custom logging solution. For example, a custom logging solution could output data for a Service Management Automation system to one of the following destinations.

- A separate SQL Server database
- A file on the local server or a remote file share
- An email, which can be sent to a shared mailbox or a log collection tool
- A Service Bus queue, which can be consumed by another service

If you are using Azure Automation, a custom logging solution could output data to one of the following destinations:

- Table storage, one of the Azure Storage services
- File storage, another one of the Azure Storage services
- An Azure SQL Server database
- An Azure Service Bus queue, which can be consumed by another service

Chapter 6, "Examples of automation scenarios," explores two of these scenarios to get you started with external logging.

Using error handling

In any operation within your environment, the inevitability of an error needs to be accounted for in your runbooks. The Windows PowerShell default behavior for terminating errors is to display the error and halt processing, which results in the runbook suspending. Managing how this behavior could impact your runbooks and processing logic is very important. One high-level consideration is the need to classify errors, for example, as follows:

- **Stop error** The error halts all runbook processing and prevents downstream events from succeeding.
- **Suspend error** The error places the runbook in a suspended state and operator intervention may be required to stop or resume processing.
- **Non-fatal error** The error does not hinder downstream events from succeeding.

With a classification system, you can more easily design scripting logic to handle each classification. As indicated in the description of a non-fatal error in the preceding list, sometimes an error condition is acceptable and the runbook can continue processing. The suspend error and non-fatal conditions may require a different method of error handling so as not to terminate processing, such as one of the following methods:

- **Try, Catch, and Finally** This method of error trapping and management works as the name suggests: Try something, catch any errors, and finally process any close-out activities. To learn about the pattern in more detail, go to <http://technet.microsoft.com/en-us/library/hh847793.aspx>.
- **Trap** The Trap method provides custom exception handling instead of the default Windows PowerShell error-handling mechanism. To learn about the pattern in more detail, go to <http://technet.microsoft.com/en-us/library/hh847742.aspx>.

NOTE The Trap method is not supported in a PowerShell Workflows block but can be used in an InlineScript block.

Utilizing both classification and error handling patterns, you can intercept error conditions and determine an appropriate action. Listing 5-1 shows an example of the Trap method with the Continue option that will attempt to use a WMI disk volume query against a fictitious server. The resulting failure will be caught, but it will not stop the runbook from further processing. If the Continue option is replaced with a Break command, the runbook enters a suspended state.

Listing 5-1 Example workflow using the Trap method to handle an error

```
workflow Test-Trap
{
    InlineScript{
        Write-Verbose "Attempting WMI query"
        Trap{
            write-verbose("[ERROR] " + $_)
            continue
        }
        gwmi win32_volume -ComputerName "NotARealServerName" -ErrorAction Stop
        write-verbose "Completed WMI Query"
    }
}
```

Listing 5-2 shows an example of the Try, Catch, and Finally method with the Continue option, which will attempt to use the same WMI disk volume query against a fictitious server. The resulting failure will be caught by the Catch block and stop the runbook from further processing. The Finally block will always run regardless of the Catch section actions.

Listing 5-2 Example workflow using the Try, Catch, and Finally method to handle an error.

`workflow Test-TryCatch`

```
{
    $VerbosePreference = "Continue"
    $servername = "NotARealServerName"
    try
    {
        gwmi win32_volume -PSComputer $servername -ErrorAction Stop
    }
    catch
    {
        Write-Verbose "Exiting runbook server was not found" -Verbose
        Exit
    }
    Finally
    {
        Write-Verbose "exiting wmi drive information" -Verbose
    }
}
```

Troubleshooting runbooks

When you are developing and testing your runbooks, your primary troubleshooting tool will be the runbook authoring view's output pane and the job history views that were introduced earlier in this chapter. These provide you with output details that you will need to identify issues and correct them inside your runbooks.

For troubleshooting issues outside the runbook development and testing space, you will need to understand how the system's infrastructure is configured and where to look when trying to identify the cause of issues. Because the Azure Automation solution is provided as a service, you will not have visibility or access to the system's underlying infrastructure. Given this, the remainder of this section will address how to troubleshoot the underlying infrastructure of a Service Management Automation system.

Logs

Service Management Automation, as discussed earlier in this chapter, logs job output to its own database; however, it also logs to the local Windows event log for operational application logging (see Figure 5-6). In this log, you will find events and errors relating to the runbook service, sandboxes, and database connectivity.

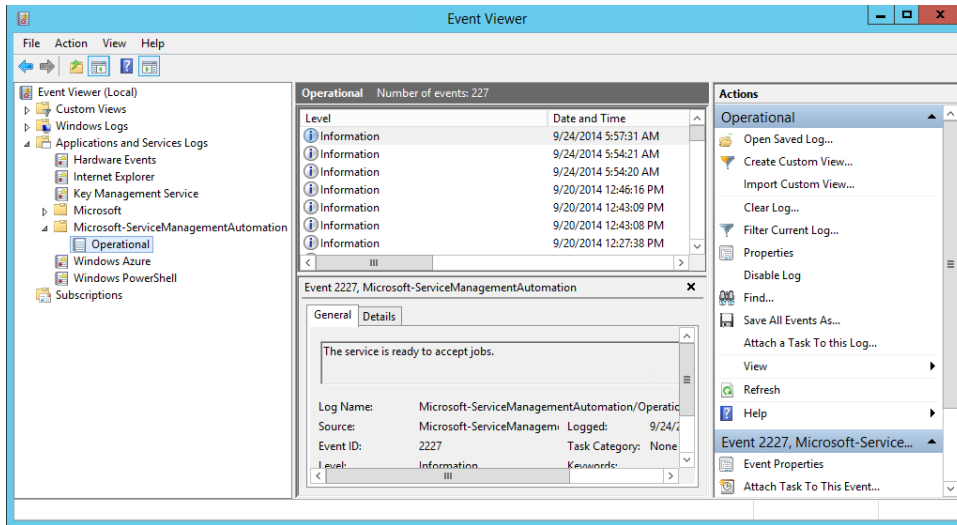
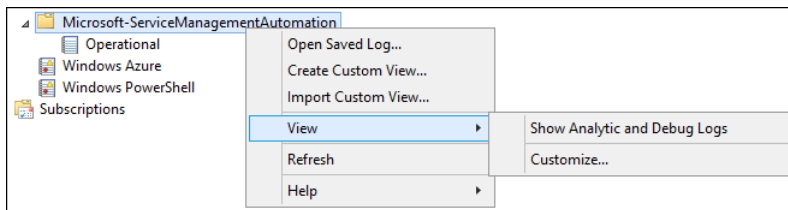


FIGURE 5-6 Service Management Automation event log

If you require more detailed event log data, you can enable the Analytic log, which provides extensive logging information. However, enabling this log should be reserved for troubleshooting very complex issues. To enable the Analytic log, follow these steps:

1. From the Event Viewer window, select the Microsoft-ServiceManagementAutomation folder in the Event Viewer tree view.
2. Right-click the folder, click View, and then click Show Analytic And Debug Logs.



3. The Analytic log appears in the Microsoft-ServiceManagementAutomation folder. Select the log, right-click it, and then click Enable Log.

Configuration files

Generally, with products such as Service Management Automation, you should not directly edit application configuration files. You should let them be managed by the application itself. However, knowing the location of configuration files and the settings they store can help you understand how the application functions.

A Service Management Automation system stores configuration files in one main location. On a server running the runbook worker role, locate the application installation directory, which by default is located in C:\Program Files\Microsoft System Center 2012 R2\Service

Management Automation\). In this folder, you will find several .config files. The most useful to review is the Orchestrator.Settings.config file, which contains information such as SQL Server details, job settings, and sandbox settings.

Certificates

During installation, the Service Management Automation web service creates a self-signed certificate by default (unless you pre-stage a certificate for use by the web service) and assigns the certificate to the web services site in Internet Information Services (IIS). The default self-signed certificate will not be trusted by other computers and services attempting to use the web service and has a lifetime of two years from creation. It is always a best practice to replace a self-signed certificate with a certificate from an internal or external certificate authority that is trusted by your computers and services.

Groups and service accounts

The web service and runbook worker roles of Service Management Automation both utilize a service account. The account used can be the same for both, but it is better for service accounts to leverage a separate account per role to help identify issues later.

During the installation of each role, the service account you provide is granted access to the Service Management Automation database with dbowner privileges and is granted the Logon as a Service right on the local server. Often, security policies override the Logon as a Service privilege by centrally managed configurations, so keep this in mind if your Service Management Automation system is not functioning.

On the runbook worker role, the service account is used as the logon account for the Runbook Service. On the web service role, the service account is used as the logon identity on the web services application pool.

During the installation of the web service role, a local group called smaAdminGroup is created on the server. This group is used to grant access accounts using the web service. By default, during the installation of the web service, only the service account specified during installation is added to the local group. Best practice is to place a group from your Active Directory domain in this local group and use the domain group for managing access to your Service Management Automation web service.

This page intentionally left blank

Examples of automation scenarios

This chapter explores several scenarios you may come across when using the Microsoft automation solutions and shows how to create the building blocks you can use to fast track your runbook development. Expanding on some topics covered throughout this book, this chapter addresses the following:

- Logging to an external file in a Service Management Automation system
- Logging to Table storage in an Azure Automation solution
- Creating an automation databus for sharing complex data structures between parent and child runbooks
- Using source control tools and processes for continuous integration to the Microsoft automation solutions

Logging to a local file

You can use Service Management Automation to send alternative streams of output to a tracing log, which can be used for troubleshooting and compliance needs. With the tracing log, you can use the custom stream to send user-defined data to a log file to determine where a fault is occurring within a runbook if the built-in logging capability is not sufficient. Writing to a tracing log creates a permanent record of the runbook output and can be as verbose as required without any chance of performance issues on the Service Management Automation database.

The Write-Log function in Listing 6-1 logs simple string messages, along with other important metadata such as an error-level setting (e.g., INFO, WARNING, or ERROR) and time and date stamps. When viewed with a tool such as the Configuration Manager Trace Log Viewer (CMtrace.exe), the trace log is color coded and easy to read (see Figure 6-1).

NOTE If you don't already use CMTrace for log file viewing, you can find it in the System Center 2012 R2 Configuration Manager Toolkit available at <http://www.microsoft.com/en-us/download/details.aspx?id=36213>.

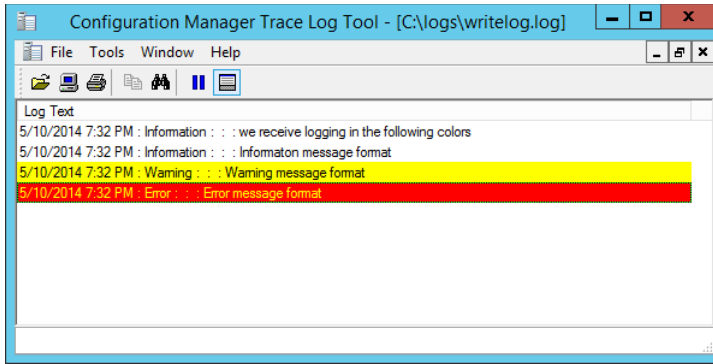


FIGURE 6-1 Using CMTrace to view trace output log files

In the following Service Management Automation example, first the location of the log file is constructed, along with a timestamp. With these variables constructed, a simple call to an Add-Content cmdlet can be run with the formatted text to be written (see Listing 6-1).

Listing 6-1 Basic Write-Log function to output messages to a text file

```
function Write-Log
{
    param
    (
        [parameter(Mandatory=$true)]
        [String]$LogFolderPath,
        [parameter(Mandatory=$true)]
        [String]$RunbookName,
        [parameter(Mandatory=$true)]
        [String]$ErrorCode,
        [parameter(Mandatory=$true)]
        [String]$Message
    )

    $LogFilePath = "$LogFolderPath\$RunbookName.log"
    $date = (Get-Date -Format g).ToString()
    Add-Content $LogFilePath "$date : $ErrorCode : $RunbookName : $Message"
}
```

To consume the Write-Log function within a runbook, use the following command:

```
Write-Log -LogFolderPath "C:\Logs " -RunbookName "Hello-World" -ErrorCode "INFO"
-Message "You successfully logged output to a file."
```

This simple example shows another option for logging runbook output that can be used as a permanent record of actions.

Logging to Table storage

Chapter 4, “Implementing automation,” introduced the concept of Integration Modules, which store common or portable code that can be re-used. To log to an external system from an Azure Automation runbook, you can create an Integration Module that writes log data to Table storage, one of the Azure Storage services. Table storage provides a persistent, cloud-based location to keep log data, which you can access from anywhere using tools such as the Microsoft Excel Power Query add-in or any of the free Azure Storage viewing tools. Note that this Integration Module also functions in Service Management Automation if you need to log output to Table storage.

NOTE To download an extended version of this Integration Module, which includes additional connection options and key-value pair data entry, refer to <http://gallery.technet.microsoft.com/scriptcenter/Azure-Storage-Tables-a7366266>.

A new Integration Module requires a custom connection type to store your Azure Storage account name and access key. Secure the Azure Storage account key and do not keep it in clear text anywhere; with this key, anyone can access your data and use your Azure Storage account from anywhere, possibly with great cost to you. In this example, the `AzureStorageTables-Automation.json` file is defined with the `StorageAccountName` and `StorageAccountKey` settings (see Listing 6-2). Note that the `StorageAccountKey` setting is defined as `IsEncrypted`, meaning its value will not be displayed in the Azure Automation interface and it will be stored as an encrypted value in the database.

Listing 6-2 Contents of the `AzureStorageTables-Automation.json` file

```
{
  "ConnectionFields": [
    {
      "IsEncrypted": false,
      "IsOptional": false,
      "Name": "StorageAccountName",
      "TypeName": "System.String"
    },
    {
      "IsEncrypted": true,
      "IsOptional": false,
      "Name": "StorageAccountKey",
      "TypeName": "System.String"
    }
  ],
  "ConnectionTypeName": "AzureStorageTables",
  "IntegrationModuleName": "AzureStorageTables"
}
```


Next, the AzureStorageTables.psm1 module file is created, including the following:

- **New-HMACSignature** A private function used to calculate a signature hash that is used to authenticate to Table storage
- **Write-AzureTableLog** A cmdlet used to write a log message to an instance of Table storage
- **New-AzureTableLog** A cmdlet used to create a new instance of Table storage for logging to

At the time of writing, the current release of the Azure PowerShell module includes support for creating and listing storage tables in a storage account but not the ability to manipulate table data, such as adding rows and columns. Therefore, this example uses the Table Service REST API to access the Table storage in the storage account.

See also For more information on the Table Service REST API, refer to <http://msdn.microsoft.com/en-us/library/azure/dd179423.aspx>.

To use the REST API, you first need to authenticate to the service. Whenever a request is sent to the service, details of the request must be signed by an HMAC signature. To perform the hash, use the storage account key. When the service receives the request, it performs the same hashing process and compares the hash delivered with the one generated. If the two hashes match, the request succeeds authentication and is processed. The cmdlets in the example Integration Module will use the private function shown in Listing 6-3 to create the required signature hash, and that signature will be included in each REST request.

Listing 6-3 Private function for calculating a signature hash

```
function New-HMACSignature
{
    <#
        .SYNOPSIS
        Create a HMAC256 Hash
        .DESCRIPTION
        Creates an HMAC256 Hash of a Table storage signature, used to authenticate to
        Table storage REST web service
    #>
    [OutputType([string])]
    param
    (
        [parameter(Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [string]$TableName,
        [parameter(Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [string]$AccountName,
        [parameter(Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
    )
}
```

```

        [string]$AccountKey,
        [parameter (Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [string]$TimeNow
    )

    # Create signature to be signed
    $signature = @{}
    $signature = "POST$([char]10)$([char]10)"
    $signature += "application/json$([char]10)"
    $signature += "$TimeNow$([char]10)"
    $signature += "/" + $AccountName + "/" + $TableName

    # Sign signature
    $accountKeyBytes = [System.Convert]::FromBase64String($AccountKey)
    $HMACSHA256 = New-Object
        System.Security.Cryptography.HMACSHA256(, $accountKeyBytes)
    $signatureUTF8 = [System.Text.Encoding]::UTF8.GetBytes($signature)
    $hmacHash = $HMACSHA256.ComputeHash($signatureUTF8)

    # Convert signature to Base64 and output
    $hmacSignature = [System.Convert]::ToBase64String($hmacHash)
    $hmacSignature
}

```

The primary cmdlet for the example Integration Module is shown in Listing 6-4. This is used to send logging messages to Table storage. This cmdlet performs the following actions:

1. Consumes parameters provided when the cmdlet is called.
2. Expands the Connection object that is provided as a hashtable. This Connection object is the object defined in the AzureStorageTables-Automation.json file created earlier.
3. Creates a timestamp for the message.
4. Requests an authentication hash from the New-HMACSignature function.
5. Constructs a hashtable of HTTP headers and a hashtable of the message body.
6. Sends the REST request and validates success.

Listing 6-4 Cmdlet used to write a log message to Table storage

```

function Write-AzureTableLog
{
    <#
        .SYNOPSIS
        Write log message to Table storage
        .DESCRIPTION
        Write log message to Table storage using the details provided.
    >

```

Table - The name of the table to log the message to. If the table does not exist then it will be created.

Message - The text value of the message to be logged.

Type - A text value describing the type of message to log, e.g., INFO / WARNING / ERROR

RunbookName - The name of the runbook that generated the log. This value is used as the PartitionKey in the table.

JobID - The value of the JobID of the instance of the Runbook that was running.

Connection - A hashtable returned from the Get-AutomationConnection activity, containing the AzureStorageTables connection object.

Requires a valid AzureStorageTables connection object.

```
#>
[CmdletBinding(HelpURI='http://aka.ms/automate')]
[OutputType([bool])]
param
(
    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [string]$Table,
    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [string]$Message,
    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [string]$Type,
    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [string]$RunbookName,
    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [string]$JobID,
    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [Hashtable]$Connection
)

$AccountName = $Connection.StorageAccountName
$AccountKey = $Connection.StorageAccountKey

$rowKey = [GUID]::NewGuid()

# Get current time
$timeNow = [System.DateTime]::UtcNow.ToString("r")
```

```

# Get hash signature for authentication to Table storage
$hash = New-HMACSignature -TableName $Table -AccountName $AccountName
    -AccountKey $AccountKey
    -TimeNow $timeNow
Write-verbose $hash -Verbose

# Create headers for request
$baseURI = "https://$AccountName.table.core.windows.net/$Table"
$authorizationHeader = ("SharedKey {0}:{1}" -f $AccountName,$hash)
write-verbose $authorizationHeader -Verbose

$headers = @{}
$headers.Add("Authorization",$authorizationHeader)
$headers.Add("Accept","application/json;odata=fullmetadata")
$headers.Add("Accept-Charset","UTF-8")
$headers.Add("Content-Type","application/json")
$headers.Add("MaxDataServiceVersion","3.0;NetFx")
$headers.Add("x-ms-date",$timeNow)
$headers.Add("x-ms-version","2014-02-14")
write-verbose $headers -Verbose

# Create body for request
$body = @{}
$body.Add("Type",$Type)
$body.Add("Message",$Message)
$body.Add("JobID",$JobID)
$body.Add("PartitionKey",$RunbookName)
$body.Add("RowKey",$rowKey)
$body = $body | ConvertTo-Json
write-verbose $body -Verbose

# Send REST request
try
{
    $output = Invoke-RestMethod -Method POST -Uri $baseURI -Headers $headers
    -Body $body
    write-verbose $output -Verbose
    return $true
}
catch [System.Net.WebException]
{
    If($_.Exception.Message -like "(404) Not Found")
    {
        # Table not created, create it now and retry
        New-AzureTableLog -Table $Table -Connection $Connection
        Write-AzureTableLog -Table $Table -Message $Message -Type $Type
    }
}

```

```

        -RunbookName $RunbookName -JobID $JobID -Connection $Connection
    }
}
catch
{
    Write-Output "Unknown error occurred."
    return $false
}
}

```

You may have noticed in the Write-AzureTableLog cmdlet that when the request is sent, it is wrapped in a Try/Catch block. This error handler checks the result of the request, and if it returns a message like "404 Not Found," you can assume the table has not yet been created. To create the table, call the final cmdlet, New-AzureTableLog (see Listing 6-5). This cmdlet performs the same types of activities as Write-AzureTableLog; however, in this case, rather than sending a log message, the body of the cmdlet sends a request to create the table you are trying to write to.

Listing 6-5 Cmdlet used to create a logging table

```

function New-AzureTableLog
{
    <#
        .SYNOPSIS
        Create Azure Storage Table
        .DESCRIPTION
        Creates a new Azure Storage Table using the name provided.

        Table - The name of the table to log message to. If the table does not exist
        then it will be created.
        Connection - A hashtable returned from the Get-AutomationConnection
        activity, containing the AzureStorageTables connection object.

        Requires a valid AzureStorageTables connection object.
    #>
    [CmdletBinding(HelpURI='http://aka.ms/automate')]
    [OutputType([string])]
    param
    (
        [parameter(Mandatory=$true)]
        [string]$Table,
        [parameter(Mandatory=$true)]
        [Hashtable]$Connection
    )

    $AccountName = $Connection.StorageAccountName

```

```

$AccountKey = $Connection.StorageAccountKey

# Get current time
$timeNow = [System.DateTime]::UtcNow.ToString("r")

# Get hash signature for authentication to Table storage
$hash = New-HMACSignature -TableName "Tables" -AccountName $AccountName
    -AccountKey $AccountKey -TimeNow $timeNow
write-verbose $hash -Verbose

# Create headers for request
$baseURI = "https://$AccountName.table.core.windows.net/Tables"
$authorizationHeader = ("SharedKey {0}:{1}" -f $AccountName,$hash)
write-verbose $authorizationHeader -Verbose

$headers = @{}
$headers.Add("Authorization",$authorizationHeader)
$headers.Add("Accept","application/json;odata=fullmetadata")
$headers.Add("Accept-Charset","UTF-8")
$headers.Add("Content-Type","application/json")
$headers.Add("MaxDataServiceVersion","3.0;NetFx")
$headers.Add("x-ms-date",$timeNow)
$headers.Add("x-ms-version","2014-02-14")
write-output $headers

# Create body for request
$body = @{}
$body.Add("TableName",$Table)
$body = $body | ConvertTo-Json
write-verbose $body -Verbose

# Send REST request
try
{
    $output = Invoke-RestMethod -Method POST -Uri $baseURI -Headers $headers
    -Body $body
    write-verbose $output -Verbose
    return $true
}
catch [System.Net.WebException]
{
    If($_.Exception.Message -like "(404) Not Found")
    {
        # Table not created, create it now and retry
        write-verbose $_.Exception.Message -Verbose
        New-AzureTableLog -Table $Table -Connection $Connection
    }
}

```

```

        Write-AzureTableLog -Table $Table -Message $Message -Type $Type
        -RunbookName $RunbookName -JobID $JobID -Connection $Connection
    }
}
catch
{
    Write-Output "Unknown error occurred."
    Write-Verbose $_.Exception.Message -Verbose
    return $false
}
}

```

To complete the Integration Module .psm1 file and make the cmdlets visible, call the `Export-ModuleMember` cmdlet to publish `Write-AzureTableLog` and `New-AzureTableLog` by using the following command:

```
Export-ModuleMember Write-AzureTableLog, New-AzureTableLog
```

Finally, use the `New-ModuleManifest` cmdlet to create the Module Definition file by using the following command:

```
New-ModuleManifest -Path <Path to AzureStorageTables.psd1>
-RootModule <Path to AzureStorageTables.psm1>
```

When the required files have been created and saved in a folder called `AzureStorageTables`, you can compress the folder as a .zip file and upload it as an Integration Module asset to Azure Automation.

After you import the Integration Module, it will be visible in the Assets view of the management portals (see Figure 6-2), and you will be able to use the defined cmdlets as activities in your runbooks.

NAME	TYPE	LAST UPDATE
Azure	Module	21/08/2014 8:57:34 AM
AzureStorageTables	Module	30/09/2014 4:16:09 PM

FIGURE 6-2 The `AzureStorageTables` Integration Module

To use the `AzureStorageTables` Integration Module in a runbook, simply call the activities in your runbook workflow or from within an `InlineScript` script block. Because the Integration Module is known to Azure Automation, you do not need to explicitly import the runbook using the `Import-Module` cmdlet. Listing 6-6 demonstrates a basic Hello-World runbook that uses the `Write-AzureTableLog` activity.

Listing 6-6 Sample runbook using the AzureStorageTables Integration Module

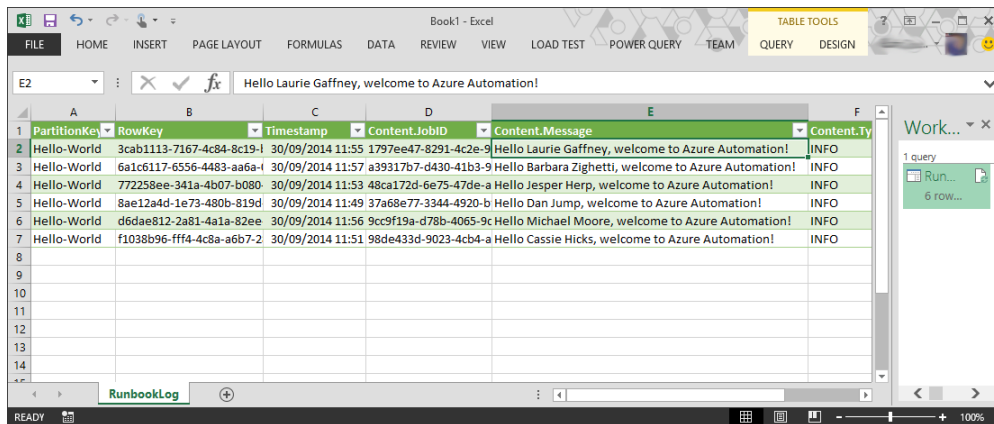
```
workflow Hello-World
{
    [OutputType([string])]
    param
    (
        [parameter(Mandatory=$true)]
        [string]$firstName,
        [parameter(Mandatory=$true)]
        [string]$lastName
    )

    $conn = Get-AutomationConnection -Name 'AzureStorageConnection'
    $RunbookName = "Hello-World"
    $logsTableName = "RunbookLogs"

    $fullName = "$firstName $lastName"

    Write-AzureTableLog -Connection $conn `
        -JobID $PSPriateMetaData.JobID.GUID `
        -Message "Hello $fullName, welcome to Azure Automation!" `
        -RunbookName $RunbookName `
        -Table $logsTableName `
        -Type "INFO"
}
```

You can view the runbook log in Table storage using Microsoft Power Query for Excel 2013 (see Figure 6-3).



PartitionKey	RowKey	Timestamp	Content.JobID	Content.Message	Content.Type
Hello-World	3cab1113-7167-4c84-8c19-1	30/09/2014 11:55	1797ee47-8291-4c2e-9	Hello Laurie Gaffney, welcome to Azure Automation!	INFO
Hello-World	6a1c6117-6556-4483-aa6a-1	30/09/2014 11:57	a39317b7-d430-41b3-9	Hello Barbara Zighetti, welcome to Azure Automation!	INFO
Hello-World	772258ee-341a-4b07-b080-	30/09/2014 11:53	48ca172d-6e75-47de-a	Hello Jesper Herp, welcome to Azure Automation!	INFO
Hello-World	8ae12a4d-1e73-480b-819d-	30/09/2014 11:49	37a69e77-3344-4920-b	Hello Dan Jump, welcome to Azure Automation!	INFO
Hello-World	d6dae812-2a81-4a1a-82ee-	30/09/2014 11:56	9cc9f19a-d78b-4065-9c	Hello Michael Moore, welcome to Azure Automation!	INFO
Hello-World	f1038b96-ff4-4c8a-a6b7-2	30/09/2014 11:51	98de433d-9023-4cb4-a	Hello Cassie Hicks, welcome to Azure Automation!	INFO

FIGURE 6-3 Runbook log imported from Table storage to Excel

NOTE To download Microsoft Power Query for Excel 2013, refer to <http://office.microsoft.com/en-us/excel/download-microsoft-power-query-for-excel-FX104018616.aspx>.

See also For guidance on using Microsoft Power Query for Excel 2013 to access Table storage, refer to <http://office.microsoft.com/en-us/excel-help/connect-to-microsoft-azuretable-storage-HA104122607.aspx?CTT=5&origin=HA104003952>.

Creating an automation databus

If you are familiar with System Center Orchestrator, you have likely used the built-in databus capability, which enables you to publish and subscribe to data throughout activities in your Orchestrator runbooks. Because each Orchestrator runbook occupies its own process, passing data between runbooks requires defining output and input types, and, overall, the databus model works well.

The Microsoft automation solutions provide similar capabilities given they are based on Windows PowerShell Workflow. Defining variable objects, collecting data, and reusing variables throughout a runbook is common practice. Even passing data between runbooks is straightforward with the use of output types and input parameters. However, how to pass complex objects between runbooks can be less than obvious when handling data flows in the Microsoft automation solutions.

You may wonder why you can't pass any object around since it's all PowerShell. Sometimes you can and sometimes you cannot. Chapter 4 introduced the concept of invoking a child runbook from either an inline call or the Start-SMARunbook cmdlet. When you call a child runbook using an inline call, when the parent runbook is started, the runbook worker service parses the workflow for inline calls and compiles all child runbooks into its workflow. Given this compilation process, the child runbook workflow runs in the same process as the parent, allowing complex objects to be passed between the parent and child workflows. Because the Microsoft automation solutions support a limited number of input parameter types on a runbook, the only caveat when passing complex types is the need to re-cast the objects when transferred to child runbooks.

The examples shown in Listing 6-7 demonstrate a very basic parent and child runbook that passes an XML object from the parent to the child using an inline call.

Listing 6-7a Parent runbook loading an XML object and passing it to child via inline call

Workflow Parent-Runbook

```
{
    $parentXml = [XML](Get-Content -Path "C:\<path>\<file>.xml")
    Child-Runbook -data $parentXml
}
```

Listing 6-7b Child runbook accepting input parameter and re-casting it as an XML object

Workflow Child-Runbook

```
{
    param
    (
        [parameter(Mandatory=$true)]
        [object]$data

    InlineScript
    {
        $childXml = [XML]$Using:data
        write-output $childXml.InnerXml.ToString()
    }
}
```

In the case of the Start-SMARunbook cmdlet, the PowerShell cmdlet calls the child runbook via the web service. Any request made to the web service, that contains objects must be able to serialize the objects as JSON. As a result, attempting to pass complex object types that can't convert to JSON objects results in an exception, for example, if the child runbook in the previous example was called using the Start-SMARunbook cmdlet. The examples shown in Listing 6-8 demonstrate a very basic parent and child runbook that passes a JSON object from parent to child using the Start-SMARunbook cmdlet.

Listing 6-8a Parent runbook creating a hashtable object, converting to JSON, and passing to a child via Start-SMARunbook

Workflow Parent-Runbook

```
{
    $parentHashtable = @{"Firstname"="Dan","Lastname"="Jump"}
    $parentJson = ConvertTo-Json -InputObject $parentHashtable
    Child-Runbook -Start-SMARunbook -Name Child-RB1 -Parameters @{"data"=$parentJson}
    -WebServiceEndpoint "https://sma.contoso.com/"
}
```

Listing 6-8b Child runbook accepting input parameter and converting it to a hashtable

Workflow Child-Runbook

```
{
    param
    (
        [parameter(Mandatory=$true)]
        [object]$data
    )

    $childObject = ConvertFrom-Json -InputObject $data
    $childHashtable=@{}
    $childObject.PSObject.Properties | foreach{$childHashtable."${$_.Name}" = $_.Value}
    write-output $childHashtable
}
```

Note in the previous example that the child runbook accepts the parameter as type “object.” Because the object has been passed via the web service and serialized, the incoming object must be converted to the type you need. In this case, the properties of the incoming object are enumerated and a hashtable based on these properties is created. This is important to understand because in this method of passing data between runbooks, the object’s data is passed, not the object itself. This is because the object is flattened for JSON, which is essentially a string, and the object’s methods are removed. Upon receipt of the object data, the child runbook must construct a new object based on the data.

For example, if the parent runbook receives an object that represents a Windows service using the PowerShell Get-Service cmdlet, then converts the result to JSON and passes it to another runbook via the Start-SMARunbook cmdlet, the resulting object is not the original object. It contains only the data of the object at that point. For the child runbook to interact with the service, it needs to create a new object based on the data received, such as the service’s name. The following example demonstrates this scenario. (These commands are all available as Scenario 6-1 in the Companion Content for this book). First, an object of the Time Service is created:

```
PS C:\> $timeSvcParent = Get-Service -Name W32Time
```

```
PS C:\> $timesvcParent | fl
```

```
Name                : W32Time
DisplayName           : Windows Time
Status                : Running
DependentServices    : {}
ServicesDependedOn   : {}
CanPauseAndContinue  : False
CanShutdown          : True
CanStop               : True
ServiceType           : Win32ShareProcess
```

Next, the object is converted to JSON as required to pass to a child runbook:

```
PS C:\> $timeSvcParentJson = ConvertTo-Json -InputObject $timeSvc
PS C:\> $timeSvcParentJson
{
  "CanPauseAndContinue": false,
  "CanShutdown": true,
  "CanStop": true,
  "DisplayName": "Windows Time",
  "DependentServices": [

  ],
  "MachineName": ".",
  "ServiceName": "W32Time",
  "ServicesDependedOn": [

  ],
  "ServiceHandle": null,
  "Status": 4,
  "ServiceType": 32,
  "Site": null,
  "Container": null,
  "Name": "W32Time",
  "RequiredServices": [

  ]
}
```

Next, the received object is converted to JSON as required in a child runbook:

```
PS C:\> $timeSvcChild = ConvertFrom-Json -InputObject $timeSvcParentJson
PS C:\> $timeSvcChild.GetType()
```

IsPublic	IsSerial	Name	BaseType
True	False	PSCustomObject	System.Object

Note that the original object's type was a ServiceController:

```
PS C:\> $timeSvcParent.GetType()
```

IsPublic	IsSerial	Name	BaseType
True	False	ServiceController	System.ComponentModel.Component

To access the service again, a new object representing it must be created:

```
PS C:\> $newTimeSvcChild = get-service -Name $timeSvcChild.Name
PS C:\> $newTimeSvcChild.GetType()
```

IsPublic	IsSerial	Name	BaseType
True	False	ServiceController	System.ComponentModel.Component

Given what you now know, you can construct a generic approach to data sharing between runbooks that suits all scenarios, which can be called the automation databus. It must comply with the following rules:

1. The automation databus will be a hashtable that is created in each runbook.
2. A parent runbook will be responsible for adding data to the automation databus.
3. When calling a child runbook, the request must always provide the automation databus object as an input.
4. Each child runbook must have a mandatory parameter to accept the automation databus object.
5. Each child runbook must output to the parent any data it needs to be made available in the automation databus, and the parent runbook must perform adding data to the automation databus.

As in the earlier example of logging to Table storage, the following example creates a new Integration Module to store the automation databus code so it can be imported and used in the Microsoft automation solutions. This Integration Module, called AutomationDatabus, consists of only a Windows PowerShell module file (AutomationDatabus.psm1) and a Windows PowerShell manifest definition file (AutomationDatabus.psd1). This Integration Module allows you to use a databus object to pass data between parent and child runbooks (see Figure 6-4).

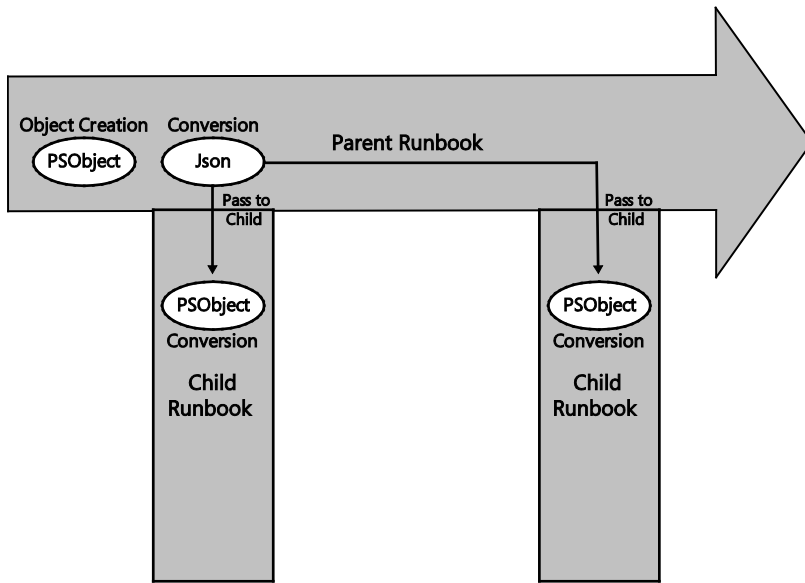


FIGURE 6-4 Runbooks leveraging databus capabilities

The AutomationDatabus.psm1 module file will consist of the following:

- An object template for the automation databus
- An Initialize-Databus cmdlet used to create the automation databus object in each runbook
- An Add-DatabusItem cmdlet used to add objects to the automation databus

To create the Integration Module, first define the automation databus object using a PObject object (see Listing 6-9). This is similar to defining a class in a C# program. From this object definition, you can create a consistent automation databus object as a copy in each runbook.

Listing 6-9 Defining a new object type for the automation databus

```
$databusClass = New-Object PObject -Property @{
    _parent = $null
    _date = $null
}
```

Next, create the Initialize-Databus cmdlet, which will be called at the start of each runbook (see Listing 6-10). This cmdlet will consume a databus as an optional parameter, used when initializing the automation databus in a child runbook. This allows data from a parent to be passed to the child. The cmdlet will then create a copy of the custom object type, `$databusClass`. If a databus parameter has been passed, the cmdlet will populate the child runbook's new automation databus with the data from the parent. If no databus parameter is

passed, the cmdlet assumes this is the parent runbook and creates a new blank automation databus. Finally, the cmdlet outputs the databus object to be used in the runbook.

Listing 6-10 Creating the cmdlet used to initialize the databus in each runbook

```
function Initialize-Databus
{
    <#
        .SYNOPSIS
            Create a Databus object
        .DESCRIPTION
            Creates a databus object and optionally imports a parent databus if supplied via
            the $InputDatabus parameter.
    #>
    [CmdletBinding(HelpURI='http://aka.ms/automate')]
    [OutputType([Object])]
    param
    (
        [Parameter(Mandatory=$false)]
        [Object]$InputDatabus
    )
    $databus = $databusClass.PSObject.Copy()

    If($InputDatabus -ne $null)
    {
        # Imports the existing parent databus into current scope.
        $parentDatabusObject = ConvertFrom-Json -InputObject $InputDatabus
        $parentDatabusObject.PSObject.Properties | ForEach{Add-Member
            -InputObject $databus
                - MemberType NoteProperty
            -Name $_.Name -Value $_.Value -Force}
    }
    else
    {
        # Creates a new databus and assumes running are in the Parent Runbook scope.
        $databus._parent = $ParentCommandName
        $databus._date = ((Get-Date).ToString())
    }
    $databus
}
```

Next, add a cmdlet to the Integration Module Add-DatabusItem, which will enable you to add objects to the databus from your runbooks (see Listing 6-11).

Listing 6-11 Creating the cmdlet used to add objects to the databus from the runbook

```
function Add-DatabusItem
{
    <#
        .SYNOPSIS
            Adds an object to a Databus
        .DESCRIPTION
            Adds a supplied object from the $InputObject parameter to the databus supplied
            via the $Databus parameter
    #>
    [CmdletBinding (HelpURI='http://aka.ms/automate')]
    [OutputType ([Object])]
    param
    (
        [Parameter (Mandatory=$true)]
        [Object] $Databus,
        [Parameter (Mandatory=$true)]
        [Object] $InputObjectName,
        [Parameter (Mandatory=$true)]
        [Object] $InputObject
    )

    Add-Member -InputObject $Databus -MemberType NoteProperty -Name $InputObjectName
        -Value $InputObject
    $databus
}
```

To complete the Integration Module .psm1 file and make the cmdlets visible, call the `Export-ModuleMember` cmdlet to publish `Initialize-Databus` and `Add-DatabusItem`:

```
Export-ModuleMember Initialize-Databus, Add-DatabusItem
```

Finally, use the `New-ModuleManifest` cmdlet to create the Module Definition file using the following command:

```
New-ModuleManifest -Path <Path to AzureStorageTables.psd1>
    -RootModule <Path to AzureStorageTables.psm1>
```

When the required files have been created and saved in a folder called `AutomationDatabus`, you can compress the folder as a .zip file and upload it as an Integration Module asset to Azure Automation.

After you import the Integration Module, it will be visible in the Assets view of the management portals (see Figure 6-5), and you will be able to use the defined cmdlets as activities in your runbooks.

NAME	↑	TYPE	LAST UPDATE
AutomationDatabus	→	Module	7/10/2014 3:40:23 PM
Azure		Module	2/10/2014 11:02:28 AM

FIGURE 6-5 The automation databus Integration Module

To use the automation databus Integration Module in your runbooks, simply call the activities in your runbook workflow. Because the Integration Module is known to Azure Automation, you do not need to explicitly import the runbook using the Import-Module cmdlet. The following example demonstrates a basic parent and child runbook, which uses the automation databus to pass object data from the parent and access it from the child. The new parent runbook created in Listing 6-12 will do the following:

1. Create a new automation databus object
2. Create a string object and a complex object from the Get-Service cmdlet
3. Add each object to the automation databus
4. Convert the automation databus to JSON and pass it to the child runbook

Listing 6-12 Sample parent runbook using the automation databus Integration Module

```
workflow Parent-Runbook
{
    $databus = Initialize-Databus

    $testText = "Hello Automation Specialists!"
    $databus = Add-DatabusItem -Databus $databus -InputObjectName "testText" -
InputObject $testText

    $testService = InlineScript{($testSvc = Get-Service | Where{$_.Name -eq "RpcSs"}) |
    convertto-json}
    $databus = Add-DatabusItem -Databus $databus -InputObjectName "testService"
    -InputObject $testService

    $db = ConvertTo-JSON -InputObject $databus
    ChildRB -parentDatabus $db
}
```

The new child runbook created in Listing 6-13 will do the following:

1. Create a new automation databus object, which will be passed to the parent databus object supplied from the parent runbook
2. Write the value of the string object
3. Convert the object from JSON so that it can be referenced correctly and write a value

Listing 6-13 Sample child runbook using the automation databus Integration Module

```
workflow ChildRunbook
{
    param
    (
        [parameter(Mandatory=$true)]
        [object]$parentDatabus
    )
    $databus = Initialize-Databus -InputDatabus $parentDatabus

    Write-Output "Value of the testText object:"
    InlineScript{$Using:databus.testText}

    Write-Output "Value of the testService object:"
    InlineScript{$testService = ConvertFrom-Json -InputObject
$Using:databus.testService.Value;$testService.DisplayName}
}
```

Now you can run a test of the Parent-Runbook runbook in the management portal and view the output to confirm the child runbook displays the data collected in the parent runbook (see Figure 6-6).

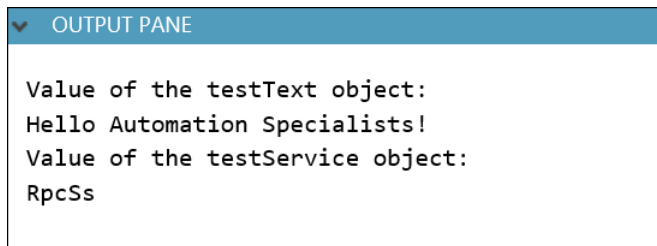


FIGURE 6-6 Runbook output when passing data using the automation databus

Using this example, you can pass object data between runbooks without having to explicitly worry about the serialization process. You can also pass as many objects as you need by using the single databus object rather than having to declare multiple parameters on each runbook. You could extend this example to include error handling routines, validation of data, or possibly even object re-inflation in child runbooks.

Continuous Integration Model

In any application lifecycle management project, the need to meet or exceed quality gates are a major requirement. The authoring of runbooks is no exception to this requirement. In complex scenarios, this becomes more pertinent: the more systems runbooks integrate with to perform tasks, the greater the risk of undesired results.

The typical model of development, testing, and then promotion into production environments will mitigate some of the risk of undesired results. Quickly promoting changes into production through continuous integration can remediate issues or add functionality without the runbook consumer having to wait long periods of time. Previous chapters covered various tools and authoring techniques you can leverage to provide such a capability. The model and process in this section show one approach to creating continuous deployment capability and will help you get started with runbook authoring and management.

The model shown in Figure 6-7 used in this section is agnostic to the tools you are using, whether they are Azure Automation and Visual Studio Online or Service Management Automation and Team Foundation Server. In both cases, applying the model provides you with a managed source control solution that allows promotion of runbooks between lifecycle levels.

The model consists of the following:

- A runbook authoring tool, as discussed in Chapter 5, “Managing runbooks”
- A source control solution containing a centralized repository of runbooks
- A local clone of the runbook repository, which is synchronized with the source control system
- For each lifecycle level in your environment (e.g., Development, Testing, Production), a branch of the runbook repository created in the source control solution
- A synchronization runbook in each lifecycle level that pulls runbooks from the appropriate branch of the runbook repository
- A log location to store details of runbook synchronization actions in each lifecycle level

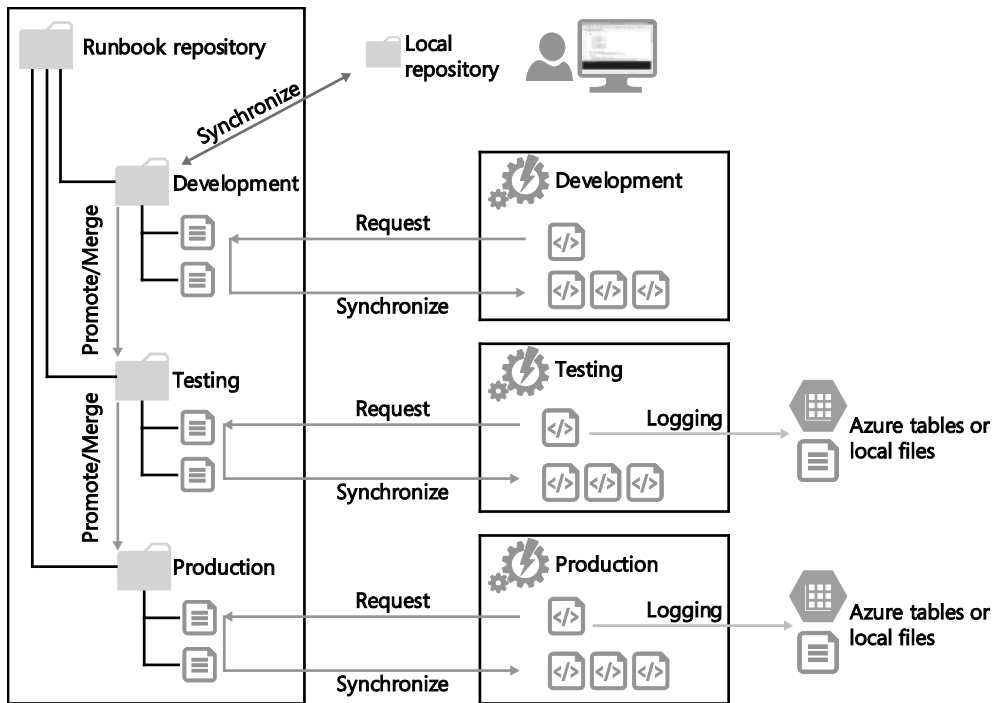


FIGURE 6-7 Continuous integration model

The change process in this model includes these steps:

1. Runbook authoring is performed through a local authoring tool or the management portal.
2. Runbooks stored in your local repository are synchronized to the source control solution in the development branch.
3. The runbook synchronization runbook in the development environment pulls all runbooks from the source control solution. At this point in the process, they can be tested and revised as required.
4. After completing development, a runbook can be merged into the testing branch using the source control solution. At this point in the process, the runbook synchronization runbook in the testing environment pulls all runbooks from the source control solution.
5. Runbooks can be tested and bugs logged as required. After runbook testing has been completed, the process can be repeated to merge the testing branch with the production branch. At this point, the runbook synchronization runbook in the production environment pulls all runbooks from the source control solution.

Note this model does not preclude the use of the management portals for runbook authoring; however, to maintain accurate records of changes in the source control solution,

you should copy any changes made via the management portals into your local repository and allow them to synchronize to the source control solution.

This book has already covered the tools and infrastructure you need to establish this kind of model, from the runbook authoring tools, to a source control solution, to the deployment of Azure Automation accounts and Service Management Automation systems. The piece that brings this type of model together, however, is the runbook synchronization processes, which fortunately have been created and published for you to use.

For an Azure Automation solution, the Sync-VsoGitRunbook example synchronizes runbooks from a Git repository in Visual Studio Online. For details of how to download and use this runbook, refer to <http://azure.microsoft.com/blog/2014/07/24/azure-automation-integrating-runbook-source-control-using-visual-studio-online/>.

For a Service Management Automation solution, the SMA-TFS integration runbooks provide the ability to synchronize with a Team Foundation Server system. For details of how to download and use these runbooks, refer to <http://blogs.technet.com/b/privatecloud/archive/2014/05/09/automation-mvp-spotlight-series-tfs-and-service-management-automation-better-together.aspx>.

About the authors



ROB COSTELLO is a Senior Consultant for Microsoft Consulting Services in Australia, with over 15 years of experience in the IT industry. He provides strategy and architecture leadership to Public Sector and Enterprise customers, as well as specializing in a wide range of on-premises and cloud-based Microsoft infrastructure technologies, including the Microsoft

Azure platform and System Center suite. Rob leads large technical teams delivering complex programs of work and maintains a passion for delivering high-quality solutions to his customers. Throughout his career, Rob has developed a love of automation technologies and seeks to use automation wherever possible to add value to his customer outcomes.



RICHARD MAUNSELL has been working with Microsoft technologies for the last 17 years. During this period, he has gained a deep insight into most aspects of Microsoft infrastructure products, deployment, and management. In recent years, this has evolved into automation of complex IT operations with various Microsoft toolsets and integration

with other IT systems.

About the series editor



MITCH TULLOCH is a well-known expert on Windows Server administration and virtualization. He has published hundreds of articles on a wide variety of technology sites and has written, contributed to, or been series editor for over 50 books. His free ebooks *Introducing Windows Server 2012* (Microsoft Press, 2012) and *Introducing Windows Server 2012 R2* (Microsoft Press, 2013) have together been downloaded over a million and a quarter times! For a complete list of free ebooks from Microsoft Press, visit the Microsoft Virtual

Academy at <http://www.microsoftvirtualacademy.com/ebooks>.

Mitch has repeatedly received the Microsoft Most Valuable Professional (MVP) award for his outstanding contributions to supporting the global IT community. He is a ten-time MVP in the technology area of Windows Server Software Packaging, Deployment & Servicing. You can find his MVP Profile page at <http://mvp.microsoft.com/en-us/mvp/Mitch%20Tulloch-21182>.

Mitch is also Senior Editor of WServerNews, a weekly newsletter focused on system administration and security issues for the Windows Server platform. With almost 100,000 IT pro subscribers worldwide, WServerNews is the largest and oldest-running Windows Server–focused newsletter in the world. Visit <http://www.wservernews.com> and subscribe to WServerNews today!

Mitch runs an IT content development business based in Winnipeg, Canada, that produces white papers and other collateral for the business decision maker (BDM) and technical decision maker (TDM) audiences. His published content ranges from white papers about Microsoft cloud technologies to reviews of third-party products designed for the Windows Server platform. Before starting his own business in 1998, Mitch worked as a Microsoft Certified Trainer (MCT) for Productivity Point.

For more information about Mitch, visit his website at <http://www.mtit.com> or follow Mitch on Twitter @mitchtulloch.



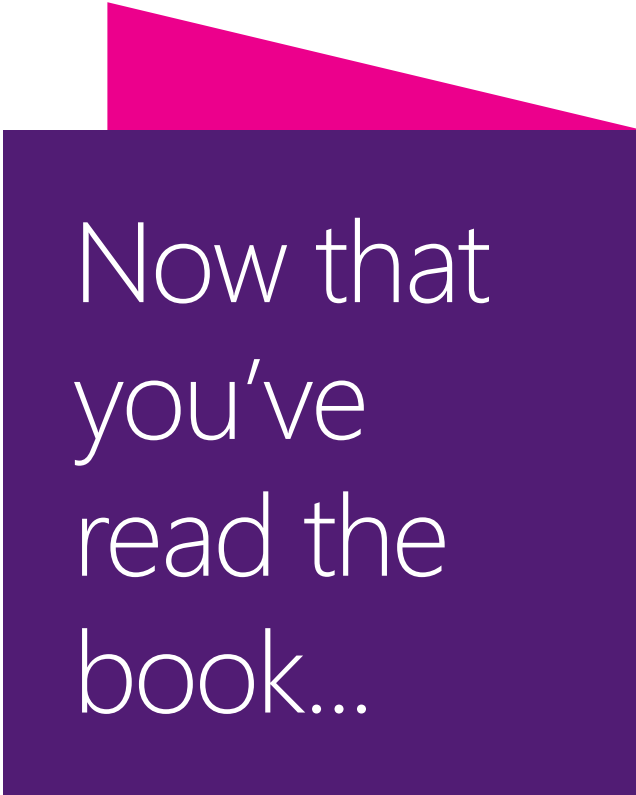
From technical overviews to drilldowns on special topics, get *free* ebooks from Microsoft Press at:

www.microsoftvirtualacademy.com/ebooks

Download your free ebooks in PDF, EPUB, and/or Mobi for Kindle formats.

Look for other great resources at Microsoft Virtual Academy, where you can learn new skills and help advance your career with free Microsoft training delivered by experts.

Microsoft Press



Now that
you've
read the
book...

Tell us what you think!

Was it useful?

Did it teach you what you wanted to learn?

Was there room for improvement?

Let us know at <http://aka.ms/tellpress>

Your feedback goes directly to the staff at Microsoft Press,
and we read every one of your responses. Thanks in advance!

