UEFI
Self-Certification Test (SCT)
Version 2.1 User Guide

Release Version 2.1
May, 2009

Revision History

Revision	Revision History	Date
2.1	Initial Release; release version number matches UEFI Specificaton number.	5/12/09

Contents

Figures

Tables

1 Introduction

1.1 Overview

The UEFI Self-Certification Tests (SCT) is a toolset for platform firmware developers to validate UEFI implementations on IA-32, X64, and Itanium Architecture-based platforms for compliance to the U*EFI 2.1 Specification*. The toolset features a Test Harness for executing built-in EFI Compliance Tests, as well as for integrating user-defined tests that were developed using the UEFI SCT open source code.

The UEFI SCT Test Harness provides two different usage models as native mode and passive mode. Please note that most network-related protocols (except SNP & PXEBC) can be tested only in passive mode. In passive mode UEFI SCT, the network-related protocol testing is included.

This document also provides descriptions of the IHV SCT. The IHV SCT is designed to aid the testing of UEFI drivers that follow the UEFI Driver Model described in the *UEFI 2.1 Specification*. There are several different classes of UEFI drivers, each with many variations. Also, this document provides guidelines on testing for Independent Hardware Vendors (IHV) for UEFI 2.1 Specification Complacency.

1.2 System Requirements

The UEFI SCT must be executed on a target system that meets the following requirements:

- The target system must have an Itanium Architecture-based platform, an X64 platform, or an IA-32 platform.
- The target system firmware must have EFI implemented per the UEFI 2.1 Specification.
- The EFI implementation on the target system must include an EFI Shell.
- The target system must have at least 100MB of disk space in the EFI file system to contain the SCT test and log files.

The UEFI SCT must have another host machine for passive mode usage. This machine must the following requirements:

- Installing Microsoft Windows 2000® or Microsoft Windows XP® operating system
- The target machine and host machine must be connected by network devices such as a switch/hub, etc.

Refer to the latest *UEFI SCT Release Notes* for other possible system requirements.

1.3 Installation

A typical installation of the UEFI SCT involves the following:

- Ensuring that the target system is configured to boot to the EFI Shell upon power-on/reset without user intervention.

  Setting the boot options is usually done using EFI Boot Manager during the target system's EFI implementation.

- Copying the UEFI SCT executable files into a default directory in the EFI file system of the target system.

  The default directory is where the target system automatically boots to after bringing up the EFI Shell. The default directory must be on a Read/Write storage medium.

The UEFI SCT comes in three versions: one for Itanium Architecture platforms, one for X64 platforms and another for IA-32 platforms. In general, all three versions bundled with each UEFI SCT release. The user must ensure that the appropriate version of the UEFI SCT is installed on the target platform prior to use.

The above is a general description of the UEFI SCT installation process. Detailed installation instructions are provided in the UEFI SCT Release Notes that accompany each UEFI SCT release. The person performing the installation must make sure that the UEFI SCT Release Notes match the UEFI SCT release being used.

2 Usage Model – Native Mode

The native mode is invoked as an EFI application from the EFI Shell. The executable filename is **SCT.efi**. This executable provides a command line interface (CLI) as well as a menu-driven interface. These are further described below.

2.1 Using the Command Line Interface

Syntax

```
SCT [-a | -c | -s <seq> | -u] | -p <MNP | IP4 | Serial>] [-r] [-g
<report>]
```

Description of SCT Parameters

Table 1 provides a description of SCT parameters.

Table 1. SCT Parameters

Options	Description
-a	Execute all test cases that are recognized by the UEFI SCT Test Harness.
-c	Continue execution of the test case in progress. This option is used to continue execution of test cases that perform system resets as part of their test routine.
-g <report>	Generate test report in .CSV format. The filename of the report is specified by **report**.
-r	Resets the environment for a fresh execution of the tests. This option removes results of previous test executions. Generally, it is used with the **-a** or **-s** options.
-s <seq>	Execute test cases in the sequence specified in the file **seq**.
-u	Start the Test Harness with the menu-driven interface.
-p	Passive Mode with specified communication layer
-f	Force the operation execution, no confirmation from user.

Selecting SCT without parameters will produce the screen display shown in Figure 1.

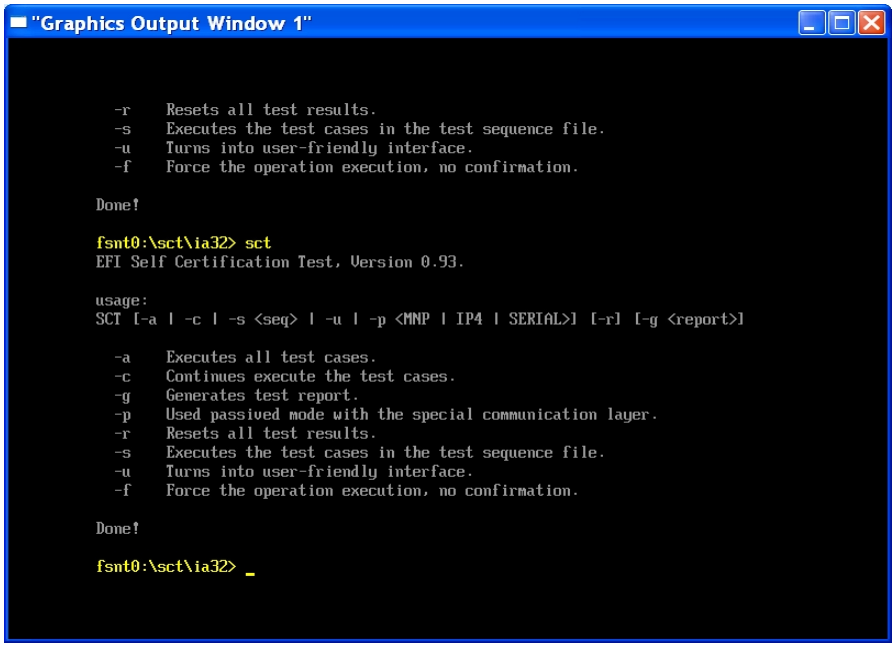


Figure 1. SCT without Parameters Screen Display

2.2 Using the Menu-Driven Interface

Syntax

```
SCT -u
```

Description

Type **SCT -u** to produce the Main Menu of the menu-driven interface.

2.2.1 Main Menu

The Main Menu (see Figure 2) contains user-selectable items for initiating a number of UEFI SCT actions.

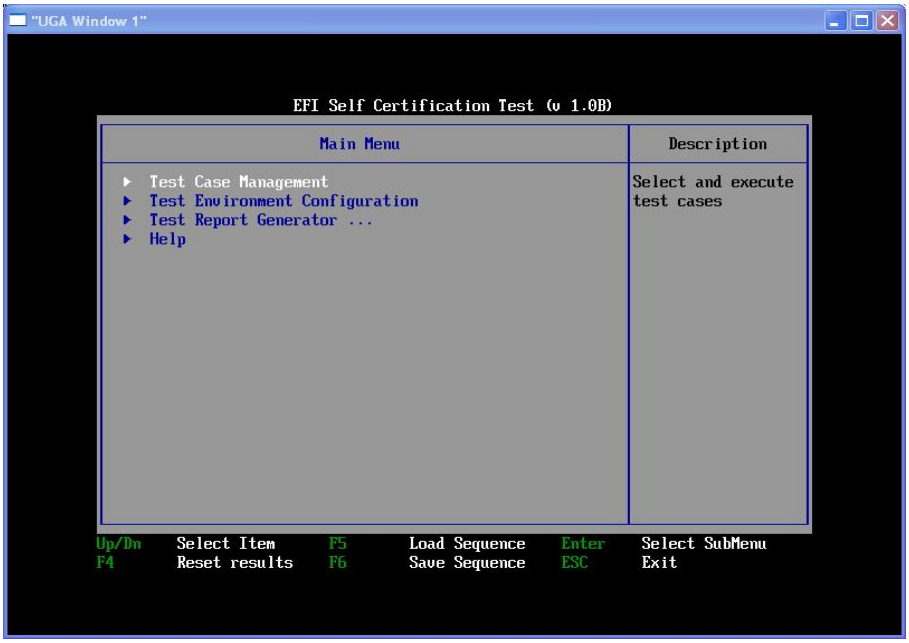


Figure 2. Main Menu Screen.

Table 2 lists and describes the major items found in the Main Menu.

Table 2. Major Items in the Main Menu of the SCT

Items	Description
Test Case Management	Selects and executes specific test cases
Test Environment Configuration	Sets the parameters for test execution, including the maximum run times for each test case, enabling/disabling screen output, etc.
Test Report Generator	Generates a test report in .CSV format. This test report can be opened by the Microsoft® Excel* or the other compatible utilities.
F4 (Reset Results)	Resets all test results. It is equivalent to invoking "**SCT -r**" in the command line.
F5 (Load Sequence)	Loads a test sequence file from the storage device. This function allows user to load, edit or execute an existing test sequence file.
F6 (Save Sequence)	Saves a user-specified test sequence into a file. This function allows the user to save selected test cases into a file, which can then be used for later test execution via "**SCT -s <seq>**" from the command line.

2.2.2 Managing Test Cases

The UEFI SCT includes a set of test cases for *UEFI 2.1 Specification* compliance testing. Note that in Figure 3 the list of test cases corresponds to the major elements of EFI as described in the *UEFI 2.1 Specification*. Note how in Figure 4 each test case can have lower-level test cases in a tree-like structure.

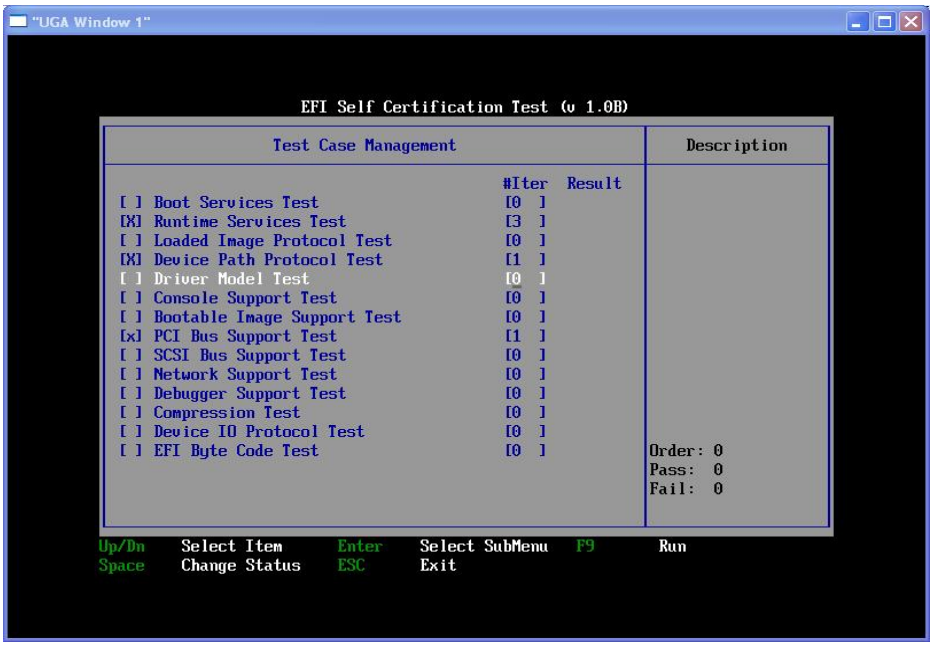


Figure 3. Test Case Management Screen.

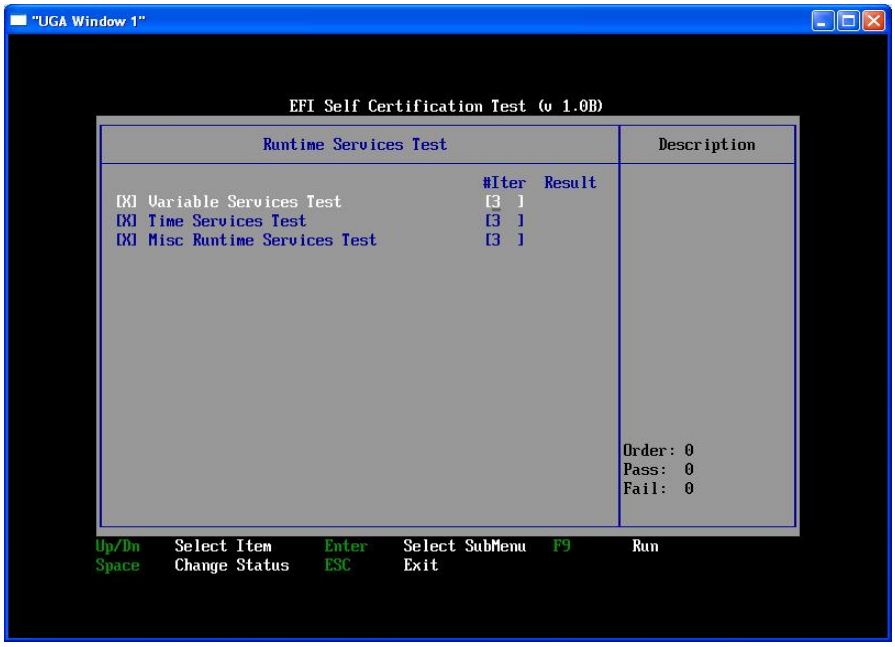


Figure 4. Run Time Services Screen.

Appendix B describes the method to specify such a tree-like hierarchy of tests for user-defined test cases. Refer to the *UEFI SCT Test Writer's Guide* for information on developing user-defined test cases.

In the menu-driven interface, the boxes on the left indicate the selected or unselected status of the corresponding test category.

[X] All test cases in this test category are selected.

[x] One or more test cases in this test category are selected, but not all test cases.

[ ] No test case in this test category has been selected.

The middle boxes below **#Iter** indicate the number of iterations to be executed for the corresponding test category.

[N] All test cases in this test category will be executed N times.

[*] The test cases in this test category have different numbers of execution iterations.

The summary results (**result**) show the execution results for the corresponding test category.

PASS All test assertions for all test cases in this test category have passed.

FAIL One or more test assertions within the test cases for this test category has failed.

test One test case in this test category was still executing.

" " No test case in this test category was executed.

The number of passed test assertions and the number of failed test assertions is displayed in the lower right corner as shown in the screenshot above. Note that the test case's place in the order of execution is also displayed. Note also that the order of execution of test cases is based on the user's order of selection of test cases to execute.

2.2.3 Configuring the Test Environment

The test environment has user-configurable items for set up, as shown in the screen display in Figure 5.

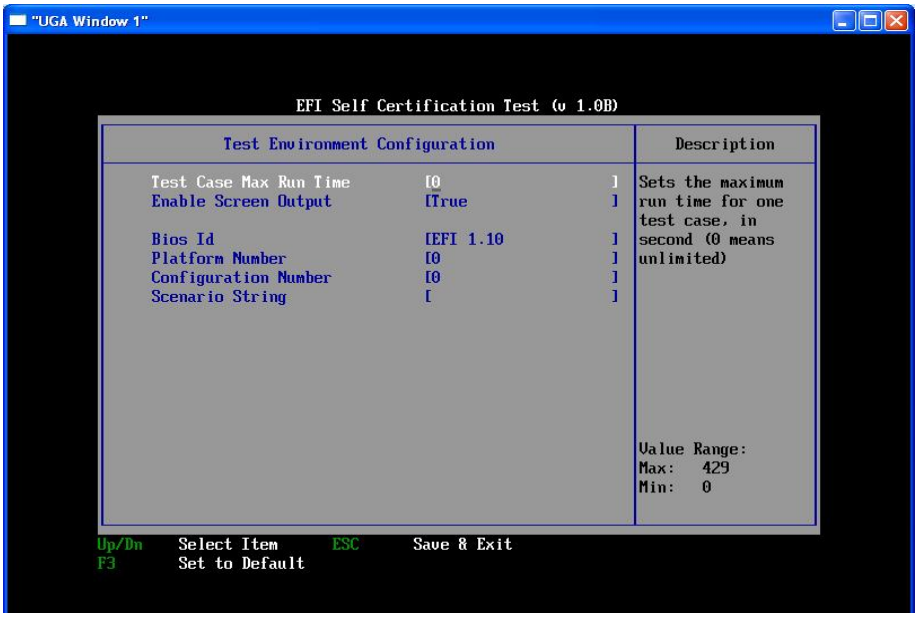


Figure 5. Test Environment Configuration.

Table 3 describes the user-configurable items for setting up the test environment.

Table 3. User-Configurable Items for Setting Up the Test Environment

Items	Description
Test Case Max Run Time	Sets the maximum execution time for the specified test case. This feature helps prevent system hangs that may occur during execution of a particular test case from indefinitely suspending the entire SCT execution run. Basically, a watchdog timer is set for every test case during execution. If the timer expires, the system automatically restarts and SCT execution automatically continues starting with the next test case in the order of execution.
Enable Screen Output	Enables/disables display of test log information on the screen.

Bios Id	A string that can be used to identify the BIOS or firmware stack of the target system under test. This information will be included in the log files of the test execution. Generally, Bios Id is used in conjunction with the other strings (identified in this table) for specifying user-controlled parameters for the test execution.
Platform Number	A number to identify the platform under test. (e.g., 865, 915). This information is included in the log files of the test execution. Generally, Platform Number is used in conjunction with the other strings (identified in this table) for specifying user-controlled parameters for the test execution.
Configuration Number	A number to specify the configuration under test. The numbers used to identify different configurations are entirely up to the user. Generally, a standard configuration is set as 0, a full configuration is set to 1, and so on. This information will be included in the log files of the test execution. Generally, Configuration Number is used in conjunction with the other strings (identified in this table) for specifying user-controlled parameters for the test execution.
Scenario String	A string to provide additional information about or further description of the test scenario for the next test execution. This information is included in the log files of the test execution. Generally, this is used in conjunction with the other strings (identified in this table) for specifying user-controlled parameters for the test execution.

2.2.4 Generating a Test Report

As shown in Figure 6 below, the user specifies the file name of the test report to be generated for the test execution. The test report file is created in the same directory where the SCT was invoked.

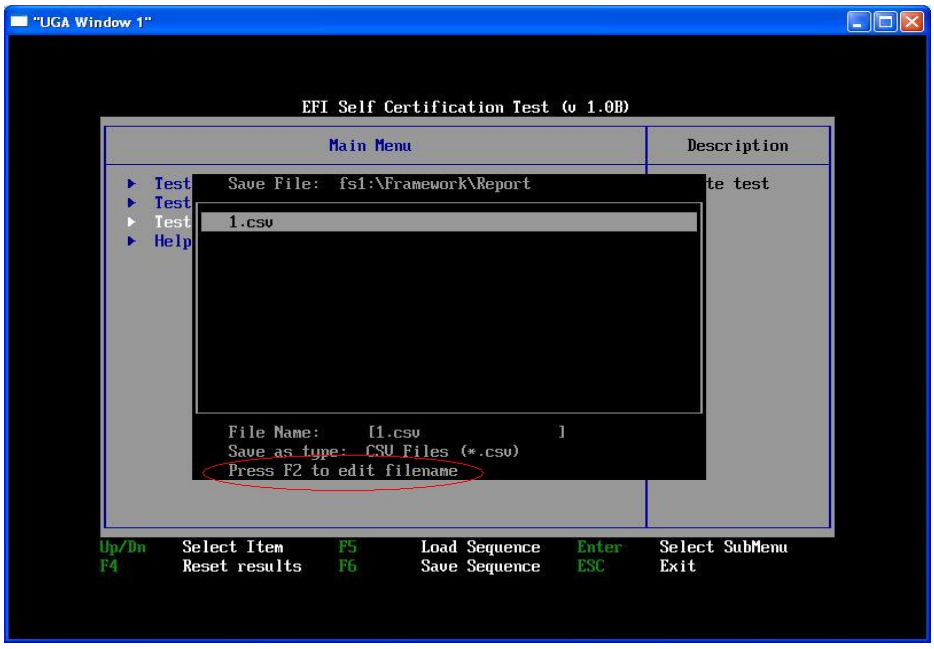


Figure 6. Generating a Test Report.

Note: *The <F2> key is used to move the cursor between the list box and the edit box.*
The <TAB> key is used in the EFI Shell to pause execution of an EFI application.

2.2.5 Loading and Saving a Test Sequence

To load a test sequence file, press the <F5> key (see Figure 7). To save a test sequence file, press the <F6> key (see Figure 8). The test sequence file is created in the same directory where the SCT was started.

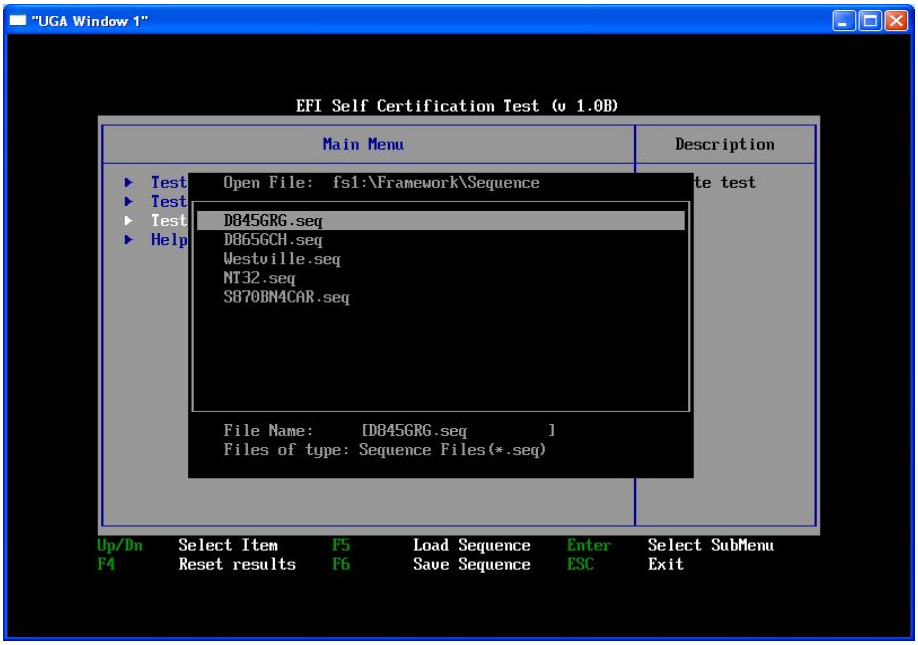


Figure 7. Press the <F5> Key to Load a Test Sequence.

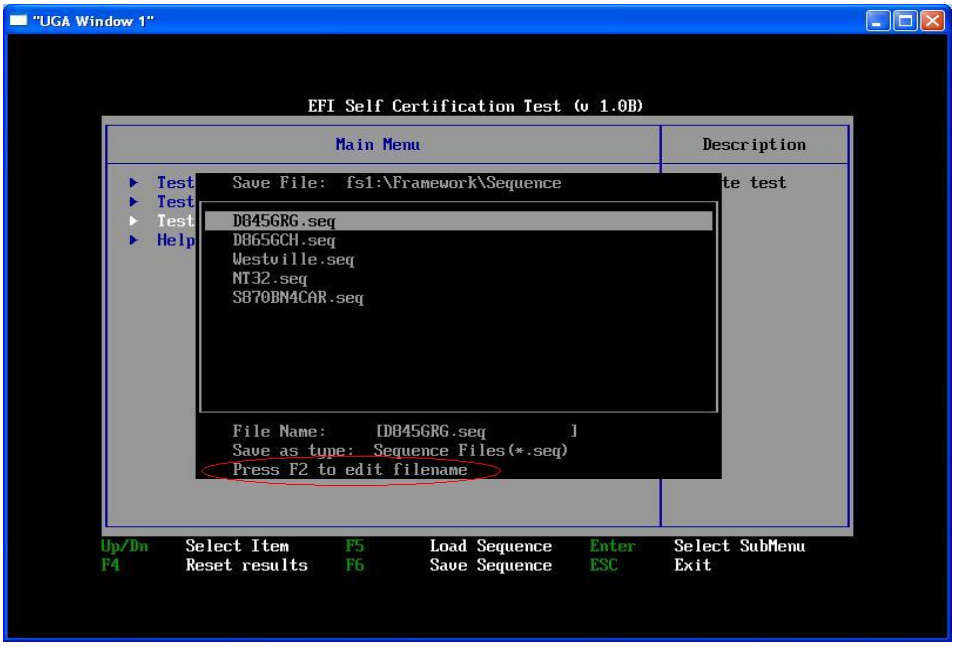


Figure 8. Press the <F6> Key to Save a Test Sequence.

2.3 Sample Usage Models

2.3.1 Executing from the Command Line Interface

1. To select the test cases to execute, invoke **`SCT –r –u`** from the EFI Shell.
2. To save the test case selection into a test sequence file, press <F6>.
3. To start the test execution, return to the EFI Shell and invoke **`SCT –s <seq>.`**
4. When test execution completes, invoke **`SCT –g <report>`** to generate the test report.

Note: *If the same test execution is to be repeated, Steps 1 and 2 can be skipped.*

2.3.2 Executing from the Menu-Driven Interface

1. Invoke SCT –r –u from the EFI Shell; select the test cases to execute.
2. Press <F9> to start test execution.
3. When test execution completes, select “Test Report Generator” to generate the test report.

2.4 Frequently Asked Questions

2.4.1 Stopping Automatic Test Execution When the System Restarts

The UEFI SCT Test Harness uses a startup script to continue test execution automatically when the system restarts. As shown in Figure 9, the startup script prompts the user to stop the Auto Run by pressing any key. (The user is given only a few seconds to press any key.) After canceling an Auto Run, the user can manually restart the test execution by typing **`startup.nsh`** or **`sct -c`**.

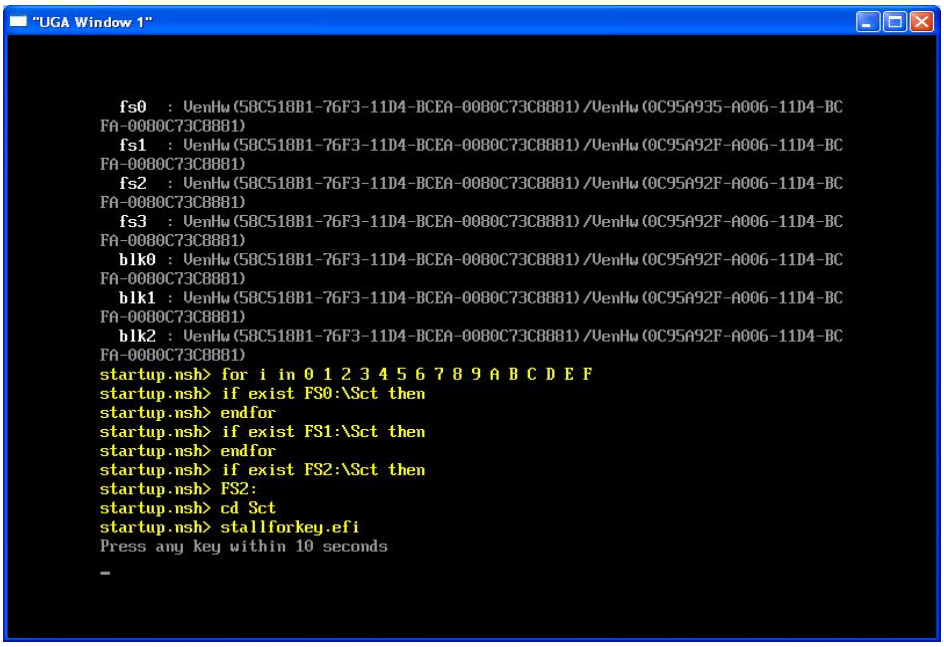


Figure 9. Press any Key within 10 Seconds to Stop the Auto Run.

2.4.2 Stopping SCT Execution While Tests Are Running

The user can manually reset the system to force a test execution to stop. In this case, a message of “system hangs or stops abnormally” is recorded for the interrupted test (see Figure 10), and the interrupted test is skipped and continued in the next restart of test execution.

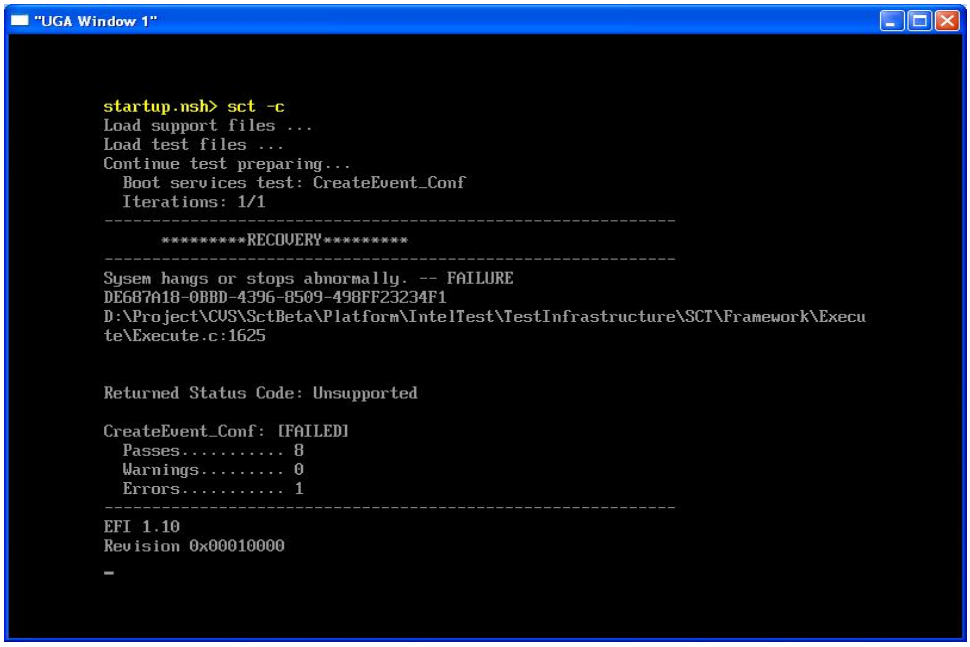


Figure 10. System Reset Records Message: "System Hangs or Stops Abnormally".

2.4.3 Removing a Test Case that Always Causes the System to Hang

A test case can be disabled using the menu-driven interface. This is useful when the user needs to disable, or to re-enable, test cases after manually stopping an Auto Run that was causing the system hang. If the test case has been executed after being disabled, there will be no effect on the test execution or to the test results. If the execution of the test case to be disabled is incomplete, or is waiting its turn in the order of execution, the test case is skipped when test execution is continued.

The following are screenshots showing the steps to removing a test case:

1. Press any key to stop Auto Run.

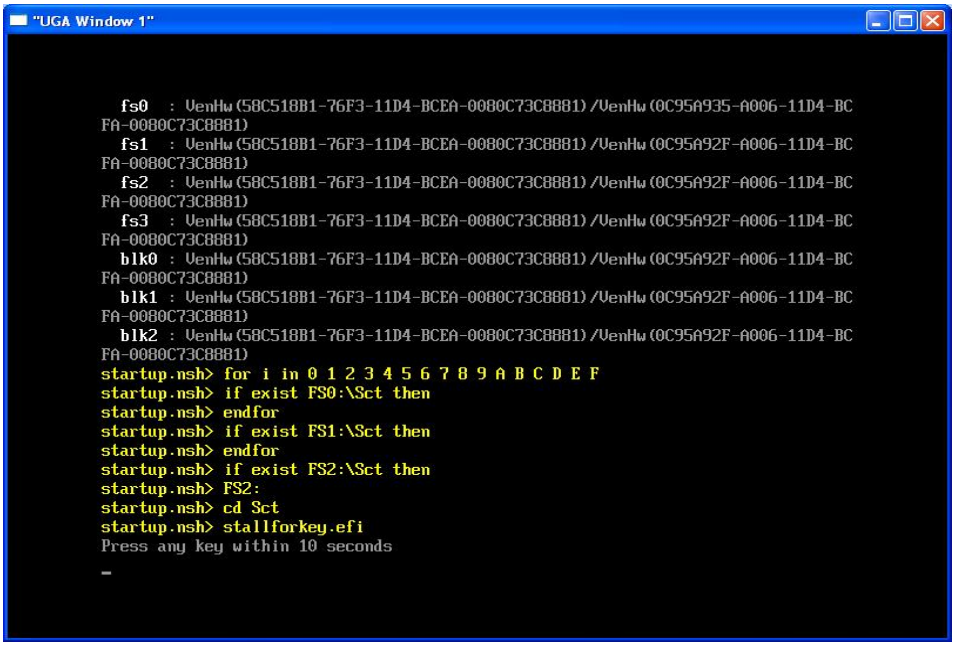


Figure 11. Press any Key to Stop Auto Run.

2. Type **`Sct -u`** to bring up the Menu-driven interface. Select [No] to discontinue execution.

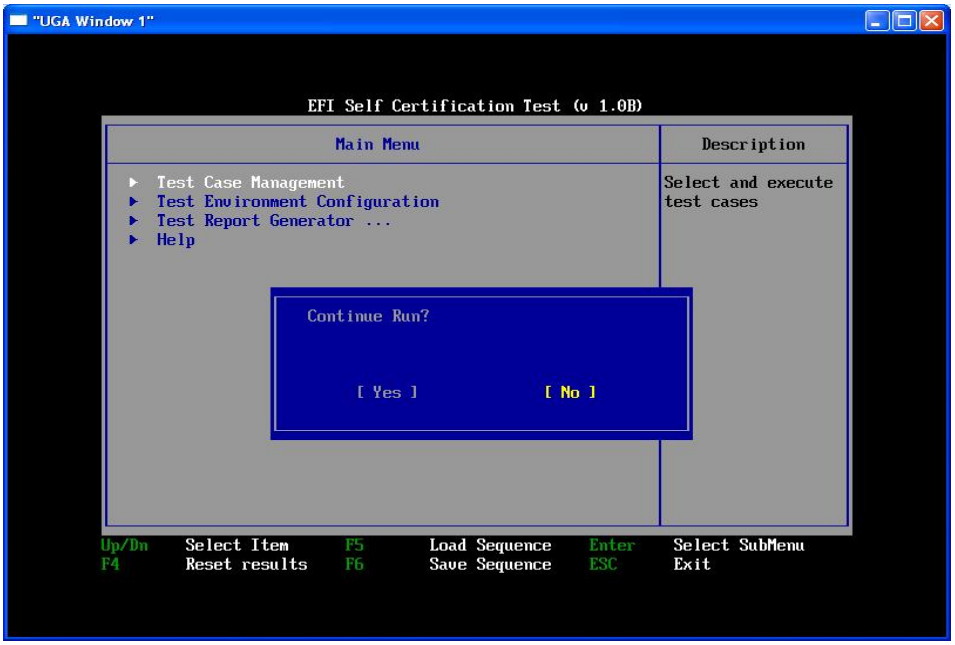


Figure 12. Select [No] to Discontinue Execution.

3. Remove the test using Test Case Management. Press <F8> to continue execution. Press <SPACE> to deselect the test. This effectively removes the test from the execution run.

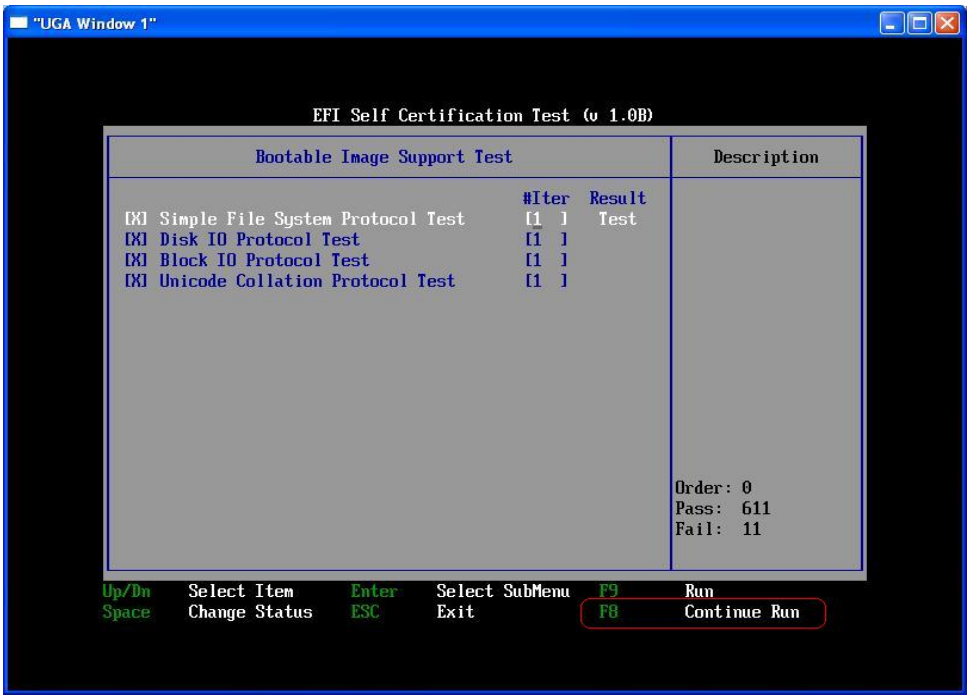


Figure 13. Press <SPACE> to Deselect the Test.

2.4.4 When There Are No Test Results after Test Execution

Some tests may not have results in the menu-driven interface or in the test report even after execution. There are two possible reasons for this.

1. The test is unable to execute at all. For example, the “Network Support Test” will not execute on a platform that has no network devices.
2. The test case does not record results in conformance to the UEFI SCT Test Development Kit. A user-defined test case can generate its own test output independent of the UEFI SCT test output format.

2.4.5 When Test Assertion Totals Are Different on Different Platforms

The total numbers in the UEFI SCT test reports show the total number of passed test assertions as well as failed test assertions. The number of applicable test assertions depends on the results of checkpoints in the tests. Platforms of different configuration or devices will cause different results for these checkpoints, and thus different sets of applicable test assertions. For example, the Block I/O test will verify the Read-Only capability when there is a CD in the CD-ROM drive. Another example is when the PCI

test verifies resource allocation only if a PCI device requires memory-mapped IO space.

3 Usage Model – Passive Mode

The UEFI SCT Agent runs in the passive mode. All the test cases can be run on the UEFI Management Side (EMS) with the UEFI SCT Agent running in the passive mode.

Note: *The following description assumes the user has built the environment on both UEFI SCT Agent side and EMS side.*

3.1 Configuring UEFI SCT Agent

This section describes the steps that are necessary to configure the UEFI SCT Agent side. The SCT Package will be produced after building the UEFI SCT and then need to be installed onto the target machine for test. The directory SctPackage will be created under the IA32, the X64, or the IPF efi/uefi build directories. Refer to Section 1.3.

The following screenshots show the steps to make UEFI SCT Agent run in the passive mode.

1. Install UEFI SCT Agent. Refer to Chapter 5 in the "UEFI SCT Getting Started" document.
2. Switch to the EFI shell.
3. When the target machine starts, select the menu Boot Manager as shown in Figure 14. Select Boot Manager.
4. Select the menu “Internal EFI Shell” in the Boot Manager as shown in Figure 15. Select Internal EFI Shell.

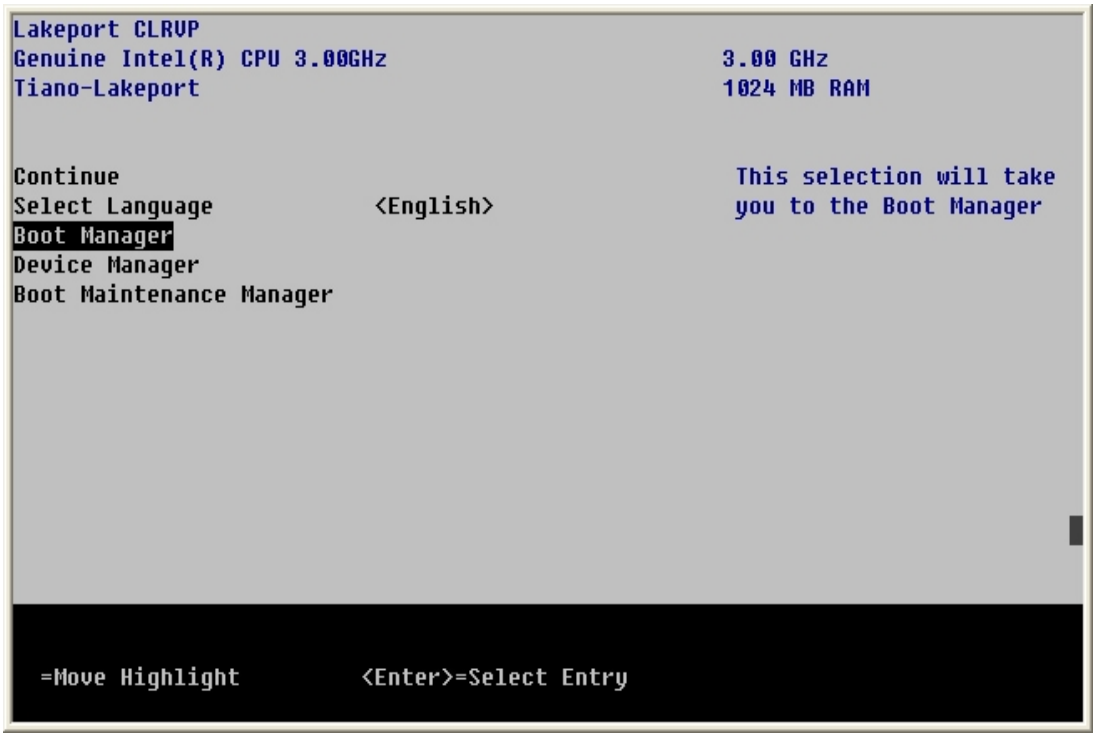


Figure 14. Select Boot Manager.

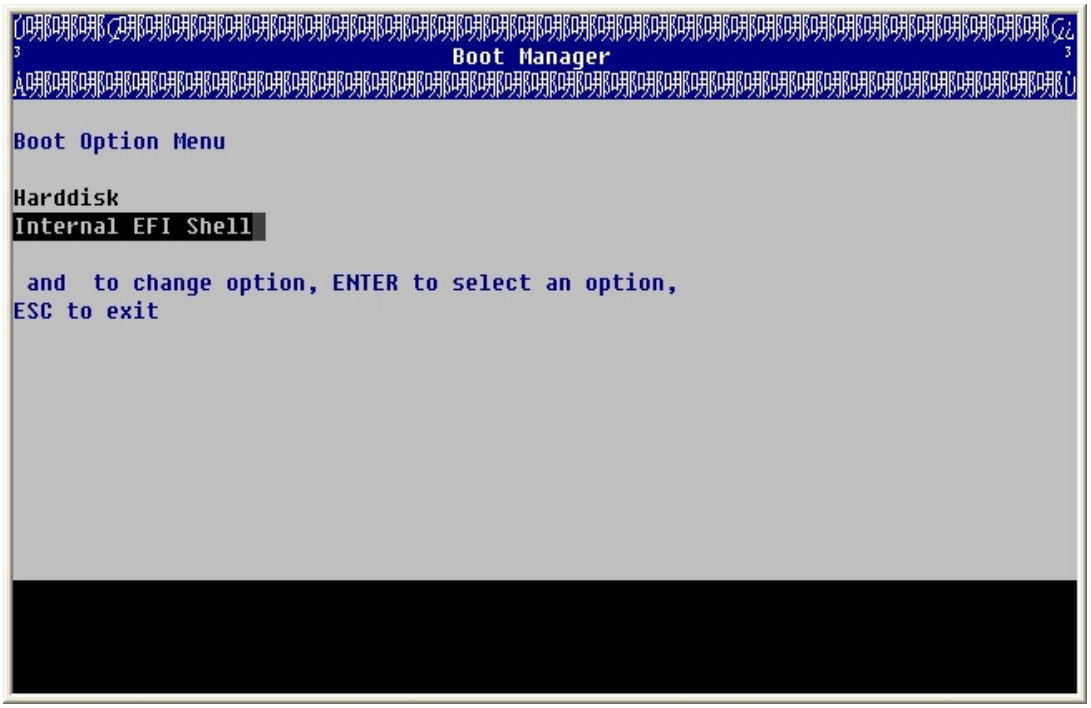


Figure 15. Select Internal EFI Shell.

To use SCT passive mode after installing UefiSctAgent, obtain the necessary network drivers and modify the startup.nsh according to the machine's configuration:

3. Put all network drivers under the "NetworkDrivers" folder of the installation disk of SCT. In this example, we assume the folder is "NetworkDrivers". You can choose any location you like.

4. Open startup.nsh under the installation disk of SCT.

5. Go to the line "if exist FS%i:\Sct\.passive.mode then", insert the following lines after it:

```
load \NetworkDrivers\Undi.efi
load \NetworkDrivers\Snp.efi
load \NetworkDrivers\Mnp.efi
load \NetworkDrivers\Arp.efi
load \NetworkDrivers\Ip4.efi
load \NetworkDrivers\Ip4Config.efi
load \NetworkDrivers\Udp4.efi
load \NetworkDrivers\Dhcp4.efi
load \NetworkDrivers\Mtftp4.efi
load \NetworkDrivers\Tcp4.efi
```

For configurations with network drivers, follow the steps below to enter SCT passive mode.

1. Assume all network drivers are under the "NetworkDrivers" folder of the installation disk of SCT.

2. In the EFI shell environment, load all network drivers as shown in Figure 16 and Figure 17. (One may also write an nsh script to load the network drives.)

```
load \NetworkDrivers\Undi.efi
load \NetworkDrivers\Snp.efi
load \NetworkDrivers\Mnp.efi
load \NetworkDrivers\Arp.efi
load \NetworkDrivers\Ip4.efi
load \NetworkDrivers\Ip4Config.efi
load \NetworkDrivers\Udp4.efi
load \NetworkDrivers\Dhcp4.efi
load \NetworkDrivers\Mtftp4.efi
load \NetworkDrivers\Tcp4.efi
```

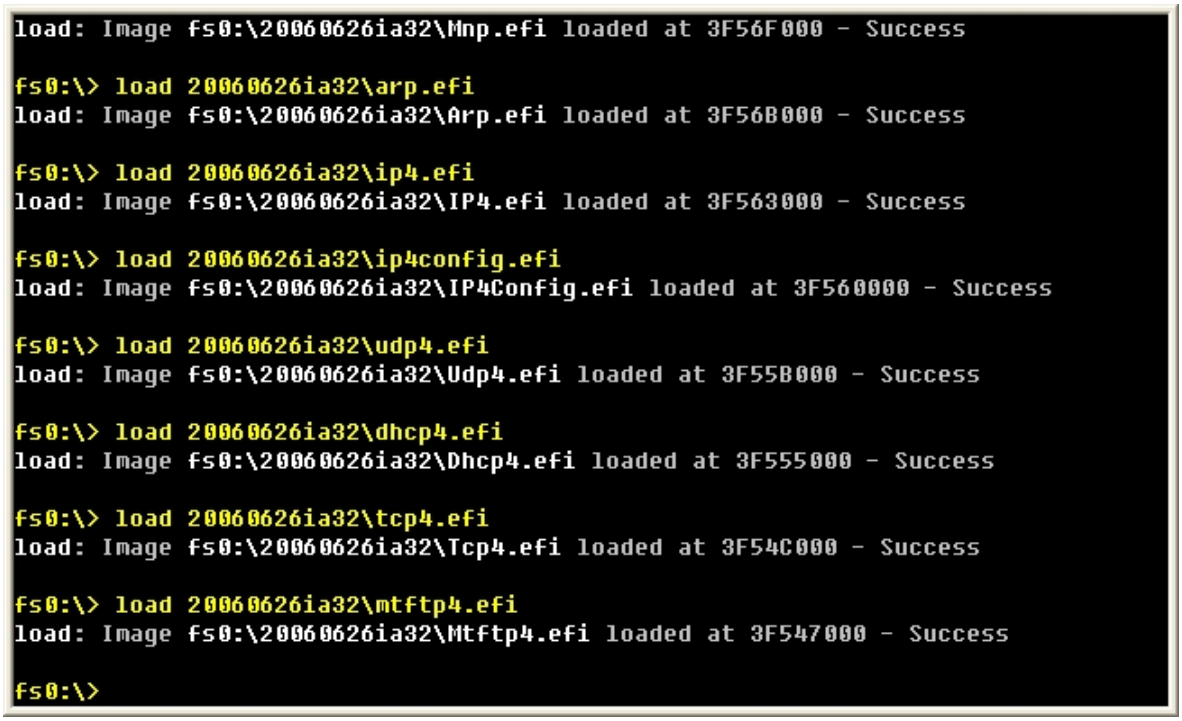


Figure 16. Load Network Drivers in Internal EFI Shell.

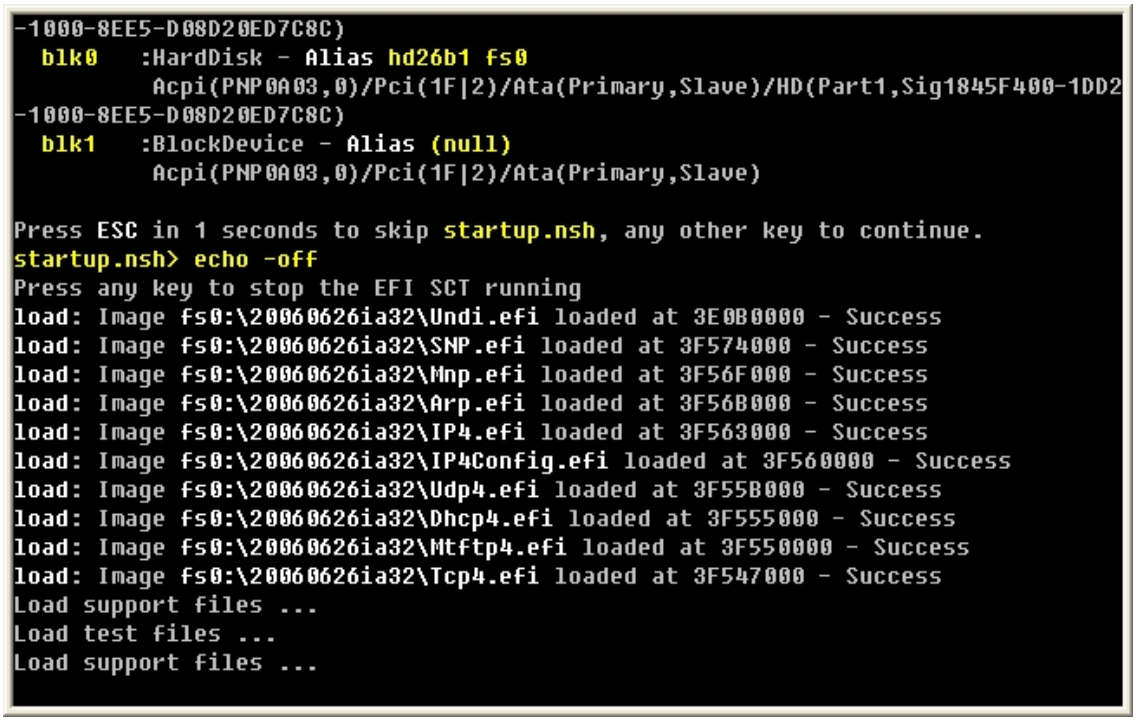


Figure 17. Auto Load Network Drivers by modifying startup.nsh script.

3. Enter the SCT folder and type **`sct -p mnp`** to run the SCT passive mode and choose the NIC as shown in Figure18 that will be used for communication between test machine and host machine.

4. Choose the NIC as shown in Figure19 that will be used for communication between test machine and host machine.

5. Type **`cd sct`** and **`sct`** to show all the SCT commands as shown below:

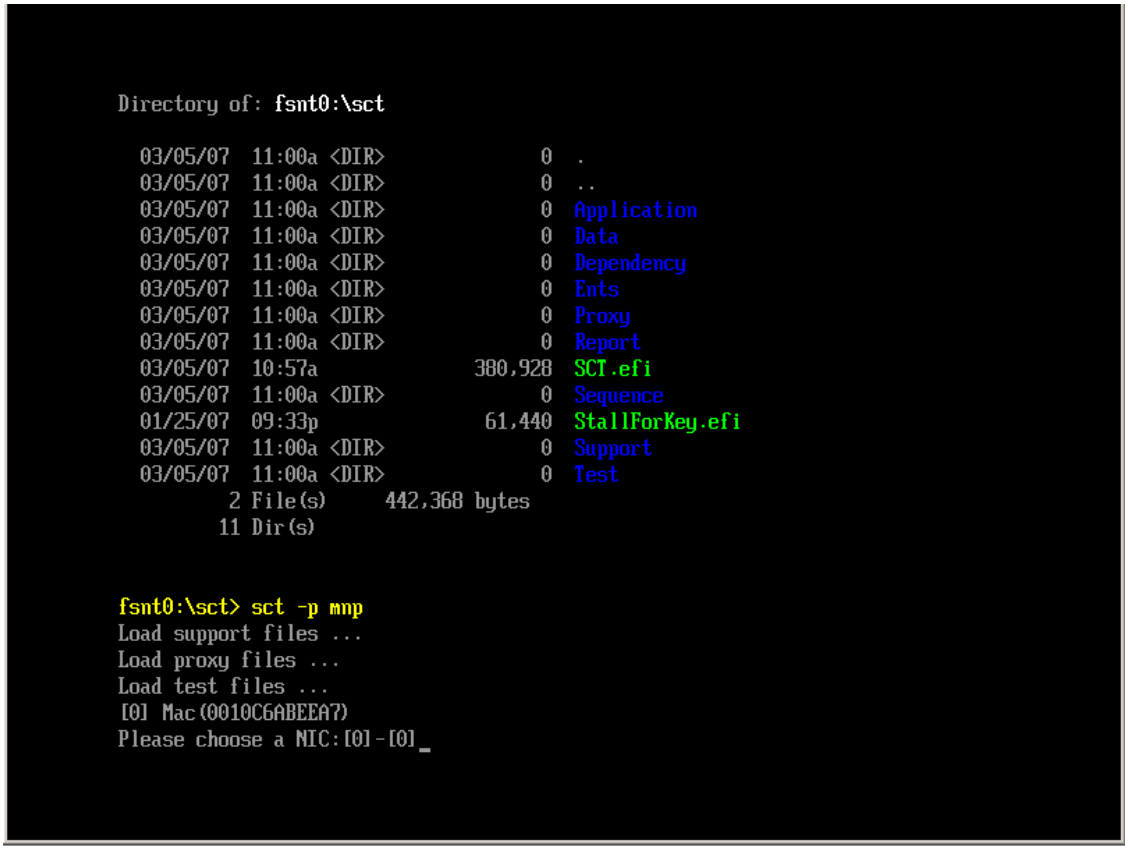


Figure 18.Choose the NIC.

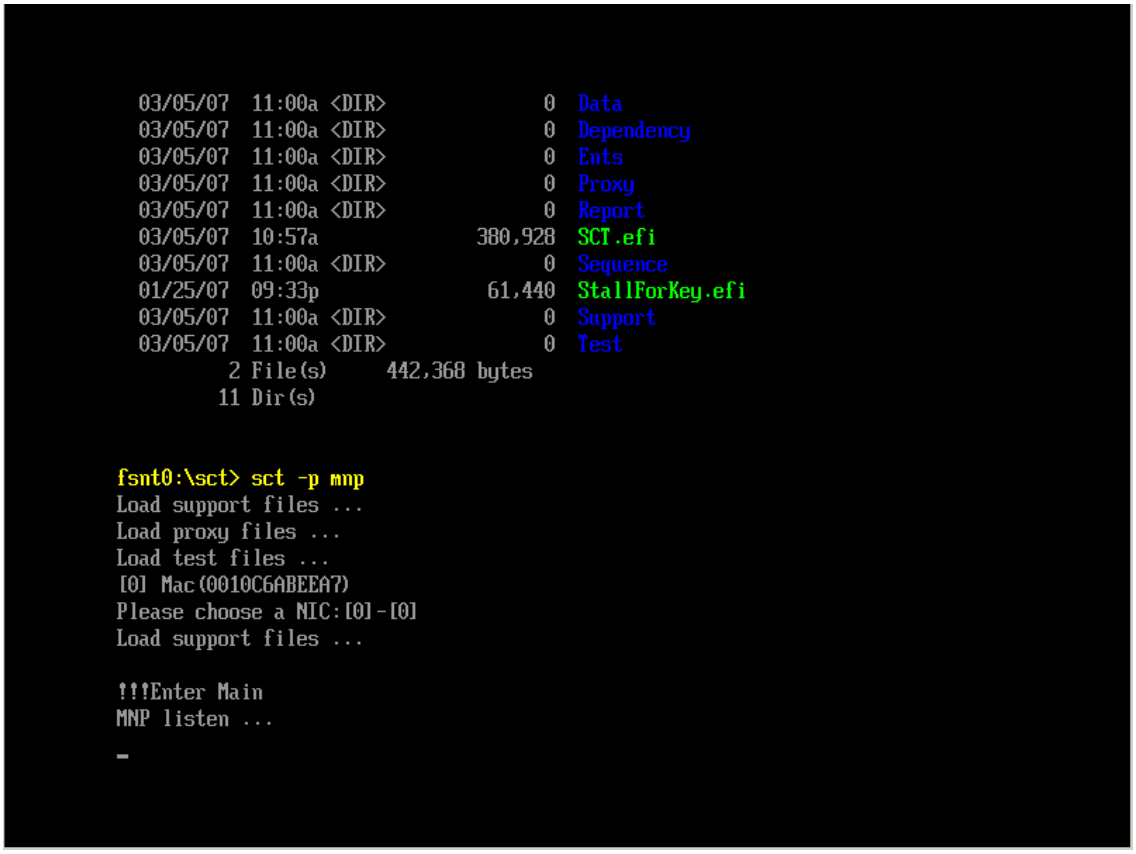


Figure 19. Using SCT Passive mode

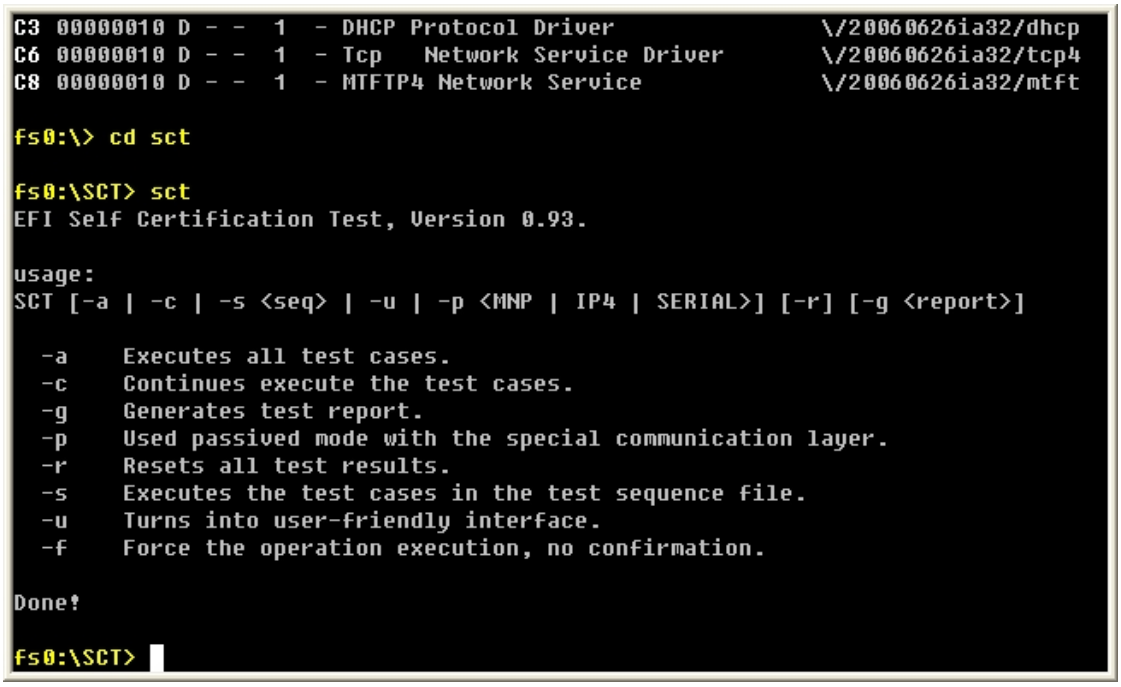


Figure 20. All the SCT commands

Note: *Systems without network drivers cannot use SCT passive mode, but you can use the compatible usage as EFI SCT. Refer to chapter 2.*

When running UEFI SCT Remote Validation, it is important to keep the test topology environment clean. For example, use one switch (hub) to connect the EFI target machine and the management host machine, but don't connect the switch (hub) to a public network or other LANs.

To run UEFI SCT with local execution usage, make sure the "\Sct\.passive.mode" file is removed.

3.2 Configuring EMS

The EMS side provides a Graphic User Interface (GUI) to run all the test cases. This section describes the steps that are necessary to configure the EMS side and all the menu functions in the EMS OS application window.

3.2.1 Configuring the EMS Interface

Run the Visual Studio .NET 2003 Command Prompt to go to the command line environment. Use the following commands to run the EMS OS application.

1. **cd \test\ems\bin**
2. **Ems Main.Tcl**

When the EMS OS application starts, two windows open. Before the main window is available, choose the host interface in the EMS Interface Configuration window. If there are more than one Network Interface cards on your local host, you need to specify the one connected to the EFI target machine. Figure 21 shows the EMS Interface Configuration window.

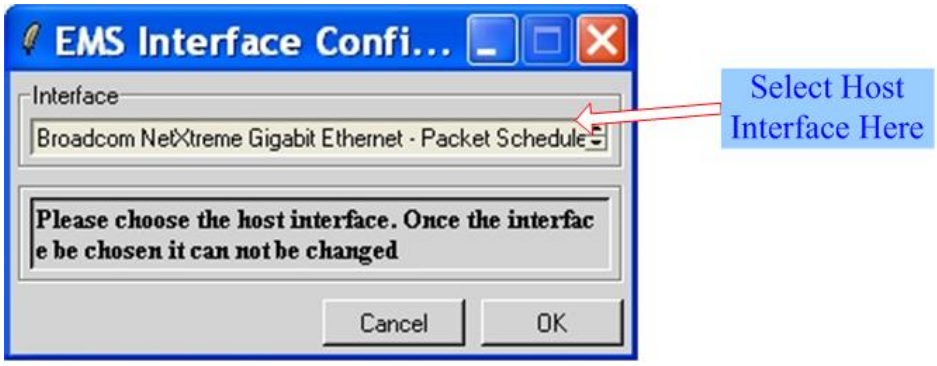


Figure 21. EMS Interface Configuration window.

3.2.2 Configuring Base Information

If you are starting the EMS OS application for the first time, configure the base information. Select the menu "File->Preference...". The "EMS Preference" window opens. The following list describes each item in the window. The "EMS Preference" window is shown in Figure 22.

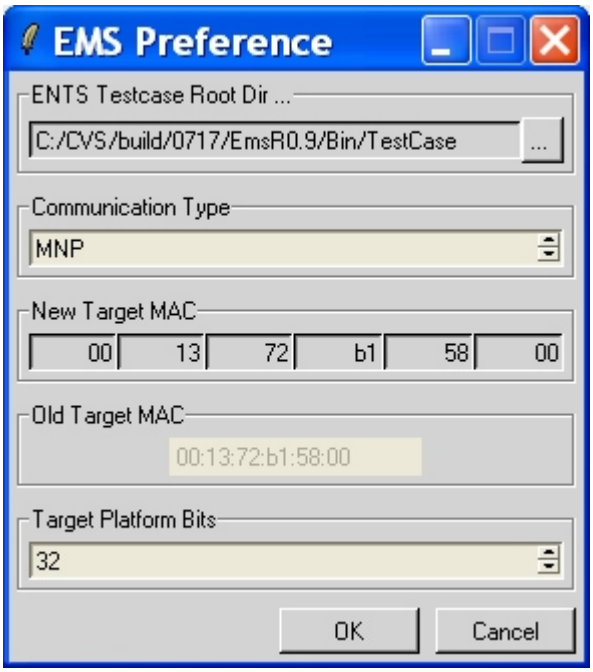


Figure 22. EMS Preference window.

1. ENTS Testcase Root Dir...

   This item refers to the root directory of all the Remote Validation test cases. Press the Browse button on the right to choose the root directory of the Remote Validation test cases.

2. Communication Type

   This item refers to the communication type between the EMS side and the UEFI SCT Agent side. Currently, MNP is the only supported communication type.

3. New Target MAC

   This item refers to the target host MAC address you want to configure. You can type **dh -p net** in the EFI Shell to get the target host MAC address as shown in Figure 23.

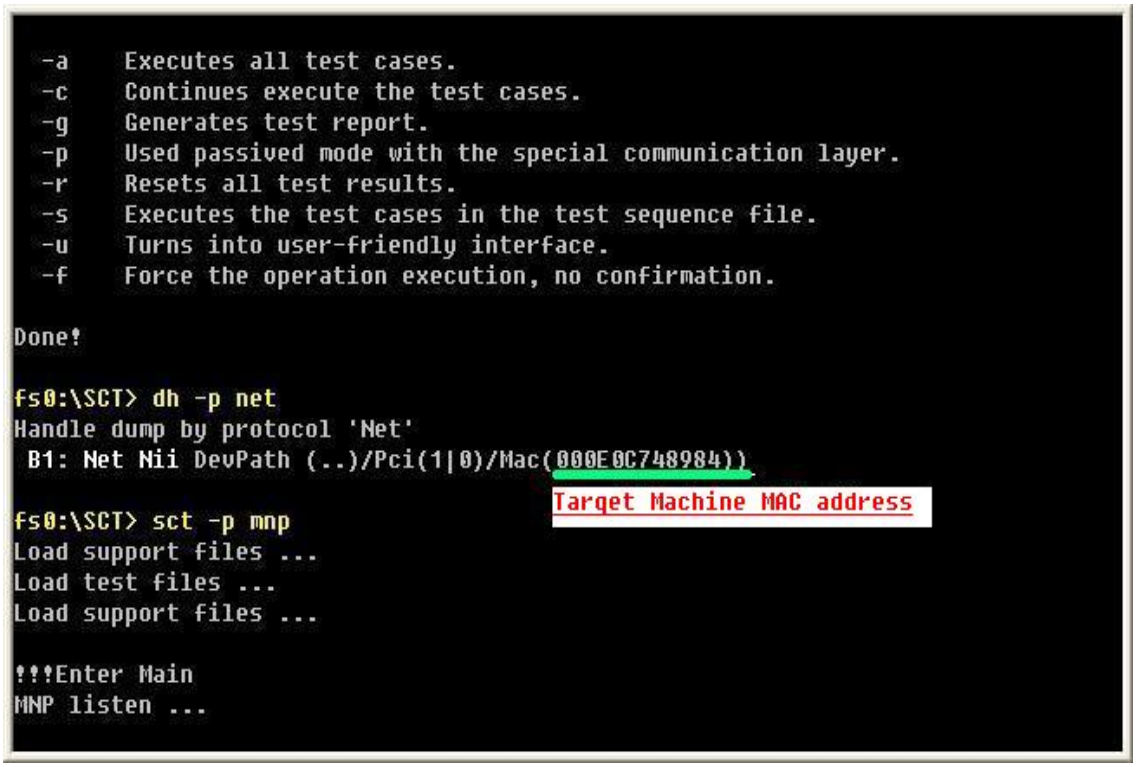


Figure 23. EMS Preference window.

4. Old Target MAC

   This item refers to the current configured target host MAC address.

5. Target Platform Bits

   This item refers to the target platform with which the EMS connects. You can choose 32bits or 64bits. Choose 32bits for the IA32 platform and 64bits for others.

After configuring, click “OK” to confirm the configurations, or click “Cancel” to abort. Clicking “OK” saves the configurations as the default settings.

3.2.3 RemoteExecution & RemoteValidation

There are two methods to validate the EFI-based machine in UEFI SCT passive mode. One is Remote Execution, and the other is Remote Validation.

- All Remote Execution test case files are located on the UEFI SCT Agent side.
  All cases are executed on the EFI side. The EMS performs case management tasks.
- All Remote Validation test case files are in Tcl scripts stored on the EMS side.
  All Remote Validation test cases use Remote Procedure Call (RPC) to perform the validation.

When the user selects the menu Windows-> RemoteExecution, the EMS side will download the CaseTree information file from the target host and generate the remote case tree by parsing the file.

When the user selects the menu Windows-> RemoteValidation, the EMS side will traverse all subdirectories under the test case root directory and generate the local case tree.

3.2.4 Reflushing the Case Tree

The case tree can change after the EMS application starts. So you must reflush the case tree when it changes. Select the menu Windows->Reflush Case Tree to regenerate the case tree. Figure 24 shows the menu in the EMS OS application window.

Note: *When reflushing case tree, the case tree GUI will be re-generated so current case selection and running result will be cleaned up on GUI.*

For Remote Execution, the EMS side will download the file CaseTree.ini from the target host again and then regenerate a remote case tree by reading the file.

For Remote Validation, the EMS side will traverse all the subdirectories of the test case root directory again and then regenerate a local case tree. (Refer to section 3.2.2.)

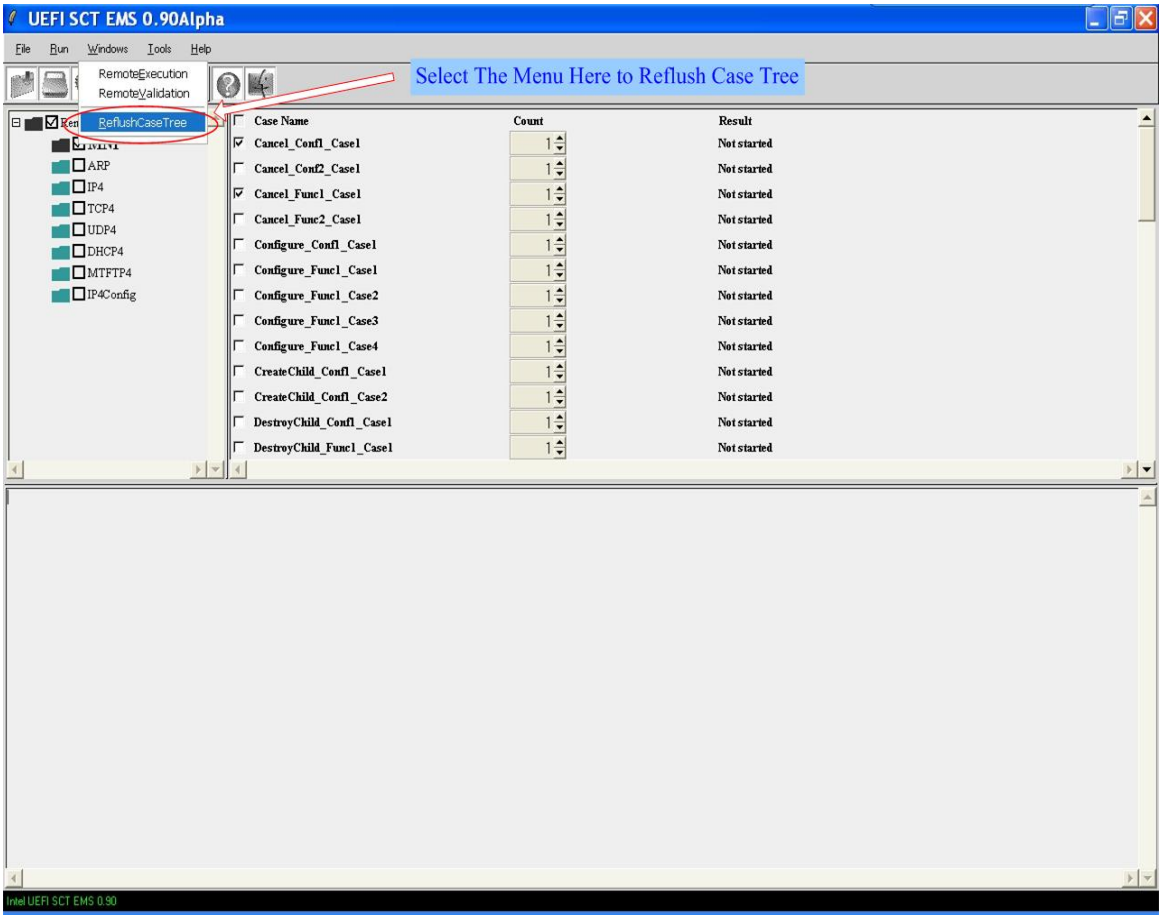


Figure 24. The Menu of Reflush Case Tree.

3.2.5 Running Test Cases

When the EMS configuration is complete and the UEFI SCT Agent is running in the passive mode, run the test cases. Figure 25 shows the EMS OS application window running the Remote Validation test cases.

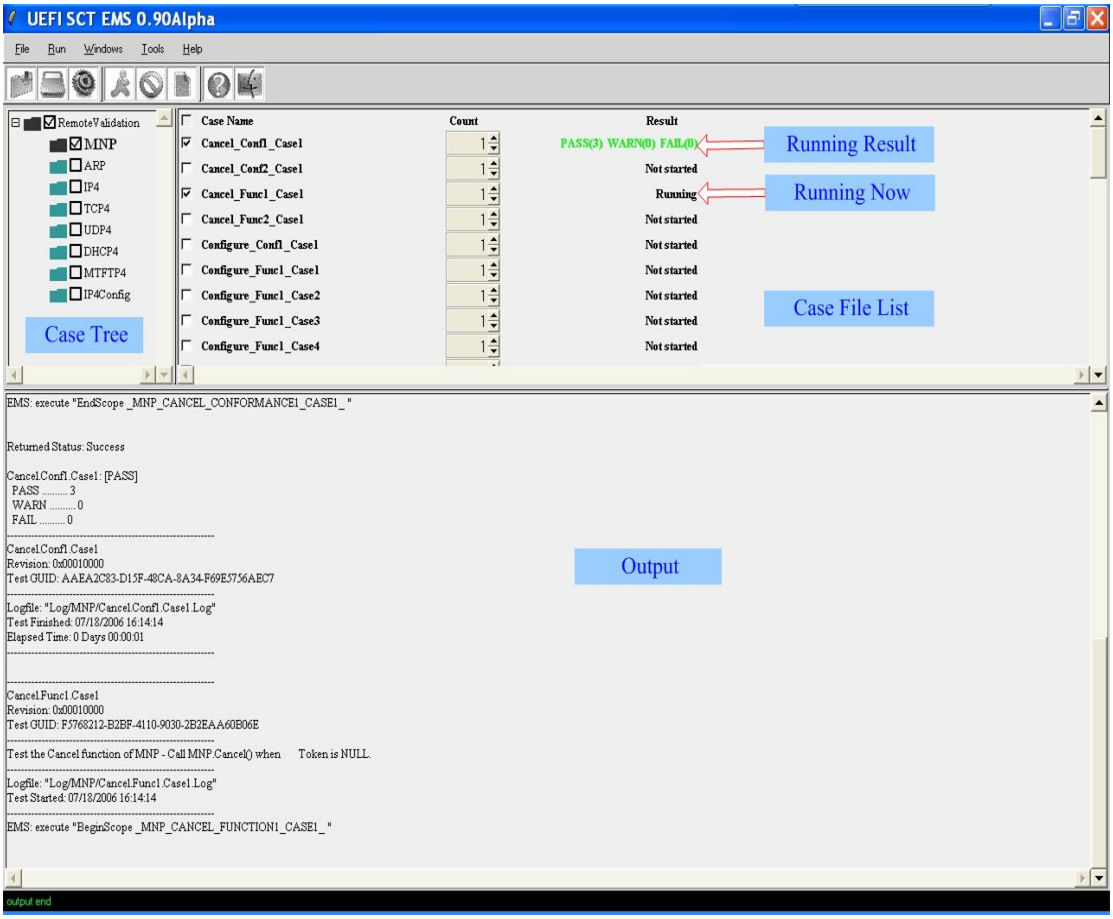


Figure 25. EMS OS application window running the Remote Validation test cases.

Table 4 describes the sub-frames in the EMS OS application window.

Table 4. Sub-Frame in the EMS OS application window

Items	Description
Case Tree	Both the RemoteExecution and the RemoteValidation case tree will be generated in this sub-frame. Select the menu Windows-> RemoteExecution or Windows-> RemoteValidation to switch the case tree.

Case File List	Lists all the case files in the selected case tree directory. Each case file has 3 elements: • Case Name: Case name. • Count: Running iteration of the corresponding test case. • Result: The result of running the selected test case. If the test case is not selected, it will show “Not started”. If the test case is still running, it will show “Running”. If error occurs, it will show “Case Error”. If the test case has been run, it will show the record assertion number of passes, warnings, and failures as shown in Figure 26.
Output	Shows the running log for the test cases. There are two kinds of log files: [Case Name].log and [Case Name].ekl. The log files are generated under the directory \bin\log\[Case Directory Name].

In the Case Tree sub-frame, each case directory has 3 elements: an icon, a check box, and a directory name text. Figure 26 shows each element and the status of each element.

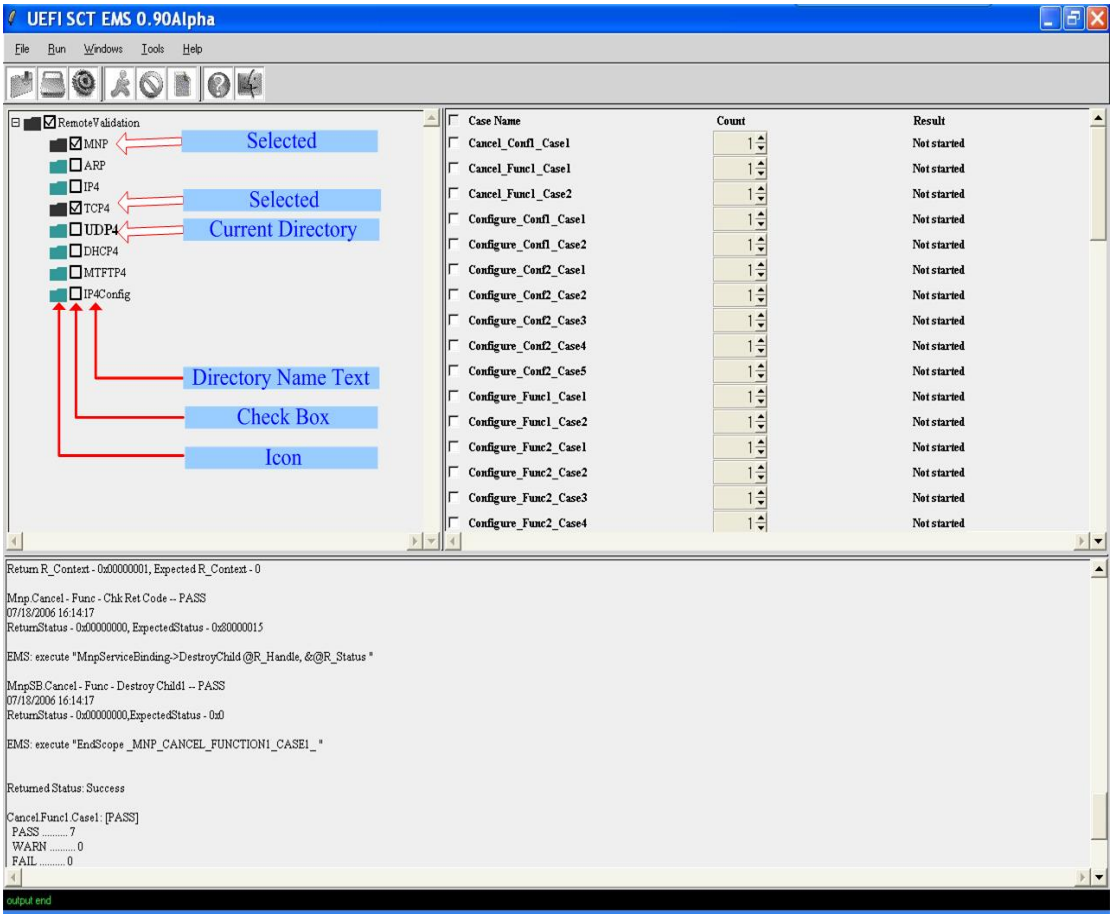


Figure 26. EMS OS application window-Case Tree Sub-frame.

Table 5 describes the usage and the different status meanings of each element for the Case Tree directory.

Table 5. Each Element in the Case Tree Sub-frame

Items	Description
Icon	You can click the Icon of a case directory to change the current directory. Status meanings: • Green Color: no case file was selected. • Black Color: one or more case files were selected.
Check Box	You can click the Check Box to select all the case files in the case directory. Status meanings: • Unchecked: no case file was selected. • Checked: one or more case files were selected.
Directory Name Text	Status meanings • Bold: Current case directory. • Regular: Not current case directory.

After selecting the cases to run, select the menu Run->Start to run the test cases. Status meanings

To stop the case when running, click the menu Run->Stop, and the test will stop after the current running test case has finished. This is to make sure the case running context is clean and that test cases won't affect each other.

3.2.6 Loading and Saving a Sequence File

Selected test cases can be saved as a sequence file. Sometimes it is more convenient to run some test cases more than one time, and this function allows one selection, rather than reselecting all the test cases again. Select all the test cases the first time, save the selection as a sequence file, and when running those test cases again, one can load the sequence file to select test cases automatically. The test sequence file is created in the same directory where the SCT was invoked.

To save a sequence file, select one or more test cases, and then select the menu "File->Save sequence file as…" Figure 27 shows the sequence file Save As window.

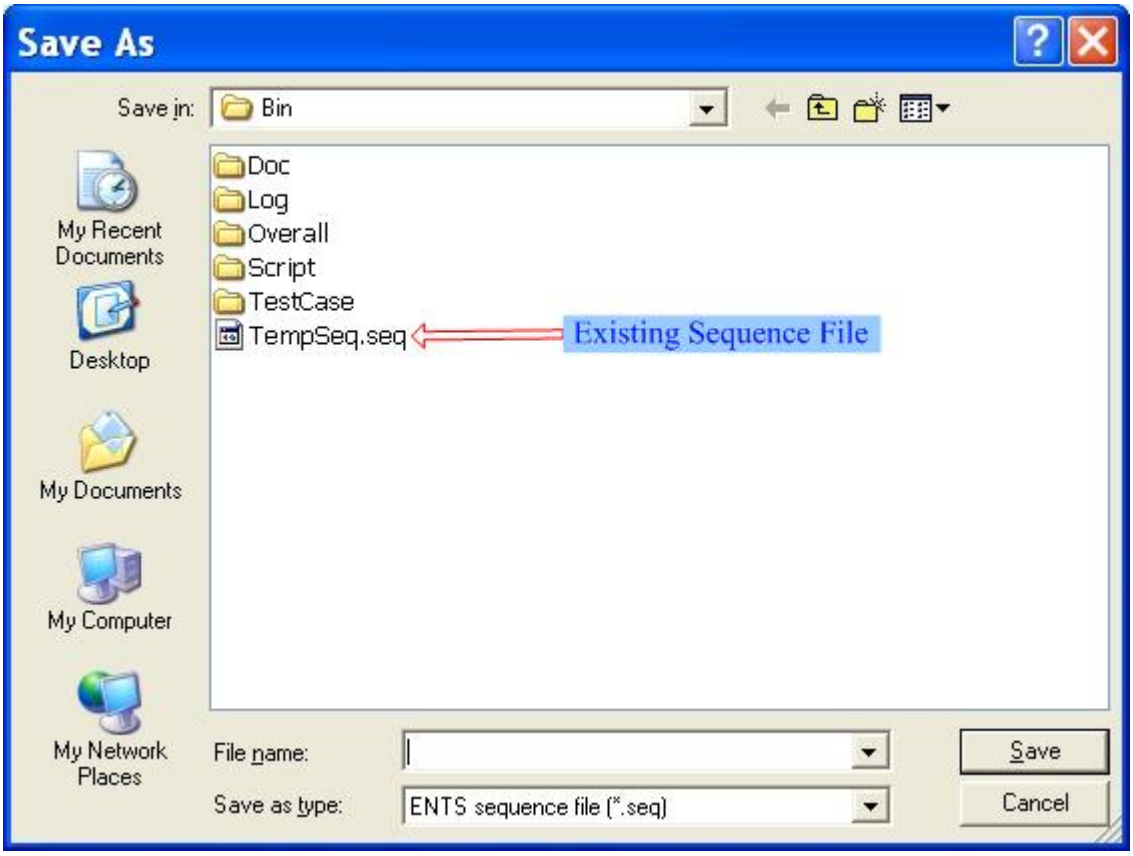


Figure 27. Sequence File Saving Window.

To load a sequence file, select the test case, then select the menu "File->Load sequence file". Figure 28 shows the sequence file loading window.

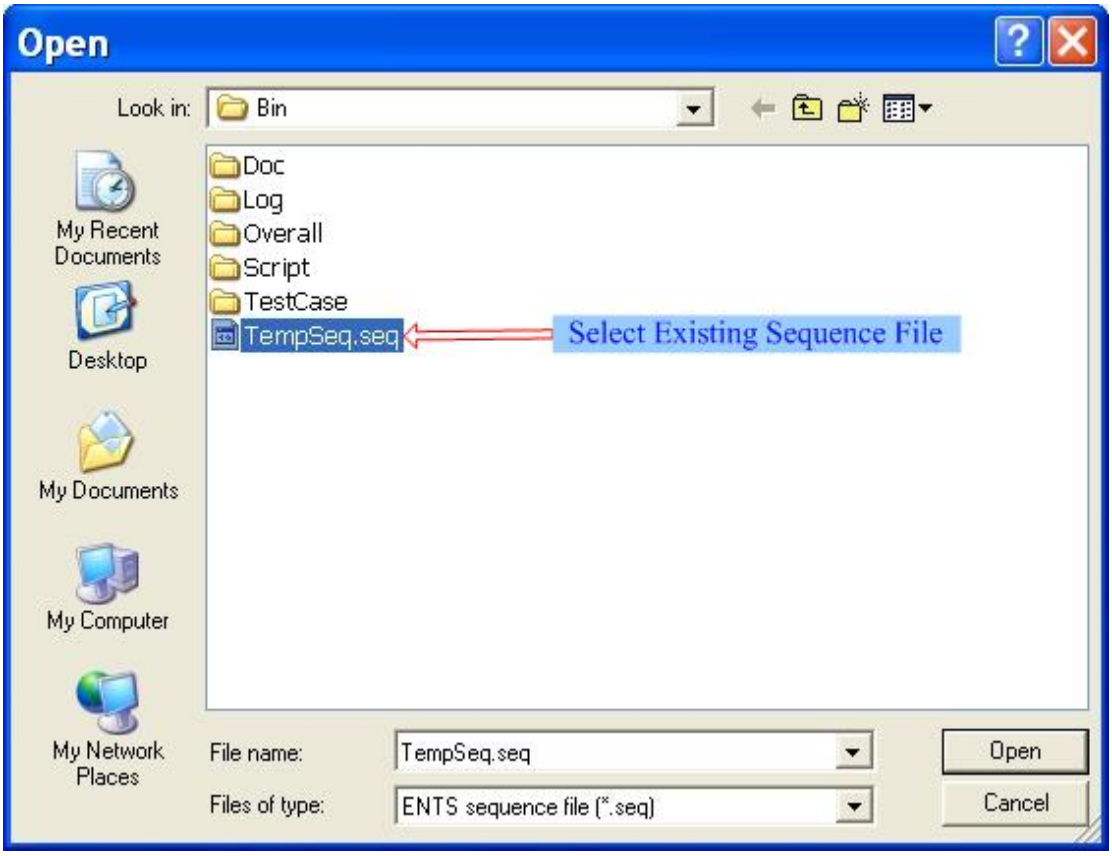


Figure 28. Sequence File Loading Window.

3.2.7 Generating Log Files

For Remote Validation, the test report file is created in the "Report" subdirectory where the EMS was invoked. Two kinds of reports are generated: one is in case-level and the other is in assertion-level.

For Remote Execution, the test report is created remotely on the EFI target machine and the test report file is transferred back to the report subdirectory where the EMS was invoked.

Note: *The report file is in CSV format and the report file is named by date and time.*

3.2.8 Using the Tools Menu

Table 6 describes each submenu function of the Tools menu.

Table 6. Submenus of the Tools Menu

Items	Description
Edit	Opens an editing window. This is a simple text editor and it provides highlighting display for UEFI SCT remote validation test cases. Figure 29 shows the functions in detail.
Clear Output	Clears current records in the Output sub-frame of the EMS OS application window.

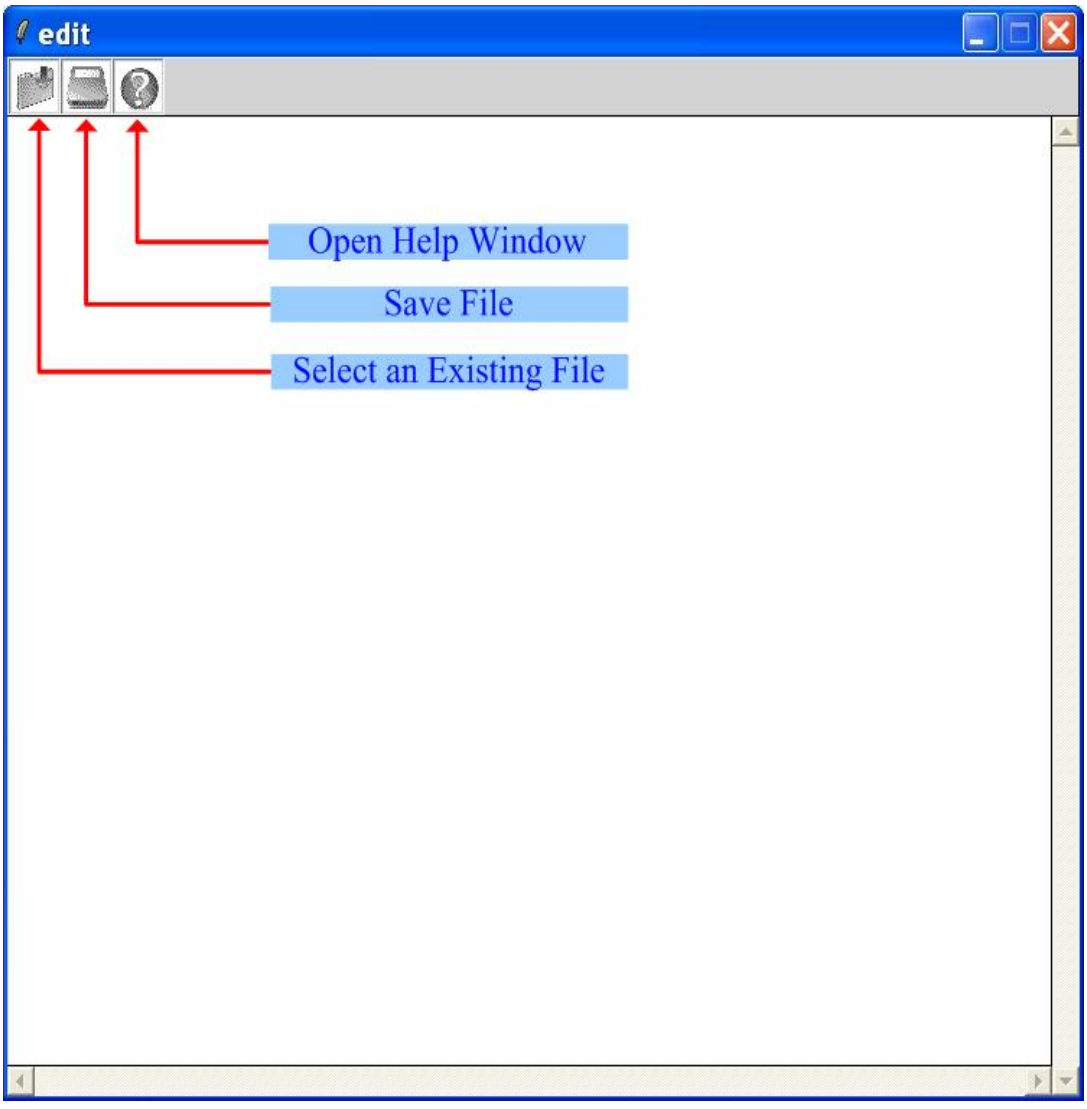


Figure 29. Editing File Window.

3.2.9 Using the Help Menu

Table 7 describes each submenu function of the Help menu.

Table 7. Submenus of the Help Menu

Items	Description
Index	Provides a quick reference on Remote Validation Tcl commands for case developers. Find detailed usage information about the commands used in the Tcl script. Figure 30 shows the functions in detail.
About ENTS…	Provides the version and copyright information about EMS.

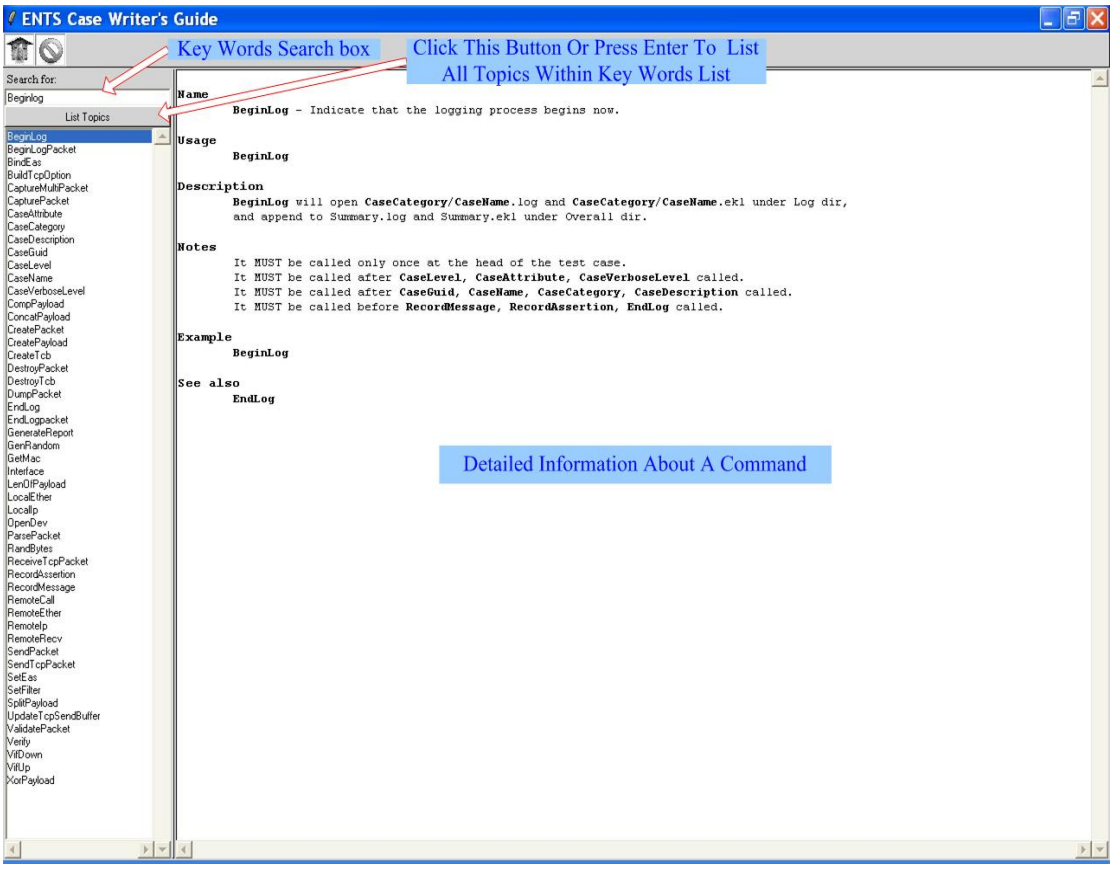


Figure 30. ENTS Case Writer's Guide Window.

4 UEFI SCT For IHV

4.1 IHV SCT Building and Installation

4.1.1 Building the IHV SCT

4.1.1.1 Setup Development System

Several Microsoft tools are required to build the EDK and the UEFI SCT source tree. The following tools must be installed on the development system.

- Microsoft Windows 2000® or Microsoft Windows XP® operating system

For 32-bit Intel architecture (IA32) platform development:

- Microsoft Visual Studio .NET® 2003 Professional (7.1)

For Intel Itanium® processor family development:

- Microsoft® C/C++ Optimizing Compiler Version 13.10.2240.8 for IA64 in the Microsoft Windows Server® 2003 DDK (Build 3790).

  When compiling for targets based on Itanium architecture, use NMAKE.EXE, LINK.EXE and LIB.EXE from the Microsoft Windows Server 2003 DDK (Build 3790).

4.1.1.2 Download Source Code

The first step is to download the source code from www.uefi.org. You must register and use the log in information you receive. Everyone is welcome to join. To build the IHV SCT, downloaded the EDK and the EFI Shell source code as well .

The IHV SCT this document referred to is compliant to UEFI Specification, so the source code of UEFI SCT and IHV SCT are in one package. Similar to UEFI SCT, building, an IHV SCT also needs the source code packages of SCT, EDK and EFI Shell. The details about where and how to get these packages are described in "*UEFI SCT User Guide*".

The packages download from website are Zip files. Users should extract these packages into different locations step by step:

- Extract the Zip file of SCT source code package to a location on your system, for example: **`C:\Test.`**
- Extract the Zip file of EDK to the SCT directory, for example, extract to **`C:\Test\UefiSct.`**

- Extract the Zip file of EFI Shell to the EDK\Other\Maintained\Application directory, for example, **C:\Test\UefiSct\Edk\Other\Maintained\Application.**

4.1.1.3 Build Configuration Files

There is only one configuration file primarily used to describe the local system for building the various build tips, its name is PlatformTools.env. Using this file, users can define or modify some environmental variables for building.

4.1.1.4 IA32 Build Tip

Run Visual Studio .NET 2003 Command Prompt to go to the command line environment. The following commands can be used to build the UEFI SCT IA32 tip. If the build is successful, an install package SctPackage will be created in the Platform\IntelTest\UEFI\IA32\ihv directory.

1. cd \test\efisct\platform\inteltest\UEFI\ia32\
2. set efi_source=c:\test\efisct
3. nmake ihv

4.1.1.5 X64 Build Tip

To build the UEFI SCT X64 tip, you just need to change the directory from IA32 to X64. an install package SctPackage will be created in the Platform\IntelTest\UEFI\X64\ihv directory.

1. cd \test\efisct\platform\inteltest\UEFI\x64\
2. set efi_source=c:\test\efisct
3. nmake ihv

4.1.1.6 IPF Build Tip

To build the UEFI SCT IPF tip, you just need to change the directory from IA32 to IPF. an install package SctPackage will be created in the Platform\IntelTest\UEFI\IPF\ihv directory.

1. cd \test\efisct\platform\inteltest\UEFI\ipf\
2. set efi_source=c:\test\efisct
3. nmake ihv

4.1.2 Installing the IHV SCT

The IHV SCT agent is a shell application, so the EFI Shell environment is a must to run IHV SCT agent. in case you don't have built-in shell for UEFI 2.1 sample code or a Tiano implementation, setup the shell environment by following the steps given below:

1. Copy the shell.efi to the target machine.

2. Add a boot option to the shell.efi just added.

3. Boot to the specified shell environment, and do the following installation steps, according to different target platforms.

The UEFI SCT Agent can be installed on the following platforms:

- IA32 Platform
- Itanium-Based Platform
- EM64T-Based Platform

4.1.2.1 Installing the IHV SCT Agent on an IA32 Platform

1. Copy the contents of the IA32 build directory SctPackage to a USB device or IDE-CD.

2. Put the USB or IDE-CD into the USB port or the IDE-CD drive and boot the system to the EFI Shell environment.

3. In EFI Shell environment, change the current drive and directory to the installation CD or USB device drive and root directory.

4. Run installIA32.efi and follow the instructions on the screen.

4.1.2.2 Installing the IHV SCT Agent on an Itanium-Based Platform

1. Copy the contents of the x64 build directory SctPackage to a USB device or IDE-CD.

2. Put the USB or IDE-CD into the USB port or the IDE-CD drive and boot the system to the EFI Shell environment.

3. In EFI Shell environment, change the current drive and directory to the installation CD or USB device drive and root directory.

4. Run install64.efi and follow the instructions on the screen.

4.1.2.3 Installing the IHV SCT Agent on an EM64T-Based Platform

1. Copy the contents of the IPF build directory SctPackage to a USB device or IDE-CD.

2. Put the USB or IDE-CD into the USB port or the IDE-CD drive and boot the system to the EFI Shell environment.

3. In EFI Shell environment, change the current drive and directory to the installation CD or USB device drive and root directory.

4. Run installX64.efi and follow the instructions on the screen.

4.2 The Usage of IHV SCT

4.2.1 Using the Command Line Interface

The command line interface of the IHV SCT agent is similar to the UEFI SCT's (see the "UEFI SCT User Guide"), but the IHV SCT does not support the passive mode. The syntax of the IHV SCT's command line is:

```
SCT [-a | -c | -s <seq> | -u][-r] [-g <report>]
```

Table 8 provides a description of SCT parameters.

Table 8. SCT Parameters

Options	Description
-a	Execute all test cases that are recognized by the IHV SCT Test Harness.
-c	Continue execution of the test case in progress. This option is used to continue execution of test cases that perform system resets as part of their test routine.
-g <report>	Generate test report in .CSV format. The filename of the report is specified by **report.**
-r	Resets the environment for a fresh execution of the tests. This option removes results of previous test executions. Generally, it is used with the **-a** or **-s** options.
-s <seq>	Execute test cases in the sequence specified in the file **seq**.
-u	Start the Test Harness with the menu-driven interface.

4.2.2 Using the Menu-Driven Interface

Syntax

```
SCT -u
```

Description

Type SCT -u to produce the Main Menu of the menu-driven interface.

4.2.2.1 Main Menu

The Main Menu (see Figure 31) contains user selectable items for initiating a number of IHV SCT actions.

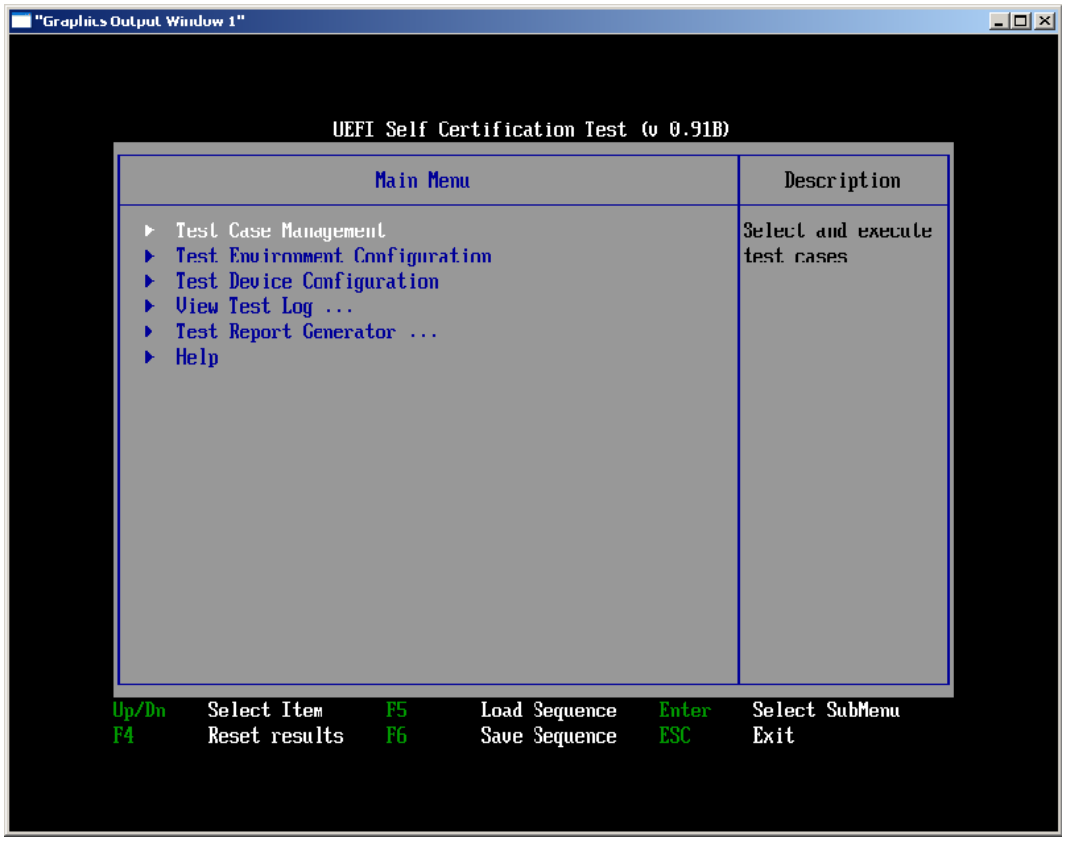


Figure 31Main Menu of IHV SCT

Table 9 lists and describes the major items found in the Main Menu.

Table 9. Major Items in the Main Menu of the SCT

Items	Description
Test Case Management	Selects and executes specific test cases
Test Environment Configuration	Sets the parameters for test execution, including the maximum run times for each test case, enabling/disabling screen output, etc.
Test Device Configuration	Selects the devices that should be tested.
Test Report Generator	Generates a test report in .CSV format. This test report can be opened by the Microsoft® Excel* or the other compatible utilities.
F4 (Reset Results)	Resets all test results. It is equivalent to invoking "**SCT -r**" in the command line.
F5 (Load Sequence)	Loads a test sequence file from the storage device. This function allows user to load, edit or execute an existing test sequence file.

F6 (Save Sequence)	Saves a user-specified test sequence into a file. This function allows the user to save selected test cases into a file that can then be used for later test execution via "**SCT -s <seq>**" from the command line.

4.2.2.2 Managing Test Cases

The UEFI compliance IHV SCT includes a set of test cases for UEFI 2.1 Specification compliance testing. The method to manage the test cases and to specify test cases in the tree-like hierarchy is described in "UEFI SCT User Guide".

In IHV SCT, only selecting the test cases in the tree-like menu is not enough to test. Selecting cases in tree-like menu just tells IHV SCT which cases should be run, in IHV SCT, users must choose which devices they want to test through "Test Device Configuration"(see section 4.2.2.3).

4.2.2.3 Test Device Configuration

The IHV SCT provides the utility of Test Device Configuration. This allows users to choose the devices for testing. The IHV SCT uses a configuration file to save a list of devices that users have chosen. During IHV SCT testing, it will only test the supported devices listed in the configuration file instead of all the supported devices in the system. In other words if the users only select the test cases through the tree-like menu but do not choose any device though the "Test Device configuration", no checkpoints in the cases will be tested.

In the IHV SCT, selecting cases in tree-like menu tells IHV SCT which cases should be run; and choosing devices through the "Test Device Configuration" tells the IHV SCT which devices should be test. The usage of Test Device Configuration is shown in Figure 32:

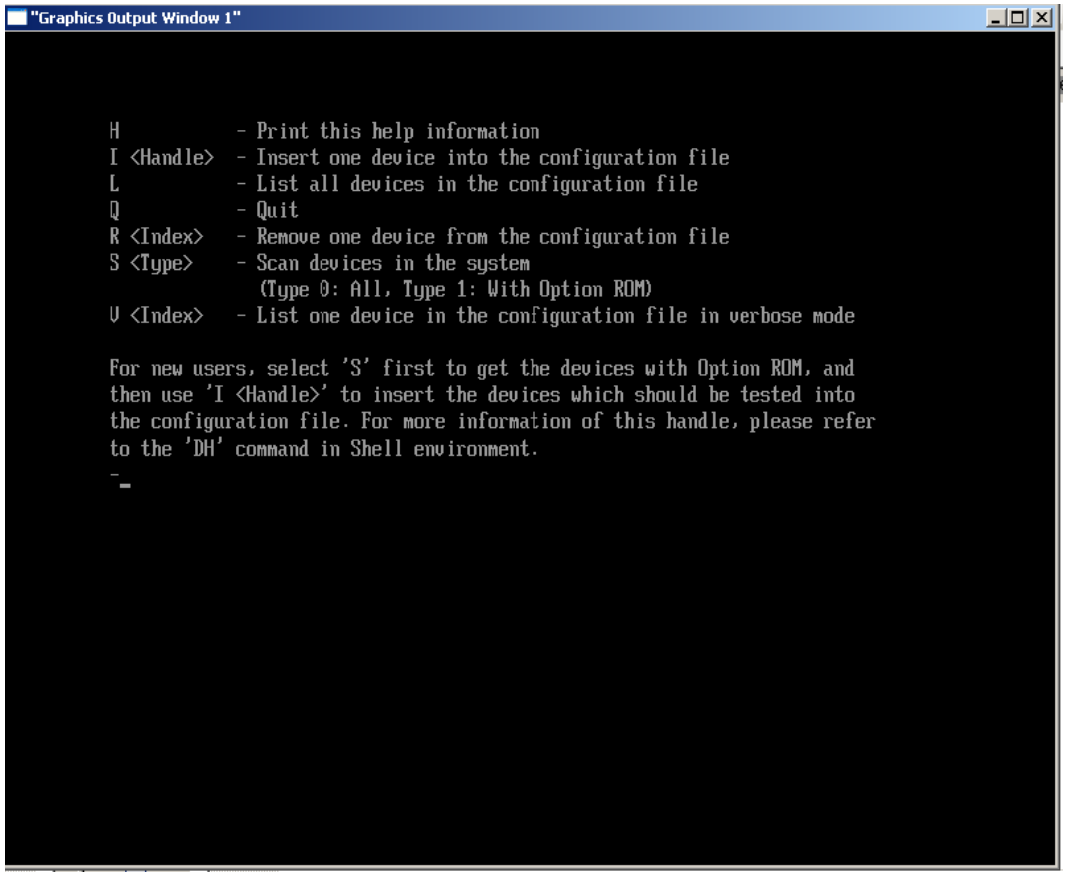
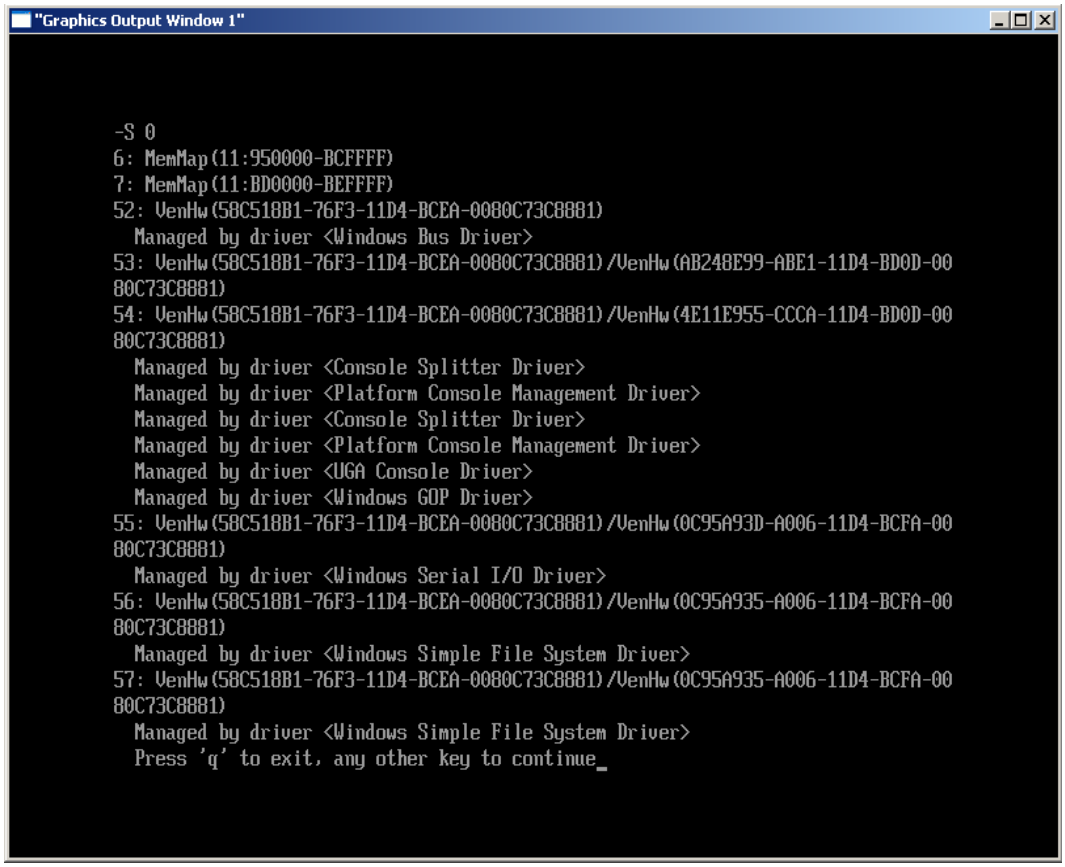


Figure 32 Test Device Configuration

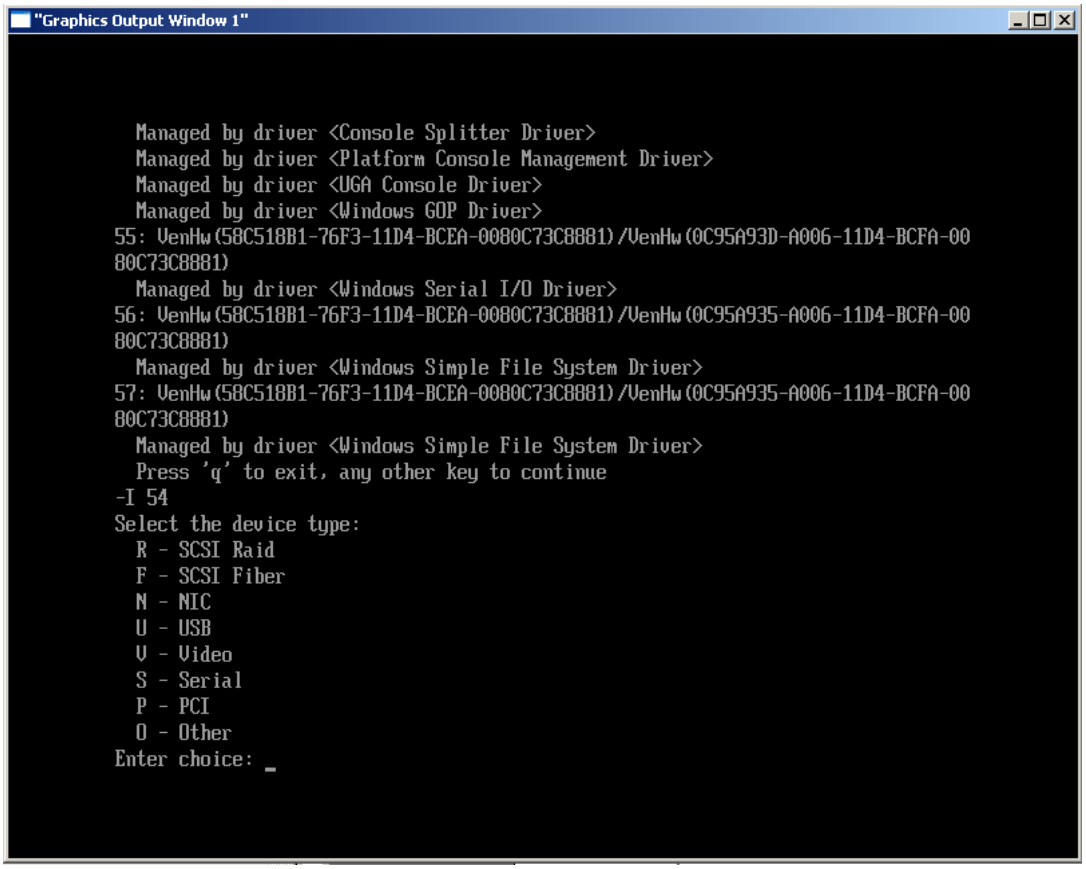
Table 10. The Items in the Menu of the Test Device Configuration

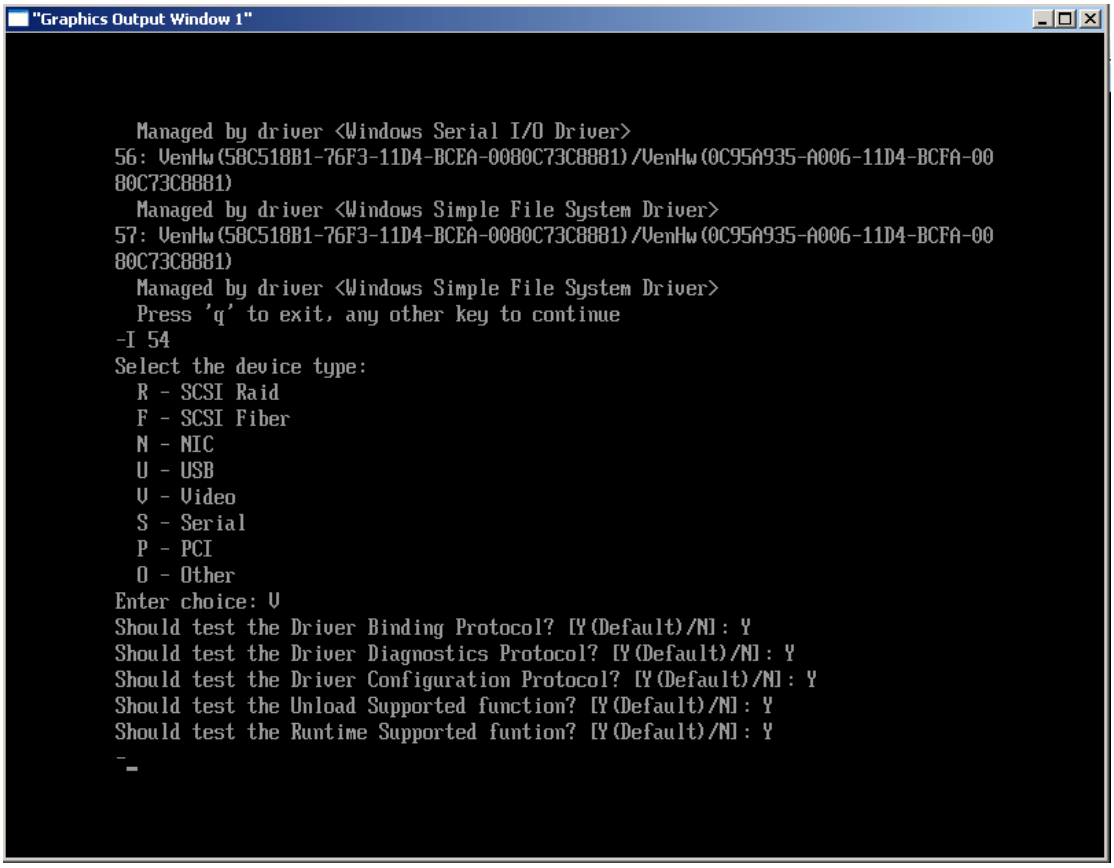
Items	Description
H	Print the help information
I <Handle>	Insert one device into the configuration file
L	List all devices in the configuration file
R <Index>	Remove one device from the configuration file
S <Type>	Scan devices in the system (Type 0: All, Type 1: With Option ROM)
V <Index>	List one device in the configuration file in verbose mode

In the IHV SCT, the usual way to test an add-in card as follows: The first thing users should do is let the SCT scan devices in the system by typing the command line “S 0” or “S 1” in the Test Device Configuration’s window. “S 0” means scan all devices, “S 1” means scan devices with option ROM. See table 10.



After SCT scan, the “Handle” of the device sought can be known, so users can insert the device sought into SCT’s configuration file by typing the command line ”I <Handle>”;


"Graphics Output Window 1"
Managed by driver <Console Splitter Driver>
Managed by driver <Platform Console Management Driver>
Managed by driver <UGA Console Driver>
Managed by driver <Windows GOP Driver>
55: VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A93D-A006-11D4-BCFA-00
80C73C8881)
Managed by driver <Windows Serial I/O Driver>
56: VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A935-A006-11D4-BCFA-00
80C73C8881)
Managed by driver <Windows Simple File System Driver>
57: VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A935-A006-11D4-BCFA-00
80C73C8881)
Managed by driver <Windows Simple File System Driver>
Press 'q' to exit, any other key to continue
-I 54
Select the device type:
R - SCSI Raid
F - SCSI Fiber
N - NIC
U - USB
V - Video
S - Serial
P - PCI
O - Other
Enter choice:



At this time, users can select test cases in the tree-like menu. The SCT can run the test only if all the operations of test device configuration are done.

Note: *If users want to start a new test because the test device configuration has been changed, the "SCT -r" operation is suggested.*

5
UEFI SCRT

5.1 Introduction

This chapter introduces the Self-Certification Runtime Test (SCRT) Utility and focuses on how to use it.

As a supplement to SCT, SCRT is invoked under the EFI shell environment and used to validate UEFI Runtime Services implementations for compliance to the *UEFI 2.1 Specification*. The source code of SCRT has been included in UEFI SCT release package and the binary of SCRT utility is generated automatically in the build process of UEFI SCT. Please refer to the instructions in the document *UEFI SCT Getting Started*. This document is included in the UEFI SCT release package to build the UEFI SCT Agent.

5.2 The Usage of SCRT

5.2.1 System Requirement

To ensure SCRT runs in the runtime environment without unexpected behavior, for targeted platforms the physical memory on the target machine is limited to the following rules:

- IA32 architecture-based platform: Physical memory <= 4G.
- EM64T architecture-based platform: Physical memory <= 32G.
- IPF architecture-based platform: Physical memory <= 4096T

5.2.2 The location of SCRT Utility

After UEFI SCT is built successfully, SCRT Utility is generated automatically and located at specified path below, including **SCRTDRIVER.efi**, **SCRTAPP.efi**, **SCRT.conf**.

UefiSct\Platform\IntelTest\UEFI\IA32\uefi21\SctPackage\IA32\SCRT IA32 Version
UefiSct\Platform\IntelTest\UEFI\X64\uefi21\SctPackage\X64\SCRT X64 Version
UefiSct\Platform\IntelTest\UEFI\IPF\uefi21\SctPackage\IPF\SCRT IPF Version

5.2.3 Run SCRT Utility

SCRT is invoked under the EFI shell environment:

1. Copy SCRT Utility into discretionary directory in EFI shell environment.

2. Change execution path to the directory that SCRT Utility is located.
3. type ‘Load SCRTDRIVER.efi’
4. type ‘SCRTAPP –f SCRT.conf’

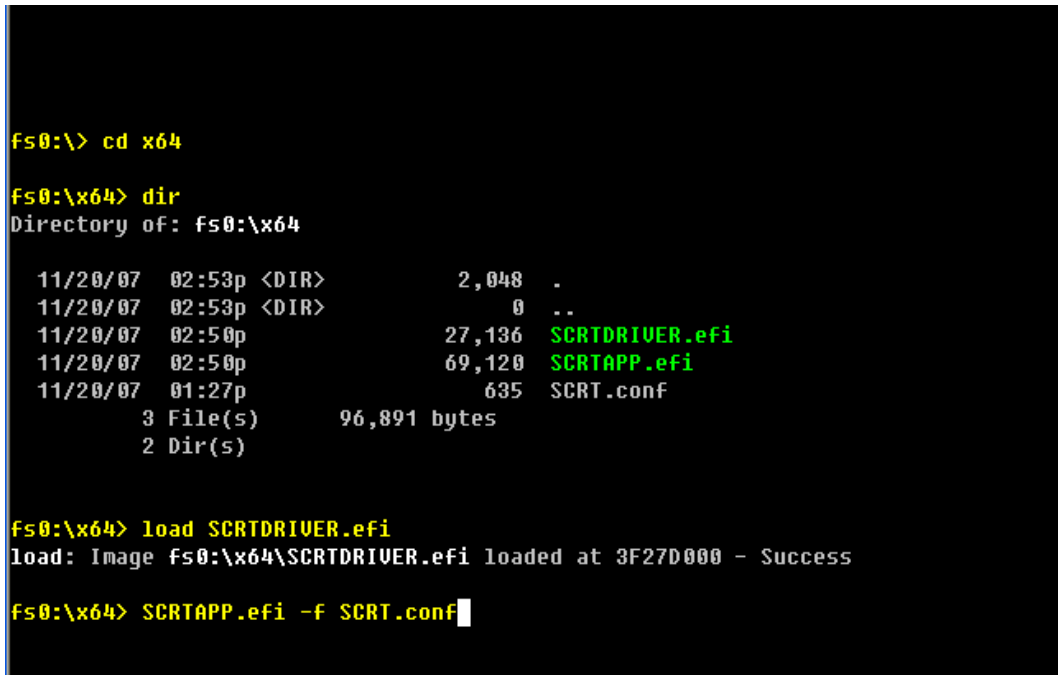


Figure 33 Run SCRT Utility with configure file

5.2.4 Configuration File

Following is an example for the usage model of the configuration file named **SCRT.conf**. SCRT check points are divided into five groups, Variable Service, Time Service, Capsule Service, MonotonicCount Service, and Reset Service.

In **SCRT.conf**, **FALSE** means to disable a runtime service test, and **TRUE** means to enable a runtime service test.

With the help of this configuration file, SCRT obtains information regarding which runtime services are needed to test in the runtime environment.

```
#
# UEFI 2.1 Runtime Test Utility SCRT Configuration file.
#
[variable]
SetVariable                 = TRUE
GetVariable                 = TRUE
GetNextVariableName         = TRUE
QueryVariableInfo           = FALSE

[time]
GetTime                     = TRUE
SetTime                     = TRUE
SetWakeupTime               = TRUE
GetWakeupTime               = TRUE

[capsule]
QueryCapsuleCapabilities    = FALSE
UpdateCapsule               = FALSE

[monotonicCount]
GetNextHighMonotonicCount   = TRUE

[reset]
ColdReset                   = TRUE
WarmReset                   = FALSE
ShutDown                    = FALSE
```

Note: *For three **reset** sub-items, only one item is allowed at a time.*

5.2.5 Analyze SCRT Test Result

Unlike SCT, SCRT cannot create a test log file automatically in a runtime environment because it lacks certain boot services. To solve this issue, SCRT records the results in a variable. After runtime test, user can run "**`SCRTAPP.efi –g SCRT.log`**" in shell environment to analyze the variable and generate a log file which is named as '**`SCRT.log`**'. It lists all requested test points and separate test results. From these messages, users can easily find which test point fails.

Besides this method, SCRT can send debug messages to Port 80 at the execution time. Using these messages, the user can analyze the failure reason.

5.2.5.1 Log File Overview

SCRT log file is divided into several groups:

```
Variable Services Test
Time Services Test
Capsule Service Test
Misc Services Test
Reset Services Test
```

The following is an example of the log file:

Note: *Sometimes the result of Reset Services Test is not correct. Please note the platform behavior to judge*

```
********************Variable Test Group********************

SetVariable             Requested
SetVariable             Pass
GetVariable             Requested
GetVariable             Pass
GetNextVariable         Requested
GetNextVariable         Pass

********************Time Test Group********************

GetTime                 Requested
GetTime                 Pass
SetTime                 Requested
SetTime                 Pass
SetWakeupTime           Requested
SetWakeupTime           Pass
GetWakeupTime           Requested
GetWakeupTime           Fail

********************Capsule Test Group********************


********************Misc Test Group********************

GetNextCount            Requested
GetNextCount            Not Test

********************Reset Test Group********************

ColdReset               Requested
ColdReset               Not Test
```

Please note the following”

- **Requested** means this test point is requested to test in runtime environment.
- **Pass** means this test point is tested successfully in runtime environment.
- **Fail** means this test point is failed during runtime test, usually it causes system hang.
- **Not Test** means this test point is not tested because some test point prior to it causes system hang.

5.2.5.2 Port 80 Display

If the target machine under test has Port 80, the hex number displayed with Port80 can be used to trace the test case workflow. For every checkpoint, Port 80 will display a unique hex number. Please refer to Appendix C for more details.

5.2.6 System Hang

SCRT validates the Runtime Services implementation in the runtime environment. If some pointers are not converted, the system hangs. If the system hangs at Nth checkpoint, the SCRT records the (N-1)th information in the test log file and displays the corresponding hex number in Port 80. Using this relationship with the enabled checkpoint sequences, users can find which checkpoint hangs.

5.3 How to Add SCRT Test Cases

SCRT is used to validate Runtime Services in a runtime environment. If a more detailed test case for runtime services is needed, users may develop the required test case, and add it to the SCRT infrastructure.

5.3.1 The Framework of SCRT Utility

SCRTDriver in the SCRT utility is responsible for performing the test cases. In **SCRTDriver** module, GUID definition for the checkpoints is declared in **Guid.h** and **Guid.c**, and test cases are located in **TestCase.c**.

To extend the test coverage, the user can add the test cases in **TestCase.c** and add the new GUID definitions **in Guid.h**/**Guid.c**.

```
SCRTDriver\
    |----Guid.h
    |----Guid.c
    |----TestCase.c
    |----Debug.c
    |----Print.c
    |----SCRTDriver.c
    |----SCRTDriver.h
    |----SCRTDriver.inf
    |----ia32
        |----Dump.c
        |----Io.c
        |----Io.h
        |----IoAccess.asm
        |----Port80.asm
    |----ipf
        |----Dump.c
        |----Io.c
        |----Io.h
        |----Port80.c
    |----x64
        |----Dump.c
        |----Io.c
        |----Io.h
        |----IoAccess.asm
        |----Port80.asm
```

Note: ***Guid.h/Guid.c*** *declares GUID definition.*

Note: ***TestCase.c*** *consists of the test cases.*

In **TestCase.c**, we allow for adding more checkpoints. For each new checkpoint, the user needs to create a new GUID for it and declare it in **Guid.h**/**Guid.c**.

5.3.2 Example: Adding a Test Case

Because the call Runtime Service **UpdateCapsule** behaves differently for different platforms—for example, a system reset—this checkpoint is not included in **TestCase.c** as a common test case. Users can add a case in **TestCase.c** to verify the service, per the example shown below.

Following is sample code to add the checkpoint in **EfiCapsuleTestVirtual(), TestCase.c:**

```
Port80(xxx);

Status = VRT->UpdateCapsule (
                xxxxx,
                xxxxx,
                xxxxx
                 );

RecordAssertion (
    Status,
    gSCRTAssertionGuidxxx,
    "RT. UpdateCapsule - should be EFI_SUCCESS",
    "%a:%d:Status - %r, Expected - %r",
    __FILE__,
    __LINE__,
    Status,
    EFI_SUCCESS
    );
```

In addition, define **gSCRTAssertionGuidxxx** in **Guide.h** and **Guide.c** as shown below:

In **Guide.c**:

```
EFI_GUID gSCRTAssertionGuidxxx = EFI_TEST_SCRT_ASSERTION_xxx_GUID;
```

In **Guide.h**:

```
#define EFI_TEST_SCRT_ASSERTION_xxx_GUID \
{ xxxxxxxx, xxxx, xxxx, { xx, xx, xx, xx, xx, xx, xx, xx } }

extern EFI_GUID gSCRTAssertionGuidxxx;
```

Appendix A
Test Report Format

A summary of SCT test results is recorded into a test report file in CSV format. The output information includes the number of passed and failed test assertions for each executed test category, as well as detailed information for each executed test assertion, passed or failed.

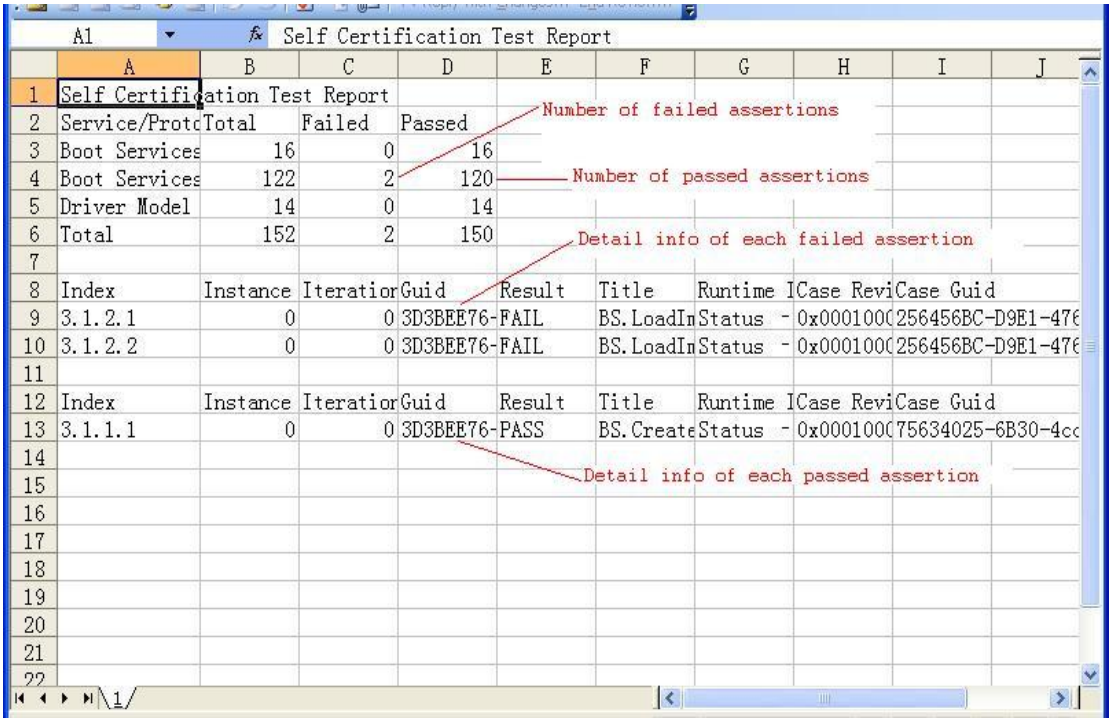
Below are the contents of a sample test report file:

```
"Self Certification Test Report"
"Service/Protocol Name","Total","Failed","Passed"
"Boot Services Test\Event, Timer, and Task Priority Services
Test","16","0","16"
"Boot Services Test\Image Services Test","121","1","120"
"Driver Model Test\Driver Binding Protocol Test","15","1","14"
"Total","152","2","150"

"Index","Instance","Iteration","Guid","Result","Title","Runtime
Information","Case Revision","Case Guid"
"3.1.2.1","0","0","3D3BEE76-3BE8-40DD-BD34-
C38AFE2BBDEB","FAIL","BS.LoadImage() - Load image fail via LOAD_FILE
protocol","Status - Unsupported, TPL - 4","0x00010000","256456BC-D9E1-
476c-B4AD-BE37E53F7940"
"3.1.2.2","0","0","3D3BEE76-3BE8-40DD-BD34-
C38AFE2BBDEC","FAIL","BS.LoadImage() - Load image fail via Device and
File path","Status - Not found, TPL - 8","0x00010000","256456BC-D9E1-
476c-B4AD-BE37E53F7940"

"Index","Instance","Iteration","Guid","Result","Title","Runtime
Information","Case Revision","Case Guid"
"3.1.1.1","0","0","3D3BEE76-3BE8-40DD-BD34-
C38AFE2BBDEA","PASS","BS.CreateEvent() - Create event with invalid event
type","Status - Invalid parameter","0x00010000","75634025-6B30-4cc4-AC5C-
6D031AE4D74C"
```

When viewed in Microsoft Excel ®, the contents of the report file appear as shown in Figure 34.



	A	B	C	D	E	F	G	H	I	J
1	Self Certifi	ation Test Report								
2	Service/Protc	Total	Failed	Passed						
3	Boot Services	16	0	16						
4	Boot Services	122	2	120						
5	Driver Model	14	0	14						
6	Total	152	2	150						
7										
8	Index	Instance	Iteratior	Guid	Result	Title	Runtime I	Case Revi	Case Guid	
9	3.1.2.1	0	0	3D3BEE76-	FAIL	BS.LoadIn	Status -	0x000100(	256456BC-D9E1-476	
10	3.1.2.2	0	0	3D3BEE76-	FAIL	BS.LoadIn	Status -	0x000100(	256456BC-D9E1-476	
11										
12	Index	Instance	Iteratior	Guid	Result	Title	Runtime I	Case Revi	Case Guid	
13	3.1.1.1	0	0	3D3BEE76-	PASS	BS.Create	Status -	0x000100(	75634025-6B30-4cc	
14										
15										
16										
17										
18										
19										
20										
21										
22										

Figure 34. Excel® File Containing Test Report in CSV Format.

Appendix B
Test Category

Information on each test category that the EFI SCT Test Harness will need for execution is provided using a category file in INI format. This file is created in the Data subdirectory.

Below are the contents of a sample category file:

```
 [Category Data]
Revision        = 0x00010000
CategoryGuid    = 7AB1E93F-B439-4e2e-B773-CA540CEBCFEF
InterfaceGuid = E9EF7553-F833-4e56-96E8-38AE679523CC
Name               = Boot Services Test\Event, Timer, and Priority
Services Test
Description     = Event, Timer, and Priority Services Test. Related to EFI
Spec 5.1.

[Category Data]
Revision        = 0x00010000
CategoryGuid   = CC129459-A197-4c8f-9422-2441E88C559A
InterfaceGuid = E9EF7553-F833-4e56-96E8-38AE679523CC
Name                = Boot Services Test\Memory Allocation Services Test
Description     = Memory Allocation Services Test. Related to EFI Spec
5.2.
```

The **CategoryGuid** is the GUID of a corresponding test file. For user-defined test cases, the GUID is defined when using the Black-Box or White-Box test interface. The **InterfaceGuid** is made up of EFI Protocol GUIDs that are currently in testing. For example, there are three GUIDs specially defined in the *EFI 1.10 Specification* for the EFI services.

Boot Services: **E9EF7553-F833-4e56-96E8-38AE679523CC**

Runtime Services: **AFF115FB-387B-4c18-8C41-6AFC7F03BB90**

Generic Services: **71652D04-BF38-434a-BCB8-6547D7FD8384**

Using the category file, the list of test categories can be changed to suit your requirements. For example, the current UEFI SCT release provides test cases for testing protocol interfaces defined in the U*EFI 2.1 Specification*. You can integrate additional test cases for these depending on the EFI implementation on the target platform. A sample category file is shown below. The highlighting marks the places where the file can be modified.

```
[Category Data]
Revision         = 0x00010000
CategoryGuid   = 7AB1E93F-B439-4e2e-B773-CA540CEBCFEF
InterfaceGuid = E9EF7553-F833-4e56-96E8-38AE679523CC
Name               = EFI Spec\Boot Services Test\Event, Timer, and
Priority Services Test
Description    = Event, Timer, and Priority Services Test. Related to EFI
Spec 5.1.

[Category Data]
Revision         = 0x00010000
CategoryGuid   = CC129459-A197-4c8f-9422-2441E88C559A
InterfaceGuid = E9EF7553-F833-4e56-96E8-38AE679523CC
Name               = EFI Spec\Boot Services Test\Memory Allocation
Services Test
Description    = Memory Allocation Services Test. Related to EFI Spec
5.2.

[Category Data]
Revision         = 0x00010000
CategoryGuid   = {GUID of user-defined test}
InterfaceGuid = E9EF7553-F833-4e56-96E8-38AE679523CC
Name               = User-defined\Boot Services Test\Event, Timer, and
Priority Services Test
Description    = Event, Timer, and Priority Services Test. Related to XXX
design document.

[Category Data]
Revision         = 0x00010000
CategoryGuid   = {GUID of user-defined test}
InterfaceGuid = E9EF7553-F833-4e56-96E8-38AE679523CC
Name               = User-defined\Boot Services Test\Memory Allocation
Services Test
Description    = Memory Allocation Services Test. Related to XXX design
document.
```

Appendix C
SCRT Assertion Information

To accomplish a runtime service test, sometimes more than one step is required. For example, to test **GetVariable** service, set a certain variable first, and then get it to test. Encode the Port 80 number as **XY**. Here **X** stands for the runtime service sequence and **Y** stands for the step sequence in this service test. Corresponding to a unique Port 80 hex number, a unique GUID and the test description are printed out to COM1/COM2.The relationship is shown in Table 11.

Table 11 shows the detailed information for each assertion in the UEFI SCRT tests. It can be used by UEFI SCRT users as a case assertion reference.

Table 11 Test Case, Port 80 display and Log file Relationship for Each assertion

Test Case	Port80 Display	GUID	Assertion	Test Description
SetVariable	11	0xbff7e548, 0xf13a, 0x497c, 0x8e, 0x21, 0xae, 0xc2, 0x37, 0xa6, 0xcc, 0xe3	**RT.SetVariable** - Set a test variable named *UEFIRuntimeVariable*, should be **EFI_SUCCESS**	Call **RT.SetVariable** with the special name and Guid. And the variable is set with 8 Bytes data size. The return status should be **EFI_SUCCESS**.
	12	0xf556b5ad, 0xaace, 0x4bf0, 0xb7, 0x24, 0xe1, 0x29, 0xee, 0x0, 0xea, 0x37	**RT.SetVariable** - Clear a test variable named *UEFIRuntimeVariable*, should be **EFI_SUCCESS**	Call **RT.SetVariable** to clear the test variable. The return status should be **EFI_SUCCESS**.
GetVariable	21	0xd66e4a7f, 0x6d54, 0x4cc0, 0xb9, 0x3b, 0xf6, 0x2f, 0x48, 0x57, 0xa6, 0xff	**RT.SetVariable** - Set a test variable named *UEFIRuntimeVariable*, should be **EFI_SUCCESS**	Call **RT.SetVariable** with the special name and Guid. And the variable is set with 8 Bytes data size. The return status should be **EFI_SUCCESS**.
	22	0xaa5c5763, 0x36cd, 0x4f00, 0x84, 0x36, 0xf4, 0xa9, 0xd5, 0xaf, 0x12, 0xfb	**RT.GetVariable** - Get the test variable named *UEFIRuntimeVariable*, should be **EFI_SUCCESS**	Call **RT.GetVariable** to get the test variable. The return status should be **EFI_SUCCESS**
	23	0xbac20972, 0x9662, 0x4f24, 0x8a, 0xac, 0x66, 0x41, 0x42, 0xb5, 0x6d, 0xde	**RT.SetVariable** - Clear a test variable named *UEFIRuntimeVariable*, should be **EFI_SUCCESS**	Call **RT.SetVariable** to clear the test variable. The return status should be **EFI_SUCCESS**.

GetNextVariableName	31	0x8bcda7a3, 0x2848, 0x413d, 0xbf, 0x5, 0x7, 0xe1, 0x9, 0x8d, 0x42, 0xd2	**RT.SetVariable** - Set a test variable named *UEFIRuntimeVariable*, should be **EFI_SUCCESS**	Call **RT.SetVariable** with the special name and Guid. And the variable is set with 8 Bytes data size. The return status should be **EFI_SUCCESS**.
	32	0x67b4e72a, 0xc792, 0x4f74, 0x92, 0x1d, 0xea, 0xb3, 0x66, 0x4f, 0x95, 0x3b	**RT.GetNextVariableName** - Get the next variable name should be **EFI_SUCCESS/EFI_NOT_FOUND**	Loop call **RT.GetNextVariableName** get the next variable. The return status should be **EFI_SUCCESS/EFI_NOT_FOUND**.
	33	0xdbb5195f, 0x3584, 0x427d, 0xa1, 0x68, 0x3f, 0x5e, 0x1d, 0x24, 0x3b, 0xb9	**RT.SetVariable** - Clear a test variable named *UEFIRuntimeVariable*, should be **EFI_SUCCESS**	Call **RT.SetVariable** to clear the test variable. The return status should be **EFI_SUCCESS**.
QueryVariableInfo	41	0x8e75d9a9, 0x3c14, 0x4095, 0xbe, 0x76, 0xad, 0xcf, 0x55, 0xab, 0x8e, 0x6c	**RT.QueryVariableInfo** - Query Variable Information of the platform should be **EFI_SUCCESS.**	Call **RT.QueryVariableInfo** to query variable information. The return status should be **EFI_SUCCESS**.
GetTime	51	0xe8cd357a, 0xd254, 0x4f7b, 0x92, 0xc3, 0x23, 0xfd, 0x4d, 0xd6, 0xc0, 0xa3	**RT.GetTime** - Get the current time and date information should be **EFI_SUCCESS**	Call **RT.GetTime** with **NULL** capabilities. The return status should be **EFI_SUCCESS**.
SetTime	61	0x6417f479, 0xa174, 0x4614, 0x80, 0xcd, 0xe6, 0x96, 0x85, 0x8c, 0xd9, 0xfa	**RT.GetTime** - Get the current time and date information should be **EFI_SUCCESS**	Call **RT.GetTime** with **NULL** capabilities. The return status should be **EFI_SUCCESS**.
	62	0xd6a3c41a, 0xe6cf, 0x42fc, 0xa0, 0x39, 0x68, 0xf8, 0x39, 0xbb, 0xbf, 0xe3	**RT.SetTime** – set the same time as just got. should be **EFI_SUCCESS**	Set time. The return status should be **EFI_SUCCESS**.

SetWakeup Time	71	0xd6b952a9, 0x3d54, 0x4277, 0xbf, 0x60, 0xab, 0xfb, 0x3, 0x71, 0x5, 0xd5	**RT.GetTime** - Get the current time and date information should be **EFI_SUCCESS**	Call **RT.GetTime** with **NULL** capabilities. The return status should be **EFI_SUCCESS**.
	72	0x3f65c680, 0xae51, 0x4830, 0xb3, 0xd1, 0xd7, 0xc9, 0x2a, 0xcd, 0x14, 0x8a	**RT.SetWakeupTime** - Set wakeup time in 1 hour later from now on, should be **EFI_SUCCESS**	Call **RT.SetWakeupTime** to set wake up time, the time is 1 hour later from now on. The return status should be **EFI_SUCCESS**.

GetWakeupTime	81	0x4611524b, 0xbfd2, 0x42d4, 0x85, 0xa8, 0x9b, 0xf, 0xd1, 0xc6, 0x27, 0xd3	**RT.GetWakeupTime** - Get the current wakeup alarm clock setting information, should be **EFI_SUCCESS.**	Call **RT.GetWakeupTime** to get the current wake up time. The return status should be **EFI_SUCCESS**.
QueryCapsuleCapabilities	91	0x5c2cbd54, 0x1388, 0x4e87, 0xab, 0x11, 0x2c, 0x12, 0x3d, 0x24, 0x5, 0xbd	**RT.QueryCapsuleCapabilities** - Query the capsule capabilities with a **NULL MaxCapsuleSize**, should be **EFI_INVALID_PARAMETER.**	Call **RT.QueryCapsuleCapabilities** to query the capsule capabilities with a **NULL MaxCapsuleSize**. The return status should be **EFI_INVALID_PARAMETER**.
UpdateCapsule	A1	0x9e39a3e3, 0xcbb6, 0x4fcc, 0xb2, 0x21, 0x73, 0x24, 0x79, 0xf1, 0x21, 0x77	**RT.UpdateCapsule** – Update Capsules with 0 **CapsuleCount**, should be **EFI_INVALID_PARAMETER.** ***Note:*** *Because this case brings on some reset or update flash behavior, it is recommended disable as default. Users can enhance this test case for their own test platform.*	Call **RT.UpdateCapsule** with 0 **CapsuleCount.** The return status should be **EFI_INVALID_PARAMETER**.
GetNextHighMonotonicCount	B1	0xda790c1e, 0xdcbf, 0x4c0e, 0xaf, 0xf7, 0x46, 0x3a, 0xc4, 0x47, 0xb0, 0x6e	**RT.GetNextHighMonotonicCount** - First get next high monotonic counter, should be **EFI_SUCCESS.**	Call **RT.GetNextHighMonotonicCount** to get next high monotonic counter. The return status should be **EFI_SUCCESS**.

ResetSystem	C1	0x1bc049bb, 0xc371, 0x46cc, 0x8d, 0x98 0xef, 0x56 0xc, 0x35 0x7f, 0x1	**RT.ResetSystem** - Machine should have code reset! We should never come here.	**RT.ResetSystem** - Machine should have code reset! We should never come here.
	C2	0x11a541a4, 0xf75d, 0x42e0, 0xa8, 0x97, 0xe7, 0x92, 0xd4, 0x37, 0xc2, 0xfc	**RT.ResetSystem** - Machine should have warm reset! We should never come here	**RT.ResetSystem** - Machine should have warm reset! We should never come here
	C3	0xe5818568, 0x4723, 0x473f, 0xbc, 0x8f, 0xb5, 0x86, 0x2e, 0xd2, 0x5e, 0xb1	**RT.ResetSystem** - Machine should have shut down! We should never come here	**RT.ResetSystem** - Machine should have shut down! We should never come here