LISA

Linux Integrated Services Automation

Users guide

Table of Contents

Chapter 1 - Overview

The LIS Automation, or LISA, is a collection of PowerShell scripts, guest side test case scripts, and host side test case scripts. The intent of these scripts is to automate the testing of the Hyper-V Linux Integration Services (LIS) on Linux VMs. The very basics of LISA are the following:

- Start a Linux VM running.
- Push files to a Linux VM.
- Start a test script running on a Linux VM.
- Collect files from the Linux VM.
- Stop the Linux VM.

The scripts are used in the context of a Test Environment. The test environment is composed of a Hyper-V server, the Virtual Machines that perform the actual tests, and an XML file that drives the test.

Terminology

The Test Environment is composed of a number of components. The following table describes each component of a typical Test Environments.

Component	Description
Hyper-V Server	The Hyper-v server hosts the Virtual Machines that perform the actual tests. Typically, the Hyper-V Server is also the Control Server.
Control Server	The server running the LISA PowerShell script which drives the automated testing. This will typically be the Hyper-V server.
Test VM	A Virtual Machine hosted on a Hyper-V server. The VM has a Linux distribution installed, and has been provisioned with any tools a test case may require. The test VM is the System Under Test (SUT).
Control Script	This is the PowerShell script that drives the automation process (lisa.ps1).
Test Script	These are a collection of Bash and PowerShell scripts. The Control server will push a guest side script to the test VM. Host side scripts are run on the control server.

Table 1-1

What's Included

There are four core PowerShell scripts that comprise the automation and are run on the Control Server. The Bash test case scripts are pushed to the Hyper-V Virtual Machines, where each Bash script performs a test. Host side test case scripts are writing in PowerShell and are run on the Control Server. The four PowerShell scripts that comprise the core automation include the following:

Power Shell script name	Description
lisa.ps1	The main PowerShell script

stateEngine.ps1	Contains the functions that drive the test state engine.
utilFunctions.ps1	Contains utility functions used by other PowerShell scripts.
OSAbstractions.ps1	Hides some of the OS specific behavior.

Basics of running a test

The automation engine is composed of the above PowerShell scripts. To run the scripts, invoke the lisa.ps1 PowerShell script and include the appropriate command line options, including an XML file. The lisa.ps1 script creates a log directory, a log file for the test run, and a log file for each test case that is performed. The level of detail written to the log file is controlled with the -dbgLevel command line option. The XML file identifies the virtual machines that will run the test scripts, the tests that can be run, and which tests each VM will run. Tests are collected into test suites. The XML file will define one or more test suites. Each VM in the XML file will specify which test suite will be run on the VM. Each VM could run a separate test suite. More commonly, all VMs will run the same test suite. Chapter 2 covers the XML file in detail. A typical command line might be:

```
.\lisa.ps1 run xml\myTests.xml -dbglevel 3
```

Chapter 3 covers the command line options in greater detail.

The following is a sample output of a test run with 2 test cases and the default dbglevel of 0.

```
PS D:\lisa> .\lisa.ps1 run .\xml\MyKvpTests.xml
LIS Automation script - version 2.0.0
Info : Sles11Sp3x64 currentTest updated to KVP_Basic
Info : Sles11Sp3x64 Status for test KVP_Basic = Success
Info : Sles11Sp3x64 currentTest updated to KVP_Add_Key_Values
Info : Sles11Sp3x64 Status for test KVP_Add_Key_Values  = Success
Info : Sles11Sp3x64 currentTest updated to done

Test Results Summary
LISA test run on 02/17/2014 10:36:58
XML file: .\xml\MyKvpTests.xml

VM: myTestVM
   Server :  localhost
   OS :  Microsoft Windows Server 2012 R2 Datacenter build 9600

   Test KVP_Basic             : Success
       Covers : KVP-01
   Test KVP_Add_Key_Values       : Success
       Covers KVP-02

Logs can be found at \\mytestserver\LisaTestResults\KvpTests-20140217-103658

PS D:\lisa>
```

Chapter 2 - Installation and Configuration

Installation involves checking out, or cloning, the project from a source repository server hosting the LISA automation scripts, creating the Virtual Machines, installing and configuring Linux on the Virtual Machines, creating and installing SSH keys, taking a snapshot, and creating the XML file.

Setup Overview

Setting up your environment to use LISA includes the following steps:

- Installing the Automation scripts.
- Creating the Test Virtual Machines.
- Installing and configuring Linux on each Test VM.
- Create SSH keys for the test Linux VMs.
- Take a snapshot of each Test VM, and rename it to ICABase.
- Create a XML file that defines the test suite, the test cases, and the VMs that will be used for the test run.

Installing the Automation Scripts

The files required for installing the ICA automation scripts can be found in the LIS/lis-test repository on github.com. To retrieve a copy, you can clone the repository:

These files need to be installed on the Control Server, which is typically the Hyper-V server hosting the VMs to be tested. To clone the files to your server, you will need install a Git client on your Control Server. Once is installed, the following commands should clone the repository:

```
mkdir D:\LIS
cd D:\LIS
git clone https://<yourUserName>:<yourPassword>@github.com/LIS/lis-test
```

Creating the Test VMs

Use the Hyper-V Manager to create Hyper-V virtual machines to run the tests. Each VM should include the following hardware as a minimum:

- 1 CPU.

- 1024MB of memory.
- Boot disk on the IDE controller.
- 1 Network adapter.

Note: If you are testing a Linux distribution that does not include the LIS drivers, your network adapter will need to be a Legacy network adapter.

Installing and configuring Linux

Once the VMs are created, install the Linux distribution to be used on each of the VMs. Make sure all the following software components have been installed and configured on each Linux VM:

- Software packages required to build a Linux kernel are installed.
- SSH, and the SSH daemon configured to start on boot.
- Port 22 open in the firewall.
- SSH daemon allows root login.
- atd package is installed, and the atd daemon is configured to start on boot.
- The dos2unix utility installed on the Linux VM.
- Configure the Linux VM with an SSH key.

Create SSH Keys

The LISA scripts use SSH to communicate with the VMs. The LISA scripts assume SSH keys have been configured. This is to avoid the use of passwords. After installing and configuring Linux, you will need to create the SSH keys. The following describes the typical steps:

1. Log onto the Linux system as root.
2. Run ssh-keygen and create a key.
   Note: This document assumes you created an RSA key and refers to the keys by their default names of id_rsa and id_rsa.pub.
3. Copy the public key id_rsa.pub to ~/.ssh/authorized_keys
   Note: if the file authorized_keys already exists, then append the public key to the authorized_keys file (cat id_rsa.pub >> ~/.ssh/authorized_keys).
4. Copy the private key (id_rsa) to the control server and convert the key into a Putty Private Key (PPK). To convert the SSH key to a Putty Private Key:
   a. Run PuttyGen
   b. In the PuttyGen window click the Load button, navigate to your private key, then save it with a .ppk extension (Putty Private Key).
   c. Copy your .ppk file to the ssh directory on the Control Server. The ssh directory is part of the directory tree you copied down from the SVN repository.
5. Test the keys to confirm everything is setup correctly.

Note: The LISA scripts use the Putty SSH client.

Take a Snap Shot of each VM

Once the VMs have been configured and working correctly, shutdown the VM and take a snapshot. Do this for each VM in the test environment. Rename the snapshot to the name: ICABase

Note: The default snapshot name used by the LISA scripts is ICABase.

One of the first steps the LISA automation script performs is to each VM listed in the .xml file is in a Hyper-V Off state and resets the VM to the ICABase snapshot. This ensures the VM is starting the test run from a known good state.

Create the XML file

The Test Environment is defined in the XML file. The XML file requires four sections:

- Global
- Test suites
- Test cases
- VMs

The global section defines general information required for the Test Environment. This is typically information that is not tied to any specific test or virtual machine. The test suites section defines one or more test suites. A test suite is just a collection of test case names. LISA will run the test cases in the order they are listed in the test suite. The test cases section defines all the test cases that can be run. A test suite is usually a subset of the test cases defined in the test cases section of the .xml file. The test cases section only defines the tests that are available to be run. The VMs section defines all the VMs that will be part of the Test Environment. Part of the VM definition includes the name of the test suite the VM will be running.

Global Section

The following is an example global section:

```
<global>
    <logfileRootDir>D:\lisa\TestResults</logfileRootDir>
    <defaultSnapshot>ICABase</defaultSnapshot>
    <email>
        <recipients>
            <to>myboss@mycompany.com</to>
            <to>myself@mycompany.com</to>
        </recipients>
        <sender>myself@mycompany.com</sender>
        <subject>LISA Test Run on server WS2012R2</subject>
        <smtpServer>mysmtphost.mycompany.com</smtpServer>
    </email>
</global>
```

The attributes found in the global section are defined in the following table.

XML Tag	Description
<logfileRootDir>	The base directory where log files are written.

<defaultSnapshot>	Name of the snapshot to reset a VM to.
<recipients>	A collection email address to send test results to.
<to>	Email address of a person to send the report to.
<sender>	The 'from' address when sending the email.
<subject>	The subject to add to the status e-mail message.
<smtpServer>	The smtp server to use when sending status e-mail messages.

Test Suites Section

This section defines one or more test suite. Each VM defined in the VMs section could run a separate test suite. The following table lists the XML tags that may be present in a test suite definition.

XML Tag	Description
<testSuite>	Start of the test suite section
<suite>	Start of a test suite definition
<suiteName>	Name of the test suite
<suiteTests>	Start of the test case list
<suiteTest>	Name of a test case that is part of the this test suite

A sample test suites section might look like the following:

```
<testSuites>
    <suite>
        <suiteName>KVP_Tests</suiteName>
        <suiteTests>
            <suiteTest>KVP_Basic</suiteTest>
            <suiteTest>KVP_Add_Key_Values</suiteTest>
        </suiteTests>
    </suite>
</testSuites>
```

Test Cases Section

The test cases section defines all the tests that are available for any virtual machine to run. A test case may specify test parameters. These are values passed to the test case. Test parameters allow a way to modify the behavior of a test case script. Some test case scripts may be used for multiple tests simply by defining a test case with different test parameters. For guest side test case scripts, the test parameters are placed in a file named constants.sh and then copied to the test VM. The test case script can then source ~/constants.sh. For host side test case scripts, the test parameters are collected into a semicolon separated string.

The following is an example of a tests section:

```
<testCases>
    <test>
        <testName>KVP_Basic</testName>
        <testScript>SetupScripts\KvpBasic.ps1</testScript>
        <timeout>600</timeout>
```

```
            <onError>Continue</onError>
            <noReboot>True</noReboot>
            <testparams>
                <param>rootDir=D:\lisa\trunk\lisablue</param>
                <param>TC_COVERED=KVP-01</param>
            </testparams>
        </test>

        <test>
            <testName>KVP_Add_Key_Values</testName>
            <testScript>VerifyKeyValue.sh</testScript>
            <files>remote-
scripts\ica\VerifyKeyValue.sh,tools/KVP/kvp_client</files>
            <PreTest>setupScripts\AddKeyValue.ps1</PreTest>
            <timeout>600</timeout>
            <onError>Continue</onError>
            <noReboot>True</noReboot>
            <testparams>
                <param>TC_COVERED=KVP-02</param>
                <param>Key=EEE</param>
                <param>Value=555</param>
                <param>Pool=0</param>
                <param>rootDir=D:\lisa\trunk\lisablue</param>
            </testparams>
        </test>
```

The above XML file fragment defines two test cases: KVP_Basic, and KVP_Add_Key_Values. The .xml file may contain as many test definitions as you need. The following table lists the XML tags that may be present in a test case definition.

XML Tag	Description
<test>	The start of a test definition.
<testName>	The name of the test.
<setupScript>	Optional. Name of a setup script to run before starting the VM. Setup scripts typically reconfigure a VM before a test.
<testScript>	The name of the test case script. For guest side test cases, this is usually a Bash script. For a host side test script, this will be a PowerShell script.
<timeout>	Optional. How long, in seconds, to wait for this test to complete.
<onError>	Optional. If set to 'Continue' additional test cases will run if this test case fails. Default value is 'Abort'
<noReboot>	Optional. If set to 'True' the VM will not be shut down after this test completes. It will transition to the next test case without shutting down. Default is False.
<files>	Optional for host side test script. A collection of files to push to the VM.
<pretest>	Optional. A PowerShell script to run on the host just before the test case script is started on the guest VM.
<testparams>	Optional. Contains test parameters that will be placed into the constants.sh script. The constants.sh file is pushed to the guest

	VM and the test case script can source it.
<uploadFiles>	Optional. Additional file to upload from the VM and put in the log directory.
<param>	Optional. A single Bash definition. The syntax must be value for a Bash script to source.

VMs Section

The VMs section defines the virtual machines in the Test Environment. These definition includes information identifying which Hyper-V server the virtual machine is on, the name of the virtual machine, the VMs IP address, and the tests the VM will run. The following is an example <VMs> section that defines two virtual machines.

```
<VMs>
    <vm>
        <hvServer>localhost</hvServer>
        <vmName>Sles11Sp3x64</vmName>
        <os>Linux</os>
        <ipv4>192.168.1.101</ipv4>
        <sshKey>id_rsa.ppk</sshKey>
        <suite>KVP_Tests</suite>
    </vm>
</VMs>
```

The XML file must contain at least one VM definition. Additional VMs may be defined. Each VM definition includes a <suite> attribute which identifies the test that will be run on the VM. The following table lists the XML tags that may be present in a VM definition.

XML tag	Description
<VMs>	The section of the XML file that contains VM definitions
<vm>	A Virtual Machine (VM) definition
<hvServer>	The IP address of the Hyper-V server hosting the VM
<vmName>	The Hyper-V name of the VM
<os>	The guest OS running on the VM
<ipv4>	The IPv4 address of the VM. Note: If the VM already has the LIS components installed, you can leave the value blank. The stateEngine.ps1 will use the LIS components to determine the IP address of the VM. e.g. <ipv4></ipv4>
<sshKey>	The SSH key to use when connecting to this VM
<suite>	Name of a test suite defined in the <testSuites> section.
<snapshotName >	Name of a snapshot if you do not want to use the default of ICABase. This overrides the global <snapshotName> setting.
<testparams>	Contains VM specific test parameters that will be placed into the constants.sh script.
<param>	A single Bash definition. This is a subkey under <testparams>

Chapter 3 - Usage

When you start a test run with LISA, you enter a command at a PowerShell command prompt. A LISA test run is driven by an XML file. The following discusses the command line syntax.

Note: Some of the Hyper-V PowerShell cmdlets will silently fail if they are not run with Administrator privilege. When starting PowerShell, right click on the PowerShell icon and select "Run as Administrator".

Command Line Syntax

To perform a test run, start the lisa.ps1 PowerShell script. The following is the command line syntax:

```
.\lisa.ps1 cmdVerb xmlFile [-debugLevel] [-email] [-help]
```

The command line arguments and options definitions are:

```
cmdVerb       : The command verb. What action to perform. Currently
                the only supported verb is:
                    run

xmFile        : Specifies the configuration for the virtual test
                environment. This argument is required.  The .xml
                file defines which virtual machines will run tests,
                and which tests each virtual machine will run.

-dbgLevel     : Optional.  Sets the level of debug messages to be
                logged. Values of 0 - 10 are supported. Default is 0.

-email        : Optional.  Send email when the test run is complete.
                Use the information in the <email> section of the
                .xml file to identify recipients and mail server
                to use.

-help         : Displays this help message.
```

A typical command would be similar to the following:

```
.\lisa.ps1 run xml\KVP_Tests.xml –dbgLevel 3 –email
```

The above command would run tests on the VMs defined in the KVP_Tests.xml file. The LISA log file, and logs from each test case, are stored in a sub directory under the directory defined by the <logfileRootDir> defined in the .xml file. The sub directory created for a test run is name after the xml file with a timestamp appended. The following is an example subdirectory name:

KVP_Tests-20140218-161957

Chapter 4 - Test Case Scripts

LISA run the test cases of a test suite on a test VM. Each test suite is a collection of test cases. Test cases are defined in the XML file. Each test case in the .xml file identifies a test script which implements the test the virtual machine will perform. There are two types of test scripts: host side script, and guest side script.

Guest side test case script

A guest side test script executes on the test VM, and is typically a Linux shell script. The requirements of the test script are simple. LISA will run the test case script as root. The script can do anything you want. However, your script must comply with a few simple requirements.

1. One of the first things the script must do is create the file state.txt in the root user's home directory and write "TestRunning" into the file. This can be done with the following shell command:

   ```
   echo "TestRunning" > ~/state.txt
   ```

2. When the test script successfully completes, it must update the status in state.txt before the script exits. This can be done with the following shell command:

   ```
   echo "TextCompleted" > ~/state.txt
   ```

3. If the test script encounters an error condition and will exit, it must update the status in state.txt before exiting. This can be done with the following shell command:

   ```
   echo "TestFailed" > ~/state.txt
   ```

The state.txt file is a simple text file and must contain a single line of text. The contents of state.txt must be one of the following text strings:

TestRunning
TestCompleted
TestAborted
TestFailed

The distinction between TestAborted and TestFailed is small. TestAborted is used to indicate the test script failed while setting up to perform the test. TestFailed is used to indicate the test script failed while performing the actual test.

The LISA PowerShell script on the control server monitors the ~/state.txt file to determine if the test is still running, has completed successfully, or aborted due to an error. All exit paths in the test script must update ~/state.txt before exiting.

Summary.log

Test scripts that run on the virtual machines have standard out and standard error redirected to a log file. This log file is collected when the test completes. In addition to the log file, the LISA automation scripts also look for a file named ~/summary.log. If the test case script creates this file, LISA will copy it from the test VM and include its contents in the final test results. The below text in purple is from the summary.log file created by the test case script.

VM: myTestVM
  Server : localhost
  OS : Microsoft Windows Server 2012 R2 Datacenter build 9600

  Test KVP_Basic : Success
    Covers : KVP-01

Logs can be found at \\mytestserver\LisaTestResults\myTests-20140217-103658

Constants.sh

Each test definition in the XML file may include one or more optional test parameters. When the control server pushes the test script to the virtual machine, it also pushes a file named constants.sh. The contents of constents.sh is created from the <testparams> section of the current test case. As an example, if your XML test case definition contained the following:

```
<testparams>
    <param>TC_COVERED=KVP-05</param>
    <param>Key=BBB</param>
    <param>Value=111</param>
</testparams>
```

The constants.sh file will contain the following:

```
TC_COVERED=KVP-05
Key=BBB
Value=111
```

Guest side test case scripts will source the constants.sh file. This implies the syntax used for test parameters must also be valid shell statements.

Guest side test script template

The following could be used as a template when creating a guest side test case script.

```
#!/bin/bash

ICA_TESTRUNNING="TestRunning"      # The test is running
ICA_TESTCOMPLETED="TestCompleted"  # The test completed successfully
ICA_TESTABORTED="TestAborted"      # Error during setup of test
ICA_TESTFAILED="TestFailed"        # Error during execution of test

CONSTANTS_FILE="constants.sh"      # Name of the constants.sh file to source
```

```
LogMsg()
{
    echo `date "+%a %b %d %T %Y"` : ${1}    # To add the timestamp output
}

UpdateTestState()
{
    echo "${1}" > ${HOME}/state.txt
}

#
# Create the state.txt file as soon as possible
#
cd ~
UpdateTestState $ICA_TESTRUNNING
LogMsg "Updated test case state to running"

#
# Delete any summary.log file left behind from a previous test
#
if [ -e ~/summary.log ]; then
    LogMsg "Cleaning up previous copies of summary.log"
    rm -f ~/summary.log
fi

#
# Source the constants file
#
if [ -e ~/${CONSTANTS_FILE} ]; then
    source ~/${CONSTANTS_FILE}
else
    msg="Error: no ${CONSTANTS_FILE} file"
    LogMsg "${msg}"
    echo "${msg}" >> ~/summary.log
    UpdateTestState $ICA_TESTABORTED
    exit 10
fi

#
# Make sure constants.sh contains the variables this script expects
#
if [ "${TC_COVERED:-UNDEFINED}" = "UNDEFINED" ]; then
    msg="The test parameter TC_COVERED is not defined in ${CONSTANTS_FILE}"
    LogMsg "${msg}"
    echo "${msg}" >> ~/summary.log
    UpdateTestState $ICA_TESTABORTED
    exit 20
fi

#
# Your test specific code starts here.
#
```

Host side test case script

A host side test script is a PowerShell script that is run on the Control Server. The Control server is the machine that is running the lisa.ps1 script and is typically the Hyper-V server hosting the VM under test. Host side test case scripts also have some requirements. The signature for the PowerShell script must be:

param ([String] $vmName, [String] $hvServer, [String] $testParams)

Where:

$vmName = is the name of the VM.

$hvServer = is the name of the Hyper-V server hosting the VM.

$testParams = a semicolon separate list of the test parameters from the test case definition in the .xml file. For example, if your XML test case definition contained the following:

```
<testparams>
    <param>TC_COVERED=KVP-05</param>
    <param>Key=BBB</param>
    <param>Value=111</param>
    <param>rootDir=D:\lisa\trunk\lisa</param>
</testparams>
```

Then the $testParams will contain the following:

```
"TC_COVERED=KVP-05;Key=BBB;Value=111;rootDir=D:\lisa\trunk\lisa"
```

LISA will run a host side test case script as a PowerShell job. When the PowerShell job starts, the current directory for the job will not be the LISA directory. It is standard practice to pass a test case parameter of rootDir that points to the correct directory for running the test case.

Host side test case script template

The following sample code could be used as a template when starting to write a host side test case script.

```
param([string] $vmName, [string] $hvServer, [string] $testParams)

#
# Your function definitions go here
#


#
# Main script body
#

#
# Make sure all command line arguments were provided
#
if (-not $vmName)
{
    "Error: vmName argument is null"
    return $False
```

```
}

if (-not $hvServer)
{
    "Error: hvServer argument is null"
    return $False
}

if (-not $testParams)
{
    "Error: testParams argument is null"
    return $False
}

#
# Initialize test related variables. Your list will vary...
#
$sshKey = $null
$ipv4 = $null
$rootDir = $null
$tcCovered = "unknown"

#
# Parse the testParams string
#
$params = $testParams.Split(";")
foreach($p in $params)
{
    $tokens = $p.Trim().Split("=")
    if ($tokens.Length -ne 2)
    {
        "Warn : Ignoring malformed test parameter '${p}'"
        continue   # malformed - just ignore the parameter
    }

    $val = $tokens[1].Trim()

    #
    # Modify the switch statement for your test
    #
    switch($tokens[0].Trim().ToLower())
    {
    "ipv4"        { $ipv4        = $val }
    "sshkey"      { $sshKey      = $val }
    "rootdir"     { $rootDir     = $val }
    "TC_COVERED"  { $tcCovered   = $val }
    default       { continue }
    }
}

#
# Make sure the required testParams were found
#
"Verify required test parameters were provided"
```

```
if (-not $rootDir)
{
    "Error: testParams is missing the rootDir parameter"
    return $False
}

#
# Lisa runs host side test scripts as a PowerShell job.
# The default directory for a PowerShell job is not the
# Lisa directory. cd to where the test code is.
#
if (-not (Test-Path $rootDir))
{
    "Error : The rootDir directory '${rootDir}' does not exist"
    return $False
}
cd $rootDir

#
# Your test specific code starts here
#



#
# The last thing a test script should do is return
# True or False to indicate if the test case passed.
#

return $True
```

Chapter 5 - Setup/Cleanup and Pre/Post Scripts

Setup and Cleanup scripts

In addition to test case scripts, LISA supports setup and cleanup scripts. Setup and cleanup scripts are scripts which are run while the test VM is in a stopped state. Setup scripts are run before the VM is booted. The purpose if a setup script is to modify the VM configuration for a specific test case. Cleanup scripts are run after a test case has completed and the VM is shutdown. A cleanup script can undo the work performed by a setup script.

Setup and cleanup scripts are PowerShell scripts run on the control server. They have the same signature as a host side test case, and are passed the test case test parameters.

```
param ([String] $vmName, [String] $hvServer, [String] $testParams)
```

Where:

$vmName = is the name of the VM.

$hvServer = is the name of the Hyper-V server hosting the VM.

$testParams = a semicolon separate list of the test parameters from the test case definition in the .xml file

A setup script is specified in a test case definition using the <setupScript> tag:

```
<setupScript>SetupScripts\AddNic.ps1</setupScript>
```

A Cleanup script is specified in a test case definition using the <cleanupScript> tag:

```
<cleanupScript>SetupScripts\RemoveNic.ps1</cleanupScript>
```

Pretest and Posttest scripts

Pretest and posttest scripts are similar to setup and cleanup scripts. They are PowerShell scripts which are run on the Control Server. The difference is that pretest and posttest scripts are run just before the test case script is started and right after the test case script completes. Pretest and posttest scripts use the same signature as setup/cleanup scripts:

```
param ([String] $vmName, [String] $hvServer, [String] $testParams)
```

Where:

$vmName = is the name of the VM.

$hvServer = is the name of the Hyper-V server hosting the VM.

$testParams = a semicolon separate list of the test parameters from the test case definition in the .xml file

A pretest script is specified in a test case definition using the <pretest> tag:

```
<PreTest>setupScripts\AddKeyValue.ps1</PreTest>
```

A posttest script is specified in a test case definition using the <posttest> tag.

Chapter 6 - Troubleshooting

Log Files

The log files are the most commonly used debugging tool. When you create a test script, be generous with the information written to the log file. Log files are created in the directory specified by the <logfileRootDir> in the XML file. Each test run creates a subdirectory with the name:

<xmlfilename>_<date/time>

where date/time is the date and time the lisa.ps1 script was started. The lisa.ps1 PowerShell script logs information in a log file named ica.log. Test scripts that run on the virtual machines have standard out and standard error redirected to a log file. This log file is collected when the test completes and stored in the log file directory. Log files collected from the virtual machines are named <VMName>_<testName>.log. For example, if the test ICTest was run on a virtual machine named SLES11, the log file would be named:

SLES11_ICTest.log

VM's Last State

Once all the tests have run on a virtual machine, the virtual machine is left in a stopped state. The VM was simply shutdown. If the failing test case was the last test to run, you can manually boot the virtual machine, log in as root, and look at the local copy of the log file, the test results, or even run the test manually. If you want to prevent any additional test from running after a failing test case, set the test case's <onError> value to Abort.

Running test scripts manually

To run a test case manually, you will need to have the constants.sh file on the VM. If this is not present, you can create one manually by creating and populating a file named constants.sh with the <testParams> from the test case definition in the XML file. Once you have created the constant.sh, just run the test case script. If you had to copy the test case script to the VM, you should consider running dos2unix to ensure the file has the proper EOL characters. In addition, you will probably need to change to mode to 755 with the chmod command.

Host side script do not use a constants.sh file. Rather, test parameters are passed to a host side PowerShell test script as a semicolon separated list of test parameters. For example, if the test case defines the test parameters as:

```
<testparams>
    <param>TC_COVERED=KVP-05</param>
    <param>Key=BBB</param>
    <param>Value=111</param>
    <param>rootDir=D:\lisa\trunk\lisa</param>
</testparams>
```

Then the $testParams variable will contain the following string:

```
"TC_COVERED=KVP-05;Key=BBB;Value=111;rootDir=D:\lisa\trunk\lisa"
```

An example of manually running a host side test script using the above test parameters would be:

```
.\setupScripts\myTestScript.ps1 –vmName myTestVM –hvServer localhost –
testParam "TC_COVERED=KVP-
05;Key=BBB;Value=111;rootDir=D:\lisa\trunk\lisa"
```

Note: Some of the PowerShell cmdlets used by test case scripts require Administrator access. Be sure your PowerShell session was started with the "Run as Administrator" option.

ISE

Another tool is ISE. This is the Integrated Scripting Environment. This tool will let you edit and run your PowerShell scripts. In addition, you can set breakpoints on individual lines of a script, examine variables, and other common debugging techniques. Stepping through your test script with ISE is a very efficient way to debug your test script.

Chapter 7 - Overview of Operation

The LISA scripts are built around the concept of a test. To perform a test, a number of steps need to be performed. In performing a test, a typical sequence of steps a VM might perform are:

- Ensure the VM is stopped.
- Run a Setup script to reconfigure the test VM.
- Start the test VM running.
- Wait for the VM to boot up.
- Push files to the VM.
- Start a test script running on a Linux VM.
- Collect files from the Linux VM.
- Shutdown the VM.
- Wait for the VM to shut down.
- Run a cleanup script against the VM.

LISA is based on a state engine. To perform the above steps, LISA will walk the VM through a number of states. The states are implemented by functions in the script stateEngine.ps1. To perform the above steps, the stateEngine.ps1 would walk the VM through the following states:

SystemDrive
RunSetupScript
StartSystem
SystemStarting
SystemUp
PushFiles
StartTest

TestStarting
TestRunning
CollectLogFiles
DetermineReboot
ShutdownSystem
ShuttingDown
RunCleanupScript
SystemDown

lisa.ps1

A test run is started by running the lisa.ps1 PowerShell script. The lisa.ps1 script performs the following:

- Validate the command line arguments.
- Checks that the specified XML file exists.
- Parses the XML file.
- Create the log file directory.
- Calls RunICTests() in stateEngine.ps1 to drive the virtual machines so they perform the tests specified in the XML file.
- If the -email option was specified, send email when the test run completes.

The process of parsing the XML file creates an in memory copy of the XML elements. This in memory copy is used as the data structure to keep track of the virtual machines and their state. Some elements of the in memory copy of the XML file will be updated as the virtual machines progress through their tests.

stateEngine.ps1

The heart of the LISA Automation is the state engine defined in stateEngine.ps1. This script defines a function to process each state a test VM may be assigned. The entry point to this script is the function RunICTests().

For each VM defined in the XML file, RunICTests() performs the following steps:

- Check if the virtual machine exists.
- If it does not exist
    - disable the virtual machine – no tests will be run on the VM.
- If it does exist
    - Add additional XML elements to the in memory XML data for the VM.
    - Ensure the virtual machine is in a Hyper-V Off state.
    - Reset the virtual machine to a snapshot named ICABase.
    - Set the VMs currentState to SystemDown.

Once this one time initialization is completed, RunICTests() will start the state engine by calling DoStateEngine().

The state engine will walk each virtual machine through a sequence of states required to perform the tests. The specifics of each state are describe in appendix B. The contents of the test case definition will influence the sequence of states a VM will transition through as it completes the test case.

The XML data for a virtual machine has an attribute named currentTest. This attribute is updated with the test the virtual machine is actually performing. Once all tests have been completed, the currentTest attribute is set to “done”. When the currentTest is set to “done”, the SystemDown state knows this virtual machine has completed all its tests, and will put the VM in the Finished state.

Each virtual machine also has an XML attribute named emailSummary. This attribute is modified by the functions RunICTest(), DoTestCompleted() and DoTestAborted(). As the virtual machine moves through its tests, information useful for an e-mail summary message is append to the emailSummary string. When all Virtual machines have completed their tests, the function SendEmail() will use information from the <global> section of the .xml file as well as the emailSummary information to create and send the e-mail status message.

Appendix

Appendix A - Sample XML file

```
<?xml version="1.0" encoding="utf-8"?>

<config>
    <global>
        <logfileRootDir>D:\lisa\TestResults</logfileRootDir>
        <defaultSnapshot>ICABase</defaultSnapshot>
        <email>
            <recipients>
                <to>myboss@mycompany.com</to>
                <to>myself@mycompany.com</to>
            </recipients>
            <sender>myself@mycompany.com</sender>
            <subject>LISA Test Run on WS2012R2</subject>
            <smtpServer>mysmtphost.mycompany.com</smtpServer>
        </email>
    </global>

    <testSuites>
        <suite>
            <suiteName>KVP</suiteName>
            <suiteTests>
                <suiteTest>KVP_Basic</suiteTest>
                <suiteTest>KVP_Add_Key_Values</suiteTest>
                <suiteTest>KVP_Modify_Key_Values</suiteTest>
                <suiteTest>KVP_Remove_Key_Values</suiteTest>
                <suiteTest>KVP_Push_Key_Values</suiteTest>
            </suiteTests>
        </suite>
    </testSuites>

    <testCases>
        <test>
            <testName>KVP_Basic</testName>
            <testScript>SetupScripts\KvpBasic.ps1</testScript>
            <timeout>600</timeout>
            <onError>Continue</onError>
            <noReboot>True</noReboot>
            <testparams>
                <param>rootDir=D:\lisa\trunk\lisablue</param>
                <param>TC_COVERED=KVP-01</param>
            </testparams>
        </test>

        <test>
            <testName>KVP_Add_Key_Values</testName>
            <testScript>VerifyKeyValue.sh</testScript>
            <files>remote-
scripts\ica\VerifyKeyValue.sh,tools/KVP/kvp_client</files>
            <PreTest>setupScripts\AddKeyValue.ps1</PreTest>
            <timeout>600</timeout>
            <onError>Continue</onError>
            <noReboot>True</noReboot>
            <testparams>
```

```
                <param>TC_COVERED=KVP-02</param>
                <param>Key=EEE</param>
                <param>Value=555</param>
                <param>Pool=0</param>
                <param>rootDir=D:\lisa\trunk\lisablue</param>
            </testparams>
        </test>

        <test>
            <testName>KVP_Modify_Key_Values</testName>
            <testScript>VerifyKeyValue.sh</testScript>
            <files>remote-
scripts\ica\VerifyKeyValue.sh,tools/KVP/kvp_client</files>
            <preTest>setupscripts\ModifyKeyValue.ps1</preTest>
            <timeout>600</timeout>
            <onError>Continue</onError>
            <noReboot>True</noReboot>
            <testparams>
                <param>TC_COVERED=KVP-03</param>
                <param>Key=EEE</param>
                <param>Value=999</param>
                <param>Pool=0</param>
                <param>rootDir=D:\lisa\trunk\lisablue</param>
            </testparams>
        </test>

        <test>
            <testName>KVP_Remove_Key_Values</testName>
            <testScript>setupscripts\DeleteKeyValue.ps1</testScript>
            <timeout>600</timeout>
            <onError>Continue</onError>
            <noReboot>True</noReboot>
            <testparams>
                <param>TC_COVERED=KVP-04</param>
                <param>Key=EEE</param>
                <param>Value=999</param>
                <param>Pool=0</param>
                <param>rootDir=D:\lisa\trunk\lisablue</param>
            </testparams>
        </test>

        <test>
            <testName>KVP_Push_Key_Values</testName>

<testScript>setupscripts\KVP_VerifyGuestCreatedItem.ps1</testScript>
            <files>tools/KVP/kvp_client</files>
            <timeout>600</timeout>
            <onError>Abort</onError>
            <noReboot>True</noReboot>
            <testparams>
                <param>TC_COVERED=KVP-05</param>
                <param>Key=BBB</param>
                <param>Value=111</param>
                <param>sshKey=rhel5_id_rsa.ppk</param>
                <param>rootDir=D:\lisa\trunk\lisablue</param>
            </testparams>
        </test>
    </testCases>
```

```
    <VMs>
      <vm>
            <hvServer>localhost</hvServer>
            <vmName>Sles11Sp3x64</vmName>
            <os>Linux</os>
            <ipv4></ipv4>
            <sshKey>rhel5_id_rsa.ppk</sshKey>
            <suite>KVP</suite>
        </vm>
    </VMs>

</config>
```

Appendix B - State Descriptions

This appendix lists each state a VM may be in, and the states that can be transitioned to. It does not list all the error handling a state may have, so some state transitions may not be listed. The file stateEngine.ps1 implements the functions that implement each state. The function names have "Do" prepended to the state name. For example, if you wanted to look at the function that implements the state SystemDown, look for the function named DoSystemDown.

SystemDown

- Make sure the VM is in a stopped state.
- Update the VMs currentTest value.
- If the current is "done"
  The VM has run all its test cases.
- Does the current test have a Setup Script?
- If yes
  Update the VMs state to RunSetupScript
  Else
  update the VMs state to StartSystem

RunSetupScript

- Verfiy the current test defines a setup script
- Run the setup script
- If the setup script returned an error
  Log a message and stop running test
  Else
  Update the VMs state to StartSystem

StartSystem

- Make sure the VM is in a stopped state.
- Start the VM.
- Allow 3 minutes for the VM to enter a Hyper-V Running state.
- If timeout starting the VM
  Log a message.
  Prevent the VM from running additional tests.
  Else
  Update the VMs state to SystemStarting.

SystemStarting

- Verify the VM is in a Hyper-V running state
- If the VM timed out in the SystemStarting state
  Update the VMs state to SlowSystemStarting
- Test port 22 on the VM to see if its SSH daemon is up.
- If the VM is listening on port 22
  Update the VMs state to SystemUp

SlowSystemStarting

- If the VM timed out in the SlowSystemStarting state
  Update the VMs state to DiagnoseHungSystem
- Test if the VM is listening on port 22.
- If port 22 is open on the VM

- Update the VMs state to SystemUp

DiagnoseHungSystem

- Set the currentTest to "done"
- Force the VM to stop.

SystemUp

- Use SSH to send an "exit" command to the VM and accept any prompt for a server key.

PushTestFiles

- Create a constants.sh file
- Add the global testParams to the constants.sh file
- Add test specific testParams to the constants.sh file
- Add VM specific testParams to the constants.sh file
- Use SSH to copy the constants.sh file to the VM.
- Send command to VM to run dos2unis on the constants.sh file
- Push files to the VM that are identified under the <files> tag of the test case definition. This includes the test script if it is a guest side test script.
- If the test case script ends in ".ps1"
  Update the VMs state to StartSP1Test
- Send command to VM to run dos2unix on the test script file.
- If current test defines a PreTest script
  Update the VMs state to RunPreTestScript
  Else
  Update the VMs state to StartTest

RunPreTestScript

- If the current test does not define a PreTest script
  Log a warning
  Update the VMs state to StartTest
- Run the pretest script
- If the pretest script returns an error
  Log a warning
- Update the VMs state to StartTest

StartTest

- Use the test script name and build a cmd string to run the test script. The command redirects stdout and stderr to a log file.
- Write the cmd string into a file named runtest.sh
- Use SSH to copy the runtest.sh script to the VM
- Use SSH to run dos2unix on the runtest.sh file
- Use SSH to chmod 755 runtest.sh
- Make sure the at daemon is running on the VM
- Submit runtest to at (at –f runtest.sh now)
- Update the VMs state to TestStarting

TestStarting

- Check if the VM has been in this state too long.
- If yes
  Log an error.
  Fail the test case.
  Update the VMs state to DetermineReboot
- Check if the test case script created the file ~/state.txt on the VM.
- If yes
  Update the VMs state to TestRunning

TestRunning

- Get the test case timeout value from the XML data.
- Has the test case timed out?
- If yes
  Log an error.
  Fail the test case
  Update the VMs state to CollectLogFiles
- Copy ~/state.txt from the VM
- If the contents of the file are not "TestRunning"
  If the contents of state.txt are "TestCompleted"
  Set testCaseResults to "Success"
  Update the VMs state to CollectLogFiles
  If the contents of state.txt are "TestAborted"
  Set testCaseResluts to "Aborted"
  Update the VMs state to CollectLogFiles
  If the contents of state.txt are "TestFailed"
  Set testCaseResults to "Failed"
  Update the VMs state to CollectLogFiles
  Else
  Abort the current test
  Update the VMs state to CollectLogFiles

CollectLogFiles

- Update the emailSummary test for the VM
- Copy the test case log file from the VM and store in the test run log directory.
- If ~/summary.log exists on the VM, add its contents to the VM's
- emailSummary.
- If the current test defines a postscript test
  Update the VMs state to RunPostTestScript
  Else
  Update the VMs state to DetermineReboot

RunPostTestScript

- If the current test does not have a posttest script
  Log a message
- Run the posttest script
- If the post script failed
  Log a message
- Update the VMs state to DetermineReboot

DetermineReboot

- Look at the test results, test case <onError>, and test case <noReboot> to determine if a reboot is required.
- If no reboot is required, but the next test has a setup script
  - Log an error
- If a reboot is required
  Update the VMs state to ShutdownSystem
  Else
  Update the current test
  Update the VMs state to SystemUp

ShutdownSystem

- Use SSH to send an "init 0" command to the VM.
- Update the VMs state to ShuttingDown

ShuttingDown

- Has the VM timed out in the ShuttingDown state
- If yes
  Update the VMs state to ForceShutdown
- Is the VM in a Hyper-V Off state
- If yes
  Does the current test have a cleanup script
  Update the VMs state to RunCleanupScript
  Else
  Update the VMs state to SystemDown

RunCleanupScript

- If the VM is not in a Hyper-V Off state
  Log an error
  Mark VM as done
  Update the VMs state to ForceShutdown
- Run the cleanup script
- If cleanup script reports an error
  Log an error
- Update the VMs state to SystemDown

ForceShutDown

- If current test has a cleanup script
  Set nextState to RunCleanupScript
  Else
  Set nextState to SystemDown
- Is the VM in the Hyyper-V off state
  Update the VMs state to nextState
  Else
  Force the VM to a Hyper-V off state.
  If not in off state after 3 minutes
  Log error
  Disable the VM
  Else
  Update the VMs state to nextState

Finished

- This state currently does not do anything.

StartPS1Test

- If test that the test case script does not exist
    Log error
    Write error to the test logfile
    Update the VMs state to PS1TestCompleted
- Build the testParams string from the <testParams>
- Add ipv4=theVMsIPv4address to the testParams string
- Add the VMs sshKey to the testParams string
- Submit the test script as a PowerShell job
- If a job ID was returned
    Update the VMs test state to PS1TestRunning
  Else
    Update the VMs test state to PS1TestCompleted

PS1TestRunning

- Get the test case timeout value
- Has the test case timed out
- If yes
    Abort the test
- Retrieve the PowerShell job ID
- If job state is Completed
    Update the VMs test state to PS1TestCompleted

PS1TestCompleted

- Receive the PowerShell job
- Create test logfile from the Job results
- Set test status as Failed
- If the last output of the test job was "True"
    Update test status as Success
- Update the VMs summary email string.
- Update the VMs state to DetermineReboot.

Appendix C - VM Provisioning

To use a virtual machine with LISA, it will need to be provisioned. The following tasks will need to be performed on the Linux VM.

- On a Hyper-V server, create the virtual machine.
- Install a supported version of Linux in the virtual machine.
- Install an SSH server.
- Configure the SSH server to start on boot.
- Enable root logins via SSH.
- Open port 22 on the firewall.
- Create an SSH key for use with LISA.
- Append the public key to the file /root/.ssh/authorized_keys
- Install the dos2unix utility.
- Install the atd utility.
- Configure the at daemon to start on boot.
- Once everything is working, shutdown the VM and use the Hyper-V Manager to take a snapshot of the virtual machine.
- Rename the snapshot to ICABase.