ExpressCluster® X *for Windows*

PP Guide (Virtualization Software)

February 25th, 2009
1st edition

Revision History

Edition	Revision Date	Description
1	February 25th, 2009	Created

Disclaimer

The contents of this document are subject to change without notice.
NEC Corporation assumes no responsibility for technical or editorial mistakes in or omissions from this document.
To obtain the benefits of the product, it is the customer's responsibility to install and use the product in accordance with this document.

Trademark Information

CONTENTS

INTRODUCTION

Target Readers and Purpose

ExpressCluster® PP Guide is intended for administrators who set up systems, system engineers who provide user support, and maintenance personnel for cluster systems.
This document introduces the software of which operations have been checked for an ExpressCluster environment. The software descriptions and setup examples in this document are provided only as a reference, and the software operations are not guaranteed.

Described Versions

This document describes the following ExpressCluster versions:

- ExpressCluster® X for Windows
- ExpressCluster® X for Windows

Organization of This Manual

SECTION I VIRTUALIZATION SOFTWARE

CHAPTER 1 Hyper-V: This chapter describes procedures for linking Hyper-V and ExpressCluster and provides related notes.

ExpressCluster Manuals

The three ExpressCluster manuals below are available. The title of each guide and a description of its role follows.

ExpressCluster X Getting Started Guide

This guide is intended for users of ExpressCluster and provides a product overview, a description of the operating environment, update information, and known problems.

ExpressClusterR X Install and Configuration Guide

This guide is intended for system engineers who install cluster systems using ExpressClusterR and for system administrators who maintain and operate installed cluster systems, and it describes requirements from for installing a cluster system using ExpressClusterR to for preparing to start operation. This guide follows the actual procedure for installing a cluster system to describe how to design a cluster system using ExpressCluster, how to install and set up ExpressCluster, how to check the system after setting it up, and how to evaluate the system before starting operation.

ExpressCluster X Reference Guide

This guide is intended for system administrators and system engineers who use ExpressCluster to install cluster systems, and it provides information about ExpressCluster operating procedures, the features of each module, maintenance, and troubleshooting. This guide is a companion to the *Install and Configuration Guide*.

Conventions

In this manual, the meanings of **Note**, **Important**, and **Related information** are as follows.

Note: This indicates information that is important but is not related to data loss or damage to the system or equipment.

Important: This indicates information that is necessary for avoiding data loss or damage to the system or equipment.

Related information: This indicates where to look for related information.

The following conventions are used in this document:

Notation	Meaning	Example
Square brackets ([ ])	These are used before and after command names and terms that appear on the GUI (including dialog boxes and menus).	Click [Start]. [Properties] dialog box
Square brackets ([ ]) within a command	These indicate that specifying the enclosed value is optional.	`clpstat -s[-h host_name]`
Monospace font (`courier`)	This indicates commands, functions, and parameters.	`clpstat -s`
Monospace bold font (**`courier`**)	This indicates text that the user actually enters at a command prompt.	Enter the following: **`clpcl –s -a`**
Monospace italic font (*`courier`*)	This indicates text that the user must replace with a valid value.	`clpstat -s[-h host_name]`

Obtaining the Latest Information

To obtain the latest product information, visit the following website.

http://www.nec.com/expresscluster

SECTION I Virtualization Software

This section describes settings that are required when linking the virtualization software and ExpressCluster, and provides related notes.

Chapter 1 Hyper-V

Terms Used in This Document

The meanings of the terms used in this document are as follows. Some terms may differ from the terms on Microsoft's Website[1], however, the meanings of the following terms are defined for this document as follows:

Term	Meaning
Physical machine	A server where Hyper-V is running.
Host OS	An operating system installed on a physical machine, Windows Server 2008, x64 Edition.
Virtual machine	A virtual server or client created on a physical machine.
Guest OS	An operating system installed on a virtual machine.
Integration service	Additional software to access Hypervisor from a virtual machine. When the guest OS is SUSE Linux[2], the integration service must be installed to enable network adapter and SCSI controller.

[1] http://www.microsoft.com/japan/windowsserver2008/technologies/hyperv.mspx

[2] Integration service for SUSE Linux supports only SUSE Linux Enterprise Server 10 with Service Pack 2.

Overview of Features

The following cluster configurations can be set up by linking Hyper-V and ExpressCluster.

Clustering between host OSs

Install ExpressCluster X on host OSs, and conduct clustering between physical machines. Not only general business applications but also a guest OS can be failed over. This chapter describes procedures for setting up a cluster in order to fail over a guest OS.

When a guest OS is a target for failover, ExpressCluster starts, stops, and monitors the guest OS. Applications on virtual machines do not need to be aware of the cluster.

When carrying out a scheduled shutdown, guest OSs can be migrated to a different physical machine without having the virtual machine shut down, similar to Quick Migration provided by Windows Server 2008 Failover Cluster (WSFC).

Clustering between guest OSs

Install ExpressCluster X on guest OSs, and conduct clustering between virtual machines. By targeting business applications running on guest OSs for failover, the availability of business applications on guest OSs can be enhanced.

The same functions as normal clustering between host OSs when business applications are failover targets are provided.

Operating Environment

- Hyper-V is supported only with x64 Edition of Windows Server 2008 Standard, Enterprise, Datacenter.

- To use Hyper-V functions, CPU must support Intel Virtualization Technology (Intel VT) or AMD Virtualization (AMD-V). In addition, Data Execution Prevention (DEP) must be enabled. Check if these functions are enabled in BIOS setting screen.

- This document describes the following operating systems.

  Host OS

  - Windows Server 2008 Standard, x64 Edition
  - Windows Server 2008 Enterprise, x64 Edition

  Guest OS (x86 and x64 versions of the following operating systems)

  - Windows Server 2003 Standard Edition SP2 or later
  - Windows Server 2003 Enterprise Edition SP2 or later
  - Windows Server 2003 Standard Edition R2 SP2 or later
  - Windows Server 2003 Enterprise Edition R2 SP2 or later
  - Windows Server 2008 Standard
  - Windows Server 2008 Enterprise
  - SUSE Linux Enterprise Server 10 with Service Pack 1
  - SUSE Linux Enterprise Server 10 with Service Pack 2

Notes

Notes on clustering between host OSs

- If Hyper-V Virtual Machine Management Service (`vmms`) has not started before executing a script (`vmstart.vbs`) to start a virtual machine, the startup of the virtual machine fails. Specify the startup retry count for virtual machine (`W#HyperV6`) in a parameter setting file `W#HyperV.bat` of `vmstart.vbs`.

- Disconnection of disk resource or mirror disk resource may fail when the failover group or cluster is stopped. Specify the disk disconnection error avoidance waiting time (`W#HyperV7`) in `W#HyperV.bat`.

Notes on clustering between guest OSs

- For setting up a shared disk cluster using clustering between guest OSs, a shared disk device (physical hard disk) is required. A virtual hard disk cannot be used as a shared disk device. In addition, logical drives (LUN) on the shared disk to be used as disk resource must be set to Offline in [Disk Management] on the physical machine.

- Shared disk cluster cannot be set up between guest OSs running on the same physical machine. Because a shared disk is locked during virtual machine startup, an error is output when starting a different virtual machine on the same physical machine, and the startup fails.

- When setting up a shared disk cluster or mirror disk cluster between instances of SUSE Linux Enterprise Server 10 with Service Pack 1, connect a virtual hard disk or physical hard disk to IDE controllers. By default, DVD drive is connected to IDE controller 1. Delete the DVD drive, and then connect a virtual hard disk or physical hard disk to IDE controller 1.

- When the cluster configuration is used, do not [Pause] or [Save] a virtual machine. If a virtual machine is paused or saved, ExpressCluster detects a heartbeat timeout, and starts a failover group on a different server. In this situation, both virtual machines where the failover group is running are shut down when executing [Startup] for the virtual machine that was paused or saved, in terms of data protection.

Installing Hyper-V

(1) Start up [Server Manager] from [Management Tools] of the [Start] menu, and then click [Add a Role].

(2) Select [Hyper-V] from Roles, and then click [Next].

(3) Notes are displayed. Click [Next] after reading the notes.

(4) Virtual network creation window appears. Select a network adapter[1] to be used for the virtual machine, and then click [Next].

(5) Click [Install].

(6) Restart the physical machine after the installation.

(7) If updates for Hyper-V are available, apply them as necessary.

[1] For ExpressCluster use, setting up two LAN systems as a heartbeat path between clusters is recommended. When setting up a cluster between guest OSs, prepare two network adapters to be used by virtual machines. In addition, when setting up a cluster between host OSs, assign the same virtual network name to the guest OS.

Creating a Virtual Machine

Note: When setting up a cluster between host OSs and creating a virtual machine, create a virtual machine following the procedure below after carrying out step (3) described in the next section *Setting Up Cluster between Host OSs*.
If virtual machines were already created, the following procedure can be skipped.

(1) Start up [Hyper-V Manager] from [Management Tools] of the [Start] menu.

(2) Right click a physical machine name displayed in Hyper-V Manager, and then click [Virtual Machine] from the [New] menu.

(3) Enter the name of the virtual machine in the [Name] field. Specify where to save the virtual machine configuration information, and then click [Next].

- For clustering between host OSs

  Save the configuration information on a shared disk or mirror disk.

- For clustering between guest OSs

  Save the configuration information to any location on a physical machine.

(4) Specify the memory assigned to the virtual machine, and then click [Next].

(5) Select a network adapter to be used for the virtual machine. Only one adapter can be selected here, however multiple adapters can be added using the virtual machine settings once a virtual machine is created.[1] After selecting a network adapter, click [Next].

(6) Specify the name of the virtual hard disk (vhd file) and where to save the virtual machine configuration information, and then click [Next].

- For clustering between host OSs

  Save the virtual hard disk on a shared disk or mirror disk.[2]

- For clustering between guest OSs

  Save the virtual hard disk to anywhere on a physical machine.

(7) Select the installation media, and then click [Next].

(8) Select [Start Virtual Machine for Creation], and then click [Finish].

(9) Guest OS installation starts. Install the guest OS.[3]

[1] When installing SUSE Linux Enterprise Server 10 with Service Pack 1, use [Legacy Network Adapter] instead of [Network Adapter].

[2] It can be saved on a shared disk or mirror disk, however, a virtual hard disk copying process will occur when exporting a virtual machine in the procedure described later.

[3] When installing SUSE Linux as a guest OS, also [C/C++ Compiler and Tools] and [Xen Virtual Machine Host Server] need to be installed. These are required to enable SUSE Linux Integration Service.

(10) Install the integration service.

- When a guest OS is Windows Server 2003/2008

  Log on the OS after a guest OS is installed. Click [Insert Integration Service Setup Disk] from the [Operate] menu. Integration service installation starts. Restart the virtual machine after the installation.

- When a guest OS is SUSE Linux

  Download Linux Integration Components for Microsoft Hyper-V from https://connect.microsoft.com/. Execute the downloaded `exe` file to deploy an `iso` image file and the installation guide. Install the integration service as instructed in the guide.

(11) Specify an IP address and computer name for the guest OS as necessary. In addition, perform disk setting and other settings when setting up a cluster between guest OSs.

Setting Up Cluster between Host OSs

In the procedure below, the failover group name, resource name, and monitoring resource name are specified as shown in the table below for convenience. Change these names in accordance with your environment as necessary.

Group, Resource, or Monitoring Resource	Name
Failover Group	`failover-vm`
Disk Resource or Mirror Disk Resource	`sd-vm`/`md-vm`
Script Resource	`script-vm`
Custom Monitoring Resource	`genw-vm`

(1) Install ExpressCluster X on each host OS as described in the *ExpressCluster X Install and Configuration Guide*.[1]

(2) Create a failover group with a disk resource (`sd-vm`) or mirror disk (`md-vm`) as described in the *ExpressCluster X Install and Configuration Guide*. In addition, check if the failover group can start up normally.

(3) Start up the failover group on the server where the failover target virtual machine resides.

(4) If a virtual machine have not been created yet, create a virtual machine as described in the previous section *Creating a Virtual Machine*.

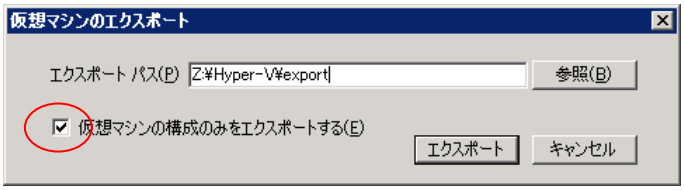
(5) Configure a network adapter and auto start action for the virtual machine.

A) Start up Hyper-V Manager. Right click the failover target virtual machine name, and then click [Configure].
B) Click [Network Adapter] under [Hardware] in the left pane. Select a network adapter to be assigned to the virtual machine from the pull-down menu of [Network]. Select [Static] for MAC address settings, and then assign a MAC address.
C) Click [Auto Start Action] under [Management] in the left pane. Select [Do nothing] as auto start action.
D) Click [OK] when the settings above are completed.

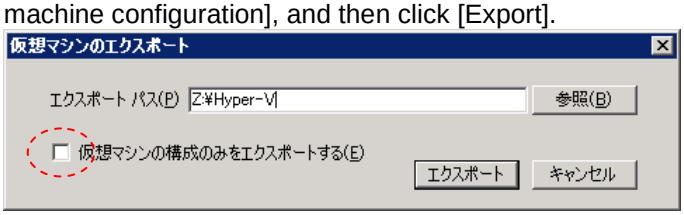
(6) Export a virtual machine to a shared disk or mirror disk.

A) If the virtual machine to be exported is running, shut down the virtual machine.
B) Right click the export target virtual machine name, and then click [Export].
   - When the virtual machine configuration information and virtual hard disk are on a shared disk or mirror disk
     Enter a folder path (for example, `Z:¥Hyper-V¥export`), other than the path to the folder where the virtual machine configuration information is saved on a shared disk or mirror disk, as an export path. Select [Export only virtual machine configuration], and then click [Export].

[1] Available from http://www.nec.co.jp/clusterpro.



- When the virtual machine configuration information and virtual hard disk are not on a shared disk or mirror disk
  Enter a path to a folder on a shared disk or mirror disk (for example, `Z:¥Hyper-V`) as an export path. Make sure to check [Export only virtual machine configuration], and then click [Export].



(7) Remove a virtual machine using Hyper-V Manager. Right click the exported virtual machine name, and then click [Remove].[1]

(8) Only when a virtual machine already saved on a shared disk or mirror disk was exported, the following steps are required. Copy the folder created under `Z:¥Hyper-V¥export` by exporting with step (6) above to the folder where the virtual machine configuration information was saved (for example, `Z:¥Hyper-V¥<virtual machine name>`). Check that each file and folder are copied as shown below.

- `config.xml` file exists under `Z:¥Hyper-V¥<virtual machine name>`.
- `<guest OS ID>.exp` file and a folder with the same name as the file exist under `Z:¥Hyper-V¥<virtual machine name>¥Virtual Machines`.

(9) Start up ExpressCluster Builder.

(10) Create a `VBScript` to start and stop a virtual machine from ExpressCluster. Create `vmstart.vbs`, `vmstop.vbs`, and `W#HyperV.bat`, as referring to sample scripts described at the end of this chapter, and save them to any location on the machine where ExpressCluster Builder is running.

(11) Create a `VBScript` to confirm virtual machine power status from ExpressCluster. Create `checkstatus.vbs` as referring to sample scripts described at the end of this chapter, and save it to any location on each physical machine (for example, `C:¥Program Files¥ExpressCluster¥bin`).

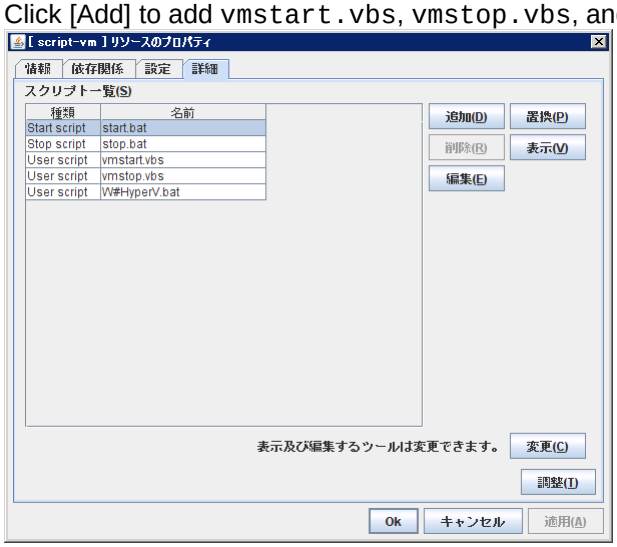
(12) Use ExpressCluster Builder to add the script resource to the failover group and edit the resource.

A) In the ExpressCluster Builder window, right click the name of the failover group (`failover-vm`) in the tree view, and then click [Add Resource].
B) Select [Script Resource] from [Type]. Specify a name for the script resource (`script-vm`), and then click [Next].
C) Select [Start script], and then click [Edit]. Editor starts, and then edit `start.bat` as referring to sample scripts described at the end of this chapter. Save the edited contents, and then close the editor.

[1] When virtual machine configuration information saved on other than a shared disk or mirror disk was exported, the virtual hard disk is no longer needed. Remove it manually.

D) Select [Stop script], and then click [Edit]. Editor starts, and then edit `stop.bat` as referring to sample scripts described at the end of this chapter. Save the edited contents, and then close the editor.

E) Click [Add] to add `vmstart.vbs`, `vmstop.vbs`, and `W#HyperV.bat`.



(13) Use ExpressCluster Builder to add a custom monitoring resource[1] to a cluster.

A) In the ExpressCluster Builder window, right click [Monitors] in the tree view, and then click [Add a Monitor Resource].

B) Select [Custom Monitoring] from [Type]. Specify a name for the custom monitoring resource (`genw-vm`), and then click [Next].

C) Click [Edit] to edit `genw.bat`. Editor starts, and then edit `genw.bat` as referring to sample scripts described at the end of this chapter. Save the edited contents and close the editor, and then click [Next].

D) Select [When Active] as [Monitoring Timing], and then click [Browse]. When a list of selectable resources appears, select the script resource (`script-vm`) you have just created.[2] Check that the script resource is specified for the target resource, and then click [Next].

E) Select the failover group name (`script-vm`) that includes the script resource (`failover-vm`) as the [Recovery Target], and then click [OK]. Edit parameters such as [Reactivation Threshold Value] as necessary[3], and then click [Finish].

(14) Stop the failover group (`failover-vm`) using WebManager or a `clpgrp` command.

(15) Suspend the cluster using WebManager or a `clpcl` command.

(16) Upload the configuration information created using ExpressCluster Builder. To upload the information, select [Upload Information File] from the Builder [File] menu.

---

[1] To use custom monitoring resource, updates (CPRO-XW020-01 or later) of EXPERSSCLUSTER must be applied.

[2] By this setting, the guest OS will be monitored immediately after the script is started, in other words, immediately after the guest OS is started. In addition, the guest OS monitoring will be stopped when the script resource stopping process is started, in other words, when the guest OS saving process is started.

[3] Adjust each parameter as referring to *ExpressCluster X Reference Guide, Chapter 7, Details of Monitoring Resource, Operations When Monitoring Resource Detects an Abnormality*.

Checking Operations of Cluster between Host OSs

(1) Resume the cluster using WebManager or a `clpcl` command.

(2) Start up the failover group using WebManager or a `clpgrp` command. On the server where the failover group is running, use Hyper-V Manager to check if the guest OS is running.

(3) Migrate the failover group using WebManager or a `clpgrp` command. On the server where the failover group was migrated to, use Hyper-V Manager to check if the guest OS is running.

(4) Shut down or reboot the physical machine where the failover group is running using WebManager or a `clpdown` command. At this point, use Hyper-V Manager to check that the failover group was migrated to a different server, and the guest OS is running.

(5) Check if a reactivation of recovery target or a failover is performed when `genw-vm` detects an abnormality caused by a guest OS shutdown using Hyper-V Manager. In addition, use Hyper-V Manager to check if the guest OS is restarted after a failover.

(6) Use Hyper-V Manager to check if other server detects a shutdown of a different server, starts a failover, and the guest OS is restarted when the physical machine is shut down from other than ExpressCluster.

(7) In addition to the above, perform items described in *ExpressCluster X Install and Configuration Guide, Chapter 7, Checking Operations, Performing Operation Verification Tests* as necessary.

Setting Up a Cluster between Guest OSs

(1) With Hyper-V Manager, right click a virtual machine where a disk resource or mirror disk resource is added to, and then click [Configure].

(2) Click [Add Hardware] of [Hardware] in the left pane, select [SCSI Controller], and then click [Add].

(3) Click [SCSI Controller] added to the list of [Hardware] in the left pane, and then click [Add].

- For disk resource

  Only physical hard disks[1] can be used as disk resource. Select a logical drive on the shared disk connected to the physical machine. Specify the same logical drive for all virtual machines.

- For mirror disk resource

  Either virtual or physical hard disks[2] can be selected as mirror disk resource.

(4) Start up the virtual machine after the disk is added.

(5) Install a guest OS supported by ExpressCluster on the virtual machine as described in *ExpressCluster X Install and Configuration Guide*.

(6) Set up a cluster using ExpressCluster Builder as described in *ExpressCluster X Install and Configuration Guide.*

(7) Upload the configuration information created using ExpressCluster Builder. To upload the information, select [Upload Information File] from the Builder [File] menu.

[1] To use a physical hard disk, the disk must be offline. Use [Disk Management] on the physical machine to check if the disk is offline.

[2] Check if the disk is offline as described above.

Checking Operations of Cluster between Guest OSs

(1) Start up the cluster using WebManager or a `clpcl` command.

(2) Migrate the failover group using WebManager or a `clpgrp` command. On the server where the failover group was migrated to, check if the failover group is started using WebManager or a `clpstat` command.

(3) Shut down or reboot the virtual machine where the failover group is running using WebManager or a `clpdown` command. At this point, check if the failover group is started on a different server using WebManager or a `clpstat` command.

(4) Use WebManager or a `clpstat` command to check if a server detects a shutdown of a partner server, and starts the failover group when the physical machine is shut down from other than ExpressCluster.

(5) In addition to the above, perform items described in *ExpressCluster X Install and Configuration Guide, Chapter 7, Checking Operations, Performing Operation Verification Tests* as necessary.

Sample Scripts

Sample scripts required to set up a guest OS failover cluster are shown below. Edit the bold text in `W#HyperV.bat` and `genw.bat` scripts for use in accordance with your environment.

vmstart.vbs

A script to start up a virtual machine.

```
'=====================================================================
' vmstart.vbs
'=====================================================================
' Title   : Import VM's config file and start the VM
' Authors : Wang Haoran
' Date    : 2009/02/16
' Revised : 2009/02/16
'=====================================================================

option explicit

dim objWMIService
dim managementService
dim fileSystem
dim objStdOut
const JobStarting = 3
const JobRunning = 4
const JobCompleted = 7
const wmiStarted = 4096
const wmiSuccessful = 0
const Enabled = 2
const Disabled = 3
const Suspended = 32769
' Instance for standard output
Set objStdOut = Wscript.StdOut

Main()


'---------------------------------------------------------------------
--
' Main
'---------------------------------------------------------------------
--
Sub Main()

   dim computer, objArgs, vmName, vm, importDirectory,
backupDirectory, vmGUID, vmDefaultRootDirectory
   dim FromPath, DestPath, ConfigXMLPath, BackupPath
   dim objManagementServiceList
   dim retry, conRetryMax, conInterval
   set fileSystem =
Wscript.CreateObject("Scripting.FileSystemObject")
   computer = "."

   set objArgs = WScript.Arguments

   if WScript.Arguments.Count = 6 then
      vmName = objArgs.Unnamed.Item(0)
      importDirectory = objArgs.Unnamed.Item(1)
      backupDirectory = objArgs.Unnamed.Item(2)
      vmGUID = objArgs.Unnamed.Item(3)
      vmDefaultRootDirectory = objArgs.Unnamed.Item(4)
      conRetryMax = CInt(objArgs.Unnamed.Item(5))
```

```
    else
   objStdOut.WriteLine "usage: cscript vmstart.vbs vmName
importDirectoryName backupDirectoryName vmGUID
vmDefaultRootDirectoryName vmmsWaitTime(second)"
       WScript.Quit(1)
    end if

    set objWMIService = GetObject("winmgmts:¥¥" & computer &
"¥root¥virtualization")

    conInterval = 1000
    for retry = 0 To conRetryMax
       set objManagementServiceList =
objWMIService.ExecQuery("select * from
Msvm_VirtualSystemManagementService")
       if objManagementServiceList.count = 0 then
          if retry < conRetryMax then
             objStdOut.WriteLine "Error! Virtual Machine Management
Service(VMMS) is not started, retry..."
          else
             objStdOut.WriteLine "Error! Virtual Machine Management
Service(VMMS) is not started."
             WScript.Quit(1)
          end If
       else
          objStdOut.WriteLine "Virtual Machine Management
Service(VMMS) has been started."
          Exit For
       end if
       WScript.Sleep(conInterval)
    next

    set managementService = objManagementServiceList.ItemIndex(0)
    FromPath = backupDirectory & "¥" & vmName
    DestPath = importDirectory

    'check if GuestOS(vmName) exist ,if exist destroy it
    if CheckVMIfExist(vmName, importDirectory, vmGUID,
vmDefaultRootDirectory, backupDirectory) then
       if fileSystem.FolderExists(FromPath) then
          set vm = GetComputerSystem(vmName)
          if DestroyVirtualSystem(vm) then
             objStdOut.WriteLine "Destroy previous vm done."
          else
             objStdOut.WriteLine "DestroyVirtualSystem Failed."
             WScript.Quit(1)
          end if
          'copy previous vm's config file to importDirectory
          fileSystem.CopyFolder FromPath, DestPath, true
       else
          objStdOut.WriteLine "error! not first time, but previous
backupd vm's config file doesn't exist!"
          objStdOut.WriteLine "Maybe backup file was deleted by user,
please check it."
          WScript.Quit(1)
       end if
    end if

    ConfigXMLPath = importDirectory & "¥" & "config.xml"
    if not fileSystem.FileExists(ConfigXMLPath) then
       'copy previous vm's config file to importDirectory
       fileSystem.CopyFolder FromPath, DestPath, true
    end if

    'edit config.xml
    UpdateConfigFile ConfigXMLPath
    objStdOut.WriteLine "UpdateConfigFile done."
```

```
    'clear previous vm that have the same GUID.
    ClearVmGuid vmDefaultRootDirectory, vmGUID

    'import VM's config file
    if ImportVirtualSystem(importDirectory, "false") then
        objStdOut.WriteLine "Import done."
    else
        objStdOut.WriteLine "ImportVirtualSystem Failed."
        WScript.Quit(1)
    end if

    set vm = GetComputerSystem(vmName)

    'objStdOut.WriteLine "importDirectory :" & importDirectory
    'objStdOut.WriteLine "backupDirectory :" & backupDirectory

    BackupPath = backupDirectory & "¥" & vmName
    'objStdOut.WriteLine "BackupPath :" & BackupPath

    if not fileSystem.FolderExists(BackupPath) then
        objStdOut.WriteLine "first time start, vm's configuration data
is backuping"
        'Clear VM's Scopes.
        ClearVMScopes(vm)
        'Backup vm's config file
        if BackupVMConfigFile(vm, backupDirectory, importDirectory)
then
            objStdOut.WriteLine "BackupVMConfigFile done."
        else
            objStdOut.WriteLine  "BackupVMConfigFile failed"
            WScript.Quit(1)
        end if
    end if

    'start VM
    if RequestStateChange(vm, "start") then
        objStdOut.WriteLine " start done."
   WScript.Quit(0)
    else
        objStdOut.WriteLine  "RequestStateChange failed"
        WScript.Quit(1)
    end if



End Sub

'-------------------------------------------------------------------
--
' Edit config.xml, set 'VmStateCopied' to 'true'
'-------------------------------------------------------------------
--
Sub UpdateConfigFile(ConfigFilePath)
    dim objDOM
    dim subNode
    dim rtResult

    Set objDOM = WScript.CreateObject("MSXML2.DOMDocument")

    ' Load copy source file
    rtResult = objDOM.load(ConfigFilePath)
    If rtResult = False Then
        objStdOut.WriteLine "Failed to load XML file " & ConfigFilePath
& " ."
   Set objDOM = Nothing
```

```
   WScript.Quit(1)
    End If

    objDOM.async = False
    set subNode =
objDOM.selectSingleNode("/configuration/VmStateCopied[@type =
'bool']")
    objStdOut.WriteLine Format1("VmStateCopied = {0}", subNode.text)
    subNode.text = "true"
    objStdOut.WriteLine Format1(" change VmStateCopied to '{0}'",
subNode.text)

    'Save the config.xml file
    objDOM.Save(ConfigFilePath)
    Set objDOM = Nothing

End Sub

'-------------------------------------------------------------------
--
' Retrieve Msvm_VirtualComputerSystem from base on its ElementName
'-------------------------------------------------------------------
--
Function GetComputerSystem(vmElementName)
    On Error Resume Next
    dim query
    query = Format1("select * from Msvm_ComputerSystem where
ElementName = '{0}'", vmElementName)
    set GetComputerSystem =
objWMIService.ExecQuery(query).ItemIndex(0)
    if (Err.Number <> 0) then
        objStdOut.WriteLine Format1("Err.Number: {0}", Err.Number)
        objStdOut.WriteLine
Format1("Err.Description:{0}",Err.Description)
        WScript.Quit(1)
    end if
End Function

'-------------------------------------------------------------------
--
' check if GuestOS vmName exist ,if exist destroy it
'-------------------------------------------------------------------
--
Function CheckVMIfExist(vmElementName, importDirectory, vmGUID,
vmDefaultRootDirectory, backupDirectory)
    'On Error Resume Next
    dim query, query2
    dim vmList, vmList2, vm
    dim count
    dim ConfigXMLPath, vmXMLPath, vmDefaultRootPath, BinFilePath,
VSVFilePath
    dim objFolder, objFiles, objFile

    CheckVMIfExist = false
    count = 0
    ConfigXMLPath = importDirectory & "¥" & "config.xml"
    vmXMLPath = importDirectory & "¥" & "Virtual Machines" & "¥" &
vmGUID & ".xml"
    vmDefaultRootPath = vmDefaultRootDirectory & "¥" & "Virtual
Machines" & "¥" & vmGUID & ".xml"
    objStdOut.WriteLine "vmXMLPath: " & vmXMLPath
    'set objFolder=filesystem.GetFolder(vmXMLDirectroy)
    'set objFiles=objFolder.Files
    'objStdOut.WriteLine "objFiles.count =" & objFiles.count

    Do
        query = Format1("select * from Msvm_ComputerSystem where
```

```
ElementName = '{0}'", vmElementName)
       set vmList = objWMIService.ExecQuery(query)

       objStdOut.WriteLine "vmList.count =" & vmList.count

       if vmList.count <> 0 then
          if (Err.Number <> 0) then
              objStdOut.WriteLine Format1("CheckVMIfExist
Err.Number: {0}", Err.Number)
              objStdOut.WriteLine Format1("CheckVMIfExist
Err.Description:{0}",Err.Description)
              WScript.Quit(1)
           end if
           CheckVMIfExist = true
           Exit Do
       elseif (not fileSystem.FileExists(ConfigXMLPath)) and
fileSystem.FileExists(vmXMLPath) then
           query2 = Format1("select * from Msvm_ComputerSystem where
ElementName = '{0}'", vmGUID)
           set vmList2 = objWMIService.ExecQuery(query2)
           objStdOut.WriteLine "vmList2.count =" & vmList2.count
           if vmList2.count <> 0 then
              objStdOut.WriteLine "Loop count: " & count & ", per loop
sleep 10 seconds."
              WScript.Sleep(10000)
           else
              Exit Do
           end if
       else
           objStdOut.WriteLine "CheckVMIfExist check OK(previous VM
doesn't exist)."
           Exit Do
       end if
   Loop

   if CheckVMIfExist = false and fileSystem.FileExists(vmXMLPath)
then
       fileSystem.DeleteFile vmXMLPath, true
       if fileSystem.FileExists(vmDefaultRootPath) then
          fileSystem.DeleteFile vmDefaultRootPath, true
       end if

       'delete previous <guid>.bin and <guid>.vsv file in backup folder
  BinFilePath = backupDirectory & "¥" & vmElementName & "¥" & "Virtual
Machines" & "¥" & vmGUID & "¥" & vmGUID & ".bin"
       VSVFilePath = backupDirectory & "¥" & vmElementName & "¥" &
"Virtual Machines" & "¥" & vmGUID & "¥" & vmGUID & ".vsv"
       'objStdOut.WriteLine "BinFilePath : " & BinFilePath
       'objStdOut.WriteLine "VSVFilePath : " & VSVFilePath
       if fileSystem.FileExists(BinFilePath) then
          fileSystem.DeleteFile BinFilePath, true
          objStdOut.WriteLine "delete previous vm's "& vmGUID & ".bin
file in backupDirectory success."
       end if
       if fileSystem.FileExists(VSVFilePath) then
          fileSystem.DeleteFile VSVFilePath, true
      objStdOut.WriteLine "delete previous vm's "& vmGUID & ".vsv file
in backupDirectory  success."
       end if

  'delete previous <guid>.bin and <guid>.vsv file in import folder
  BinFilePath = importDirectory & "¥" & "Virtual Machines" & "¥" &
vmGUID & "¥" & vmGUID & ".bin"
  VSVFilePath = importDirectory & "¥" & "Virtual Machines" & "¥" &
vmGUID & "¥" & vmGUID & ".vsv"
       if fileSystem.FileExists(BinFilePath) then
          fileSystem.DeleteFile BinFilePath, true
```

```
          objStdOut.WriteLine "delete previous vm's "& vmGUID & ".bin
file in importDirectory success."
       end if
       if fileSystem.FileExists(VSVFilePath) then
          fileSystem.DeleteFile VSVFilePath, true
      objStdOut.WriteLine "delete previous vm's "& vmGUID & ".vsv file
in importDirectory success."
       end if
    end if

End Function

'-------------------------------------------------------------------
--
' Change status of a virtual machine
'-------------------------------------------------------------------
--
Function RequestStateChange(computerSystem, action)
    dim objInParam, objOutParams

    objStdOut.WriteLine Format2("RequestStateChange({0}, {1})",
computerSystem.ElementName, action)

    RequestStateChange = false
    set objInParam =
computerSystem.Methods_("RequestStateChange").InParameters.Spawn
Instance_()

    if action = "Suspended" then
        objInParam.RequestedState = Suspended
    elseif action = "start" then
        objInParam.RequestedState = Enabled
    elseif action = "stop" then
        objInParam.RequestedState = Disabled
    else
       objStdOut.WriteLine Format1("change state to {0} is not
support!", action)
       WScript.Quit(1)
    end if

    set objOutParams =
computerSystem.ExecMethod_("RequestStateChange", objInParam)

    if (WMIMethodStarted(objOutParams)) then
        if (WMIJobCompleted(objOutParams)) then
            objStdOut.WriteLine Format2("VM {0} was {1} successfully",
computerSystem.ElementName, action)
            RequestStateChange = true
        end if
    end if

End Function

'-------------------------------------------------------------------
--
' Define a virtual system
'-------------------------------------------------------------------
--
Function ImportVirtualSystem(importDirectory, generateNewID)

    dim objInParam, objOutParams
    ImportVirtualSystem = false

    if Not fileSystem.FolderExists(importDirectory) then
        objStdOut.WriteLine Format1("importDirectory({0}) doesn't
exists.", importDirectory)
    end if
```

```
    set objInParam =
managementService.Methods_("ImportVirtualSystem").InParameters.S
pawnInstance_()
    objInParam.GenerateNewID = generateNewID
    objInParam.ImportDirectory = importDirectory

    set objOutParams =
managementService.ExecMethod_("ImportVirtualSystem", objInParam)

    if objOutParams.ReturnValue = wmiStarted then
        if (WMIJobCompleted(objOutParams)) then
            ImportVirtualSystem = true
        end if
    elseif (objOutParams.ReturnValue = wmiSuccessful) then
        ImportVirtualSystem = true
    else
        objStdOut.WriteLine Format1("DefineVirtualSystem failed with
ReturnValue {0}", wmiStatus)
    end if

End Function

'-------------------------------------------------------------------
--
' Destroy a virtual system
'-------------------------------------------------------------------
--
Function DestroyVirtualSystem(computerSystem)

    dim objInParam, objOutParams

    DestroyVirtualSystem = false
    set objInParam =
managementService.Methods_("DestroyVirtualSystem").InParameters.
SpawnInstance_()
    objInParam.ComputerSystem = computerSystem.Path_.Path

    set objOutParams =
managementService.ExecMethod_("DestroyVirtualSystem", objInParam)

    if (objOutParams.ReturnValue = wmiStarted) then
        if (WMIJobCompleted(objOutParams)) then
            DestroyVirtualSystem = true
        end if
    elseif (objOutParams.ReturnValue = wmiSuccessful) then
        DestroyVirtualSystem = true
    else
        objStdOut.WriteLine Format1("DestroyVirtualSystem failed
with ReturnValue {0}", objOutParams.ReturnValue)
    end if

End Function

'-------------------------------------------------------------------
--
' Export a virtual system
'-------------------------------------------------------------------
--
Function ExportVirtualSystem(computerSystem, exportDirectory)
    dim vmPath, vmTmpPath, tmpExportDirectory
    dim objInParam, objOutParams

    ExportVirtualSystem = false
    tmpExportDirectory = exportDirectory & "¥" & "tmp"
    vmPath = exportDirectory & "¥" & computerSystem.ElementName
    vmTmpPath = tmpExportDirectory & "¥" & computerSystem.ElementName
```

```
    if Not fileSystem.FolderExists(exportDirectory) then
        fileSystem.CreateFolder(exportDirectory)
        objStdOut.WriteLine "Warn, previous backuped vm's config file
doesn't exist!"
    else
        if fileSystem.FolderExists(vmTmpPath) then
            fileSystem.DeleteFolder vmTmpPath, true
        end if

    end if

    set objInParam =
managementService.Methods_("ExportVirtualSystem").InParameters.S
pawnInstance_()
    objInParam.ComputerSystem = computerSystem.Path_.Path
    objInParam.CopyVmState = false
    objInParam.ExportDirectory = tmpExportDirectory

    set objOutParams =
managementService.ExecMethod_("ExportVirtualSystem", objInParam)

    if objOutParams.ReturnValue = wmiStarted then
        if (WMIJobCompleted(objOutParams)) then
            ExportVirtualSystem = true
            objStdOut.WriteLine "vmPath: " & vmPath
            if fileSystem.FolderExists(vmPath) then
               fileSystem.DeleteFolder vmPath, true
            end if
            fileSystem.MoveFolder vmTmpPath, vmPath
        end if
    elseif (objOutParams.ReturnValue = wmiSuccessful) then
        ExportVirtualSystem = true

        if fileSystem.FolderExists(vmPath) then
            fileSystem.DeleteFolder vmPath, true
        end if
        fileSystem.MoveFolder vmTmpPath, vmPath
    else
        objStdOut.WriteLine Format1("DefineVirtualSystem failed with
ReturnValue {0}", wmiStatus)
    end if

End Function

'-----------------------------------------------------------------
--
' Backup a virtual system's config file
'-----------------------------------------------------------------
--
Function BackupVMConfigFile(computerSystem, exportDirectory,
vmrootDirectory)
    Dim tmpExportDirectory
    Dim FromPath, DestPath, BinFilePath

    BackupVMConfigFile = false

    'export VM's config file
    if ExportVirtualSystem(computerSystem, exportDirectory) then
        objStdOut.WriteLine "Export done."
    else
        objStdOut.WriteLine "ExportVirtualSystem Failed."
        Exit Function
    end if

    'move status file to temp folder
```

```
    FromPath = vmrootDirectory & "¥" & "Virtual Machines" & "¥" &
computerSystem.Name & "¥" & "*"
    DestPath = exportDirectory & "¥" & computerSystem.ElementName &
"¥" & "Virtual Machines" & "¥" & computerSystem.Name
    BinFilePath = vmrootDirectory & "¥" & "Virtual Machines" & "¥"
& computerSystem.Name & "¥" & computerSystem.Name & ".bin"

    if fileSystem.FileExists(BinFilePath) then
        fileSystem.CopyFile FromPath, DestPath, true
        objStdOut.WriteLine "move status file done"
    else
        objStdOut.WriteLine "vm's status file doesn't exists."
    end if

    BackupVMConfigFile = true

End Function

'-------------------------------------------------------------------
--
' Handle wmi return values
'-------------------------------------------------------------------
--
Function WMIMethodStarted(outParam)

    dim wmiStatus

    WMIMethodStarted = false

    if Not IsNull(outParam) then
        wmiStatus = outParam.ReturnValue

        if  wmiStatus = wmiStarted then
            WMIMethodStarted = true
        end if

    end if

End Function

'-------------------------------------------------------------------
--
' Handle wmi Job object
'-------------------------------------------------------------------
--
Function WMIJobCompleted(outParam)

    dim WMIJob, jobState

    set WMIJob = objWMIService.Get(outParam.Job)

    WMIJobCompleted = true

    jobState = WMIJob.JobState

    while jobState = JobRunning or jobState = JobStarting
        objStdOut.WriteLine Format1("In progress... {0}%%
completed.",WMIJob.PercentComplete)
        WScript.Sleep(1000)
        set WMIJob = objWMIService.Get(outParam.Job)
        jobState = WMIJob.JobState
    wend

    if (jobState <> JobCompleted) then
        objStdOut.WriteLine Format1("ErrorCode:{0}",
WMIJob.ErrorCode)
        objStdOut.WriteLine Format1("ErrorDescription:{0}",
```

```
WMIJob.ErrorDescription)
       WMIJobCompleted = false
   end if

End Function

'-------------------------------------------------------------------
--------------
' The string formating functions to avoid string concatenation.
'-------------------------------------------------------------------
--------------
Function Format2(myString, arg0, arg1)
   Format2 = Format1(myString, arg0)
   Format2 = Replace(Format2, "{1}", arg1)
End Function

'-------------------------------------------------------------------
--------------
' The string formating functions to avoid string concatenation.
'-------------------------------------------------------------------
--------------
Function Format1(myString, arg0)
   Format1 = Replace(myString, "{0}", arg0)
End Function

'-------------------------------------------------------------------
--------------
'  clear vm's ScopeOfResidence value
'  vm import failer be due to SVCMM installed.
'-------------------------------------------------------------------
--------------
Function ClearVMScopes(computerSystem)

   Dim VMSystemGlobalSettingData
   Dim Result

   Set VMSystemGlobalSettingData =
(computerSystem.Associators_("MSVM_ElementSettingData",
"Msvm_VirtualSystemGlobalSettingData")).ItemIndex(0)
   VMSystemGlobalSettingData.ScopeOfResidence = ""
   Result =
managementService.ModifyVirtualSystem(computerSystem.Path_.Path,
VMSystemGlobalSettingData.GetText_(1))

   objStdOut.WriteLine "ClearVMScopes Result:" & Result
   if Result <> 0 then
      objStdOut.WriteLine "ClearVMScopes Error : " & Result
   end if

End Function

'-------------------------------------------------------------------
--------------
'  clear previous vm that have the same GUID.
'-------------------------------------------------------------------
--------------
Sub ClearVmGuid(vmDefaultRootDirectory, vmGUID)
   Dim vmDefaultRootPath
   vmDefaultRootPath = vmDefaultRootDirectory & "¥" & "Virtual
Machines" & "¥" & vmGUID & ".xml"

   if (fileSystem.FileExists(vmDefaultRootPath)) then
      fileSystem.DeleteFile vmDefaultRootPath, true
      objStdOut.WriteLine "previous vm that have the same GUID is
deleted."
   end if
```

```
End Sub
```

vmstop.vbs

A script to save a virtual machine.

```
'=====================================================================
=
' vmstop.vbs
'=====================================================================
=
' Title   : Save VM's state, export VM's config file then destroy
the VM
' Authors : Wang Haoran
' Date    : 2009/02/16
' Revised : 2009/02/16
'=====================================================================
=

option explicit

dim objWMIService
dim managementService
dim fileSystem
dim objStdOut
const JobStarting = 3
const JobRunning = 4
const JobCompleted = 7
const wmiStarted = 4096
const wmiSuccessful = 0
const Enabled = 2
const Disabled = 3
const Suspended = 32769
' Instance for standard output
Set objStdOut = Wscript.StdOut

Main()


'---------------------------------------------------------------------
---
' Main
'---------------------------------------------------------------------
---
Sub Main()

    dim computer, objArgs, vmName, vm, backupDirectory,
vmrootDirectory
    dim DestPath, FromPath

    set fileSystem =
Wscript.CreateObject("Scripting.FileSystemObject")
    computer = "."
    set objWMIService = GetObject("winmgmts:¥¥" & computer &
"¥root¥virtualization")
    set managementService = objWMIService.ExecQuery("select * from
Msvm_VirtualSystemManagementService").ItemIndex(0)

    set objArgs = WScript.Arguments
    if WScript.Arguments.Count = 3 then
        vmName = objArgs.Unnamed.Item(0)
        backupDirectory = objArgs.Unnamed.Item(1)
        vmrootDirectory = objArgs.Unnamed.Item(2)
    else
        'WScript.Echo "usage: cscript ExportVirtualSystem.vbs
```

```
vmName backupDirectoryName vmrootDirectoryName"
   objStdOut.WriteLine "usage: cscript vmstop.vbs vmName
backupDirectoryName vmrootDirectoryName"
       WScript.Quit(1)
    end if

    set vm = GetComputerSystem(vmName)
    objStdOut.WriteLine "GetComputerSystem complete"

    if IsVMStarted(vm) then
        'save VM's state
        if RequestStateChange(vm, "Suspended") then
            objStdOut.WriteLine " Save done."
        else
            objStdOut.WriteLine  "RequestStateChange failed"
            WScript.Quit(1)
        end if
    else
        objStdOut.WriteLine "warn! vm isn't started, Don't backup
vm then destroy it."
    end if

    'Clear VM's Scopes.
    ClearVMScopes(vm)

    'Backup vm's config file
    if BackupVMConfigFile(vm, backupDirectory, vmrootDirectory)
then
        objStdOut.WriteLine "BackupVMConfigFile done."
    else
        objStdOut.WriteLine  "BackupVMConfigFile failed"
        WScript.Quit(1)
    end if

    'destroy the VM
    if DestroyVirtualSystem(vm) then
        objStdOut.WriteLine "Destroy done."
        'WScript.Quit(0)
    else
        objStdOut.WriteLine "DestroyVirtualSystem Failed."
        WScript.Quit(1)

    end if

    'copy exported vm's config file to up folder
    FromPath = backupDirectory & "¥" & vm.ElementName
    DestPath = vmrootDirectory
    fileSystem.CopyFolder FromPath, DestPath, true
    'fileSystem.DeleteFolder backupDirectory, true
    objStdOut.WriteLine "copy exported vm's config file done"

End Sub

'-----------------------------------------------------------------
---
' Retrieve Msvm_VirtualComputerSystem from base on its ElementName
'-----------------------------------------------------------------
---
Function GetComputerSystem(vmElementName)
    On Error Resume Next
    dim query
    query = Format1("select * from Msvm_ComputerSystem where
ElementName = '{0}'", vmElementName)
    set GetComputerSystem =
objWMIService.ExecQuery(query).ItemIndex(0)
    if (Err.Number <> 0) then
        objStdOut.WriteLine Format1("Err.Number: {0}", Err.Number)
```

```
       objStdOut.WriteLine
Format1("Err.Description:{0}",Err.Description)
       WScript.Quit(1)
   end if
End Function

'-------------------------------------------------------------------
---
' Change status of a virtual machine
'-------------------------------------------------------------------
---
Function RequestStateChange(computerSystem, action)
   dim objInParam, objOutParams

   objStdOut.WriteLine Format2("RequestStateChange({0}, {1})",
computerSystem.ElementName, action)

   RequestStateChange = false
   set objInParam =
computerSystem.Methods_("RequestStateChange").InParameters.Spaw
nInstance_()

   if action = "Suspended" then
      objInParam.RequestedState = Suspended
   elseif action = "start" then
      objInParam.RequestedState = Enabled
   elseif action = "stop" then
      objInParam.RequestedState = Disabled
   else
     objStdOut.WriteLine Format1("change state to {0} is not
support!", action)
     WScript.Quit(1)
   end if

   set objOutParams =
computerSystem.ExecMethod_("RequestStateChange", objInParam)

   if (WMIMethodStarted(objOutParams)) then
      if (WMIJobCompleted(objOutParams)) then
         objStdOut.WriteLine Format2("VM {0} was {1}
successfully", computerSystem.ElementName, action)
         RequestStateChange = true
      end if
   end if

End Function

'-------------------------------------------------------------------
---
' Export a virtual system
'-------------------------------------------------------------------
---
Function ExportVirtualSystem(computerSystem, exportDirectory)
   dim vmPath, vmTmpPath, tmpExportDirectory
   dim objInParam, objOutParams

   ExportVirtualSystem = false
   tmpExportDirectory = exportDirectory & "¥" & "tmp"
   vmPath = exportDirectory & "¥" & computerSystem.ElementName
   vmTmpPath = tmpExportDirectory & "¥" &
computerSystem.ElementName

   if Not fileSystem.FolderExists(exportDirectory) then
      fileSystem.CreateFolder(exportDirectory)
      objStdOut.WriteLine "Warn, previous backuped vm's config file
doesn't exist!"
   else
```

```
        if fileSystem.FolderExists(vmTmpPath) then
            fileSystem.DeleteFolder vmTmpPath, true
        end if

    end if

    set objInParam =
managementService.Methods_("ExportVirtualSystem").InParameters.
SpawnInstance_()
    objInParam.ComputerSystem = computerSystem.Path_.Path
    objInParam.CopyVmState = false
    objInParam.ExportDirectory = tmpExportDirectory

    set objOutParams =
managementService.ExecMethod_("ExportVirtualSystem", objInParam)

    if objOutParams.ReturnValue = wmiStarted then
        if (WMIJobCompleted(objOutParams)) then
            ExportVirtualSystem = true

            if fileSystem.FolderExists(vmPath) then
                fileSystem.DeleteFolder vmPath, true
            end if
            fileSystem.MoveFolder vmTmpPath, vmPath
        end if
    elseif (objOutParams.ReturnValue = wmiSuccessful) then
        ExportVirtualSystem = true

        if fileSystem.FolderExists(vmPath) then
            fileSystem.DeleteFolder vmPath, true
        end if
        fileSystem.MoveFolder vmTmpPath, vmPath
    else
        objStdOut.WriteLine Format1("DefineVirtualSystem failed
with ReturnValue {0}", wmiStatus)
    end if

End Function

'-------------------------------------------------------------------
---
' Backup a virtual system's config file
'-------------------------------------------------------------------
---
Function BackupVMConfigFile(computerSystem, exportDirectory,
vmrootDirectory)
    Dim tmpExportDirectory
    Dim FromPath, DestPath, BinFilePath

    BackupVMConfigFile = false

    'export VM's config file
    if ExportVirtualSystem(computerSystem, exportDirectory) then
        objStdOut.WriteLine "Export done."
    else
        objStdOut.WriteLine "ExportVirtualSystem Failed."
        Exit Function
    end if

    'move status file to temp folder

    FromPath = vmrootDirectory & "¥" & "Virtual Machines" & "¥" &
computerSystem.Name & "¥" & "*"
    DestPath = exportDirectory & "¥" & computerSystem.ElementName
& "¥" & "Virtual Machines" & "¥" & computerSystem.Name
    BinFilePath = vmrootDirectory & "¥" & "Virtual Machines" & "¥"
& computerSystem.Name & "¥" & computerSystem.Name & ".bin"
```

```
    objStdOut.WriteLine "FromPath: " & FromPath
    objStdOut.WriteLine "DestPath : " & DestPath
    objStdOut.WriteLine "BinFilePath: " & BinFilePath
    if fileSystem.FileExists(BinFilePath) then
        fileSystem.CopyFile FromPath, DestPath, true
        objStdOut.WriteLine "move status file done"
    else
        objStdOut.WriteLine "vm's status file doesn't exists."
    end if

    BackupVMConfigFile = true

End Function

'------------------------------------------------------------------
---
' Destroy a virtual system
'------------------------------------------------------------------
---
Function DestroyVirtualSystem(computerSystem)

    dim objInParam, objOutParams

    DestroyVirtualSystem = false
    set objInParam =
managementService.Methods_("DestroyVirtualSystem").InParameters
.SpawnInstance_()
    objInParam.ComputerSystem = computerSystem.Path_.Path

    set objOutParams =
managementService.ExecMethod_("DestroyVirtualSystem",
objInParam)

    if (objOutParams.ReturnValue = wmiStarted) then
        if (WMIJobCompleted(objOutParams)) then
            DestroyVirtualSystem = true
        end if
    elseif (objOutParams.ReturnValue = wmiSuccessful) then
        DestroyVirtualSystem = true
    else
        objStdOut.WriteLine Format1("DestroyVirtualSystem failed
with ReturnValue {0}", objOutParams.ReturnValue)
    end if

End Function

'------------------------------------------------------------------
---
' Handle wmi return values
'------------------------------------------------------------------
---
Function WMIMethodStarted(outParam)

    dim wmiStatus

    WMIMethodStarted = false

    if Not IsNull(outParam) then
        wmiStatus = outParam.ReturnValue

        if  wmiStatus = wmiStarted then
            WMIMethodStarted = true
        end if

    end if

End Function
```

```
'---------------------------------------------------------------
---
' Handle wmi Job object
'---------------------------------------------------------------
---
Function WMIJobCompleted(outParam)

    dim WMIJob, jobState

    set WMIJob = objWMIService.Get(outParam.Job)

    WMIJobCompleted = true

    jobState = WMIJob.JobState

    while jobState = JobRunning or jobState = JobStarting
        objStdOut.WriteLine Format1("In progress... {0}%%
completed.",WMIJob.PercentComplete)
        WScript.Sleep(1000)
        set WMIJob = objWMIService.Get(outParam.Job)
        jobState = WMIJob.JobState
    wend

    if (jobState <> JobCompleted) then
        objStdOut.WriteLine Format1("ErrorCode:{0}",
WMIJob.ErrorCode)
        objStdOut.WriteLine Format1("ErrorDescription:{0}",
WMIJob.ErrorDescription)
        WMIJobCompleted = false
    end if

End Function

'---------------------------------------------------------------
---------------
' The string formating functions to avoid string concatenation.
'---------------------------------------------------------------
---------------
Function Format2(myString, arg0, arg1)
    Format2 = Format1(myString, arg0)
    Format2 = Replace(Format2, "{1}", arg1)
End Function

'---------------------------------------------------------------
---------------
' The string formating functions to avoid string concatenation.
'---------------------------------------------------------------
---------------
Function Format1(myString, arg0)
    Format1 = Replace(myString, "{0}", arg0)
End Function

'---------------------------------------------------------------
---------------
' Check VM if is started
'---------------------------------------------------------------
---------------
Function IsVMStarted(computerSystem)

    if computerSystem.EnabledState <> Enabled then
        IsVMStarted = false
    else
        IsVMStarted = true
    end if

End Function
```

```
'-------------------------------------------------------------
---------------
'  clear vm's ScopeOfResidence value
'  vm import failer be due to SVCMM installed.
'-------------------------------------------------------------
---------------
Function ClearVMScopes(computerSystem)

   Dim VMSystemGlobalSettingData
   Dim Result

   Set VMSystemGlobalSettingData =
(computerSystem.Associators_("MSVM_ElementSettingData",
"Msvm_VirtualSystemGlobalSettingData")).ItemIndex(0)
   VMSystemGlobalSettingData.ScopeOfResidence = ""
   Result =
managementService.ModifyVirtualSystem(computerSystem.Path_.Path
, VMSystemGlobalSettingData.GetText_(1))

   objStdOut.WriteLine "ClearVMScopes Result:" & Result
   if Result <> 0 then
      objStdOut.WriteLine "ClearVMScopes Error : " & Result
   end if

End Function
```

W#HyperV.bat

A parameter file for vmstart.vbs and vmstop.vbs. Edit the bold text for use as necessary.

```
rem ************************************
rem *          W#HyperV.BAT            *
rem *                                  *
rem * Title   : Hyper-V Parameters     *
rem * Authors : Wang Haoran            *
rem * Date    : 2009/02/16             *
rem * Revised : 2009/02/16             *
rem ************************************

rem -----------------------------------------------
rem Environment variable list set in W#HyperV.BAT
rem W#HyperV1 :  Virtual machine name
rem W#HyperV2 :  Directory name where virtual machine configuration file is saved
rem W#HyperV3 :  Backup directory name of virtual machine
rem W#HyperV4 :  GUID name of virtual machine
rem W#HyperV5 : Default directory name where virtual machine configuration file is
saved
rem W#HyperV6 :  Startup retry count for virtual machine
rem W#HyperV7 :  Disk disconnection error avoidance waiting time [sec]
rem -----------------------------------------------

SET W#HyperV1=  Ex. w2k8-x86jp-vm
SET W#HyperV2= Ex. "Z:¥Hyper-V¥w2k8-x86jp-vm"
SET W#HyperV3= Ex. "Z:¥Hyper-V¥w2k8-x86jp-vm¥backup"
SET W#HyperV4= Ex. 8DECCAFE-385E-4FDF-99A2-534D76B132C2
SET W#HyperV5= Ex. "C:¥ProgramData¥Microsoft¥Windows¥Hyper-V"
SET W#HyperV6= Ex. 30
SET W#HyperV6= Ex. 30
```

checkstatus.vbs

A script to check virtual machine startup status.

```
'=============================================================
=
' Title   : Check virtual machine status
' Authors : Takaaki Fukunaga
' Date    : 2009/02/16
' Revised : 2009/02/16
'=============================================================
=
' VM State
'     0: Unknown
'     2: Enabled (Running)
'     3: Disabled
'    10: Rebooted
'    11: Reset
' 32768: Paused
' 32769: Suspended (Save completed)

Option Explicit

'-------------------------------------------------------------
-
' Variables
'-------------------------------------------------------------
-
Public objWMI
Public VM
Public VMList
Public VMName
Public VMEnabledState

Main()


'-------------------------------------------------------------
-
' Main()
'-------------------------------------------------------------
-
Sub Main()
   dim objArgs

   set objArgs = WScript.Arguments

   VMName = objArgs.Unnamed.Item(0)

   checkVMState()
   For Each VM In VMList
     VMEnabledState = VMList.ItemIndex(0).EnabledState

     WScript.Stdout.WriteLine VMEnabledState

     If VMEnabledState=0 then
       WScript.Stdout.WriteLine "Unknown"
       WScript.Quit (1)
     ElseIf VMEnabledState=3 then
       WScript.Stdout.WriteLine "Disabled"
       WScript.Quit (2)
     Else
       WScript.Quit (0)
     End If
   Next
End Sub


'-------------------------------------------------------------
-
```

```
' checkVMState()
'-------------------------------------------------------------
-
Sub checkVMState()
   set objWMI = GetObject("winmgmts:¥¥.¥root¥virtualization")
   set VMList = objWMI.ExecQuery("SELECT * FROM Msvm_ComputerSystem
Where ElementName='" & VMName & "'")
   if VMList.count = 0 then
     WScript.Stdout.WriteLine "There is no such VM '" & VMName &
"' in this system."
     WScript.Quit (3)
   end if
End Sub
```

start.bat

A script to start up vmstart.vbs.

```
rem ************************************
rem *            start.bat             *
rem *                                  *
rem * Title   : Start script file      *
rem * Authors : Wang Haoran            *
rem * Date    : 2009/02/16             *
rem * Revised : 2009/02/16             *
rem ************************************

rem ----------------------------------------
rem Environment variable settings used for batch
rem ----------------------------------------

CALL W#HyperV.BAT

rem ------------------------------------------------
rem Environment variable list set in W#HyperV.BAT
rem W#HyperV1 : Virtual machine name
rem W#HyperV2 : Directory name where virtual machine configuration file is saved
rem W#HyperV3 : Backup directory name of virtual machine
rem W#HyperV4 : GUID name of virtual machine
rem W#HyperV5 : Default directory name where virtual machine configuration file is
saved
rem W#HyperV6 : Startup retry count for virtual machine
rem W#HyperV7 : Disk disconnection error avoidance waiting time [sec]
rem ------------------------------------------------

rem ************************************
rem Startup factor check
rem ************************************
IF "%CLP_EVENT%" == "START" GOTO NORMAL
IF "%CLP_EVENT%" == "FAILOVER" GOTO FAILOVER
IF "%CLP_EVENT%" == "RECOVER" GOTO RECOVER

rem ExpressCluster Server not running
GOTO no_arm




rem ************************************
rem Normal startup response process
rem ************************************
:NORMAL
```

```
rem Disk check
IF "%CLP_DISK%" == "FAILURE" GOTO ERROR_DISK

rem Move to script path
cd %CLP_SCRIPT_PATH%

rem *************
rem Normal business process
rem *************
cscript
vmstart.vbs %W#HyperV1% %W#HyperV2% %W#HyperV3% %W#HyperV4% %W#
HyperV5% %W#HyperV6%

rem Priority check
IF "%CLP_SERVER%" == "OTHER" GOTO ON_OTHER1

rem *************
rem Process at top priority
rem (Ex.) ARMBCAST /MSG "Starting on top priority server" /A
rem *************
GOTO EXIT

:ON_OTHER1
rem *************
rem Process at other than top priority
rem (Ex.) ARMBCAST /MSG "Starting on a server other than top priority server" /A
rem *************
GOTO EXIT




rem ************************************
rem Recovery response process
rem ************************************
:RECOVER

rem *************
rem Recovery process after cluster restored
rem *************

GOTO EXIT




rem ************************************
rem Failover response process
rem ************************************
:FAILOVER

rem Disk check
IF "%CLP_DISK%" == "FAILURE" GOTO ERROR_DISK

rem Move to script path
cd %CLP_SCRIPT_PATH%

rem *************
rem Business application startup and recovery process after failover
rem *************
cscript
vmstart.vbs %W#HyperV1% %W#HyperV2% %W#HyperV3% %W#HyperV4% %W#
```

```
HyperV5% %W#HyperV6%

rem Priority check
IF "%CLP_SERVER%" == "OTHER" GOTO ON_OTHER2

rem *************
rem Process at top priority
rem (Ex.) ARMBCAST /MSG "Starting on top priority server (after failover)" /A
rem *************
GOTO EXIT

:ON_OTHER2
rem *************
rem Process at other than top priority
rem (Ex.) ARMBCAST /MSG "Starting on a server other than top priority server (after
failover)" /A
rem *************
GOTO EXIT



rem ************************************
rem Exception process
rem ************************************

rem Disk related error process
:ERROR_DISK
ARMBCAST /MSG "Connecting switching partition failed" /A
GOTO EXIT


rem ARM not running
:no_arm
ARMBCAST /MSG "ExpressCluster Server is not running" /A



:EXIT
```

stop.bat

A script to start up `vmstop.vbs`.

```
rem ************************************
rem *            stop.bat              *
rem *                                  *
rem * Title   : Stop script file       *
rem * Authors : Wang Haoran            *
rem * Date    : 2009/02/16             *
rem * Revised : 2009/02/16             *
rem ************************************

rem ----------------------------------------
rem Environment variable settings used for batch
rem ----------------------------------------

CALL W#HyperV.BAT

rem ------------------------------------------------
rem Environment variable list set in W#HyperV.BAT
rem W#HyperV1 : Virtual machine name
rem W#HyperV2 : Directory name where virtual machine configuration file is saved
rem W#HyperV3 : Backup directory name of virtual machine
rem W#HyperV4 : GUID name of virtual machine
```

```
rem W#HyperV5 : Default directory name where virtual machine configuration file is
saved
rem W#HyperV6 : Startup retry count for virtual machine
rem W#HyperV7 : Disk disconnection error avoidance waiting time [sec]
rem ------------------------------------------------


rem ************************************
rem Startup factor check
rem ************************************
IF "%CLP_EVENT%" == "START" GOTO NORMAL
IF "%CLP_EVENT%" == "FAILOVER" GOTO FAILOVER

rem ExpressCluster Server not running
GOTO no_arm




rem ************************************
rem Normal end response process
rem ************************************
:NORMAL

rem Disk check
IF "%CLP_DISK%" == "FAILURE" GOTO ERROR_DISK

rem Move to script path
cd %CLP_SCRIPT_PATH%

rem *************
rem Normal business process
rem *************
cscript vmstop.vbs %W#HyperV1% %W#HyperV3% %W#HyperV2%

rem To avoid error when disconnecting disk
armsleep %W#HyperV7%

rem Priority check
IF "%CLP_SERVER%" == "OTHER" GOTO ON_OTHER1

rem *************
rem Process at top priority
rem (Ex.) ARMBCAST /MSG "Shutting down on top priority server" /A
rem *************
GOTO EXIT


:ON_OTHER1
rem *************
rem Process at other than top priority
rem (Ex.) ARMBCAST /MSG "Shutting down on a server other than top priority
server" /A
rem *************
GOTO EXIT




rem ************************************
rem Failover response process
rem ************************************
```

```
:FAILOVER

rem Disk check
IF "%CLP_DISK%" == "FAILURE" GOTO ERROR_DISK

rem Move to script path
cd %CLP_SCRIPT_PATH%

rem *************
rem Business application startup and recovery process after failover
rem *************
cscript vmstop.vbs %W#HyperV1% %W#HyperV3% %W#HyperV2%

rem To avoid error when disconnecting disk
armsleep %W#HyperV7%

rem Priority check
IF "%CLP_SERVER%" == "OTHER" GOTO ON_OTHER2

rem *************
rem Process at top priority
rem (Ex.) ARMBCAST /MSG "Shutting down on top priority server (after failover)" /A
rem *************
GOTO EXIT

:ON_OTHER2
rem *************
rem Process at other than top priority
rem (Ex.) ARMBCAST /MSG "Shutting down on a server other than top priority server
(after failover)" /A
rem *************
GOTO EXIT




rem ************************************
rem Exception process
rem ************************************

rem Disk related error process
:ERROR_DISK
ARMBCAST /MSG "Connecting switching partition failed" /A
GOTO EXIT


rem ARM not running
:no_arm
ARMBCAST /MSG "ExpressCluster Server is not running" /A


:EXIT
```

genw.bat

A script to start up `checkstatus.vbs`. Edit the bold text for use as necessary.

```
rem ************************************
rem *             genw.bat             *
rem * Title   : Start up checkstatus.vbs  *
rem * Authors : Takaaki Fukunaga         *
rem * Date    : 2009/02/16              *
rem * Revised : 2009/02/16               *
rem ************************************
```

```
set vmname= Ex. w2k8-x86jp-vm
set vbs_path= Ex. C:¥Program Files¥ExpressCluster¥bin¥checkstatus.vbs

echo START

cscript "%vbs_path%" %vmname%
if errorlevel 1 goto ERROR

:NORMAL
exit 0

:ERROR
exit 1
```