UEFI

Self-Certification Test

Case Writer's Guide

Version 0.91
July1, 2009

Revision History

Revision Number	Description	Revision Date
<0.9>	Internal draft.	June, 2009
0.91	Formatting and general editing; Contribution to UEFI	July, 2009

Contents

1 Introduction 1

1.1 Overview 1

1.2 Organization 1

1.3 Conventions Used in This Document 1

1.4 References 2

2 The UEFI SCT Source Package 3

2.1 Overview 3

2.2 Source Package Layout 3

2.3 System Requirements 4

2.4 Other Required Source for SCT Build 4

2.4.1 Required Source: 4

2.5 Build Process 4

2.5.1 Build UEFI SCT 5

2.5.2 Build IHV SCT 6

2.5.3 Build Step 6

3 Test Case Quick Started 7

3.1 3.1 Overview 7

3.2 The Components of Test Case 7

3.3 Test Case Description File 7

3.4 Header Files 9

3.4.1 ProtocolNameBBTestMain.h 9

3.4.2 ProtocolName.h 14

3.4.3 Guid.h 15

3.5 Source Files 16

3.5.1 ProtocolNameBBTestMain.c 16

3.5.2 <ProtocolName>Protocol.c 18

3.5.3 ProtocolNameBBTestFunction.c 19

3.5.4 ProtocolNameBBTestConformance.c 21

3.5.5 Guid.c 21

3.6 IHV Test Case 22

3.6.1 ProtocolNameBBTestMain.h 22

3.6.2 ProtocolName.c 22

3.6.3 ProtocolNameBBTestMain.c 22

4 Work Items for Build and Execution 23

4.1 Preparation for Build 23

4.2 Preparation for Execution 23

4.2.1 SctPkg\Config\Category.ini 23

4.2.2 SctPkg\Config\Data\GuidFile.txt 24
4.2.3 SctPkg\CommonGenFramework.bat 25
4.3 Required Work Items after Execution - Case Spec update 25
4.4 EFI_COMPONENT_NAME2_PROTOCOL Test 25
4.4.1 GetDriverName() 25
4.5 Other Required Work Items 26
Appendix A Test Support Services 27
A.1 Standard Test Services 27
A.2 Test Recovery Services 29
A.3 Test Logging Services 33
A.4 Test Profile Services 37

1 Introduction

1.1 Overview

This document defines the guidelines for writing the test cases under UEFI Self-Certification Test (SCT) and introduces the UEFI SCT source tree. This document is intended for anyone interested in developing or porting tests for the UEFI SCT. The main objective is to provide test case writers a standard for writing *UEFI 2.3 Specification* compliant test cases, with shared output, reporting and logging features. These features allow test case writers to concentrate their efforts on test functionality.

1.2 Organization

This UEFI SCT Case Writer's Guide is organized as listed below:

Table 1 Organization of the EFI SCT Case Writer's Guide

Chapter / Appendix	Description
1. Introduction	Introduces the UEFI SCT Writer's Guide and topics related to using the document.
2. Downloading and Building	Describes the downloading and building procedures of the UEFI SCT source tree.
3. Test Case Quick Started	Describes the detail procedures to create a test case which can be executed in the UEFI SCT Harness.
4. Preparations for Build and Execution	Describes the detail procedures before build and execution.
Appendix A. Test Support Services	Defines the test support protocols, and introduce the usage model.

1.3 Conventions Used in This Document

Term

Definition

UEFI

Unified Extensible Firmware Interface

UEFI Driver

Modular chunk of firmware code that supports chipset or platform features. Reusable in multiple system contexts. Code relating to a device or function that is run in the DXE phase of execution.

SCT

Self Certification Test

Shell

UEFI Shell is a simple, interactive “command interpreter” hat allows UEFI device drivers to be loaded, UEFI applications to be launched, and operating systems to be booted.

1.4 References

- *Unified Extensible Firmware Interface Specification*, Version 2.3
- *UEFI SCT Case Spec*, Version 2.1
- *UEFI SCT Getting Started*, Version 2.1
- *UEFI SCT User Guide*, Version 2.1
- *UEFI SCT Glossary*

2 The UEFI SCT Source Package

2.1 Overview

The chapter discusses the folder organization of UEFI SCT source package, how to prepare build environment and how to build it.

2.2 Source Package Layout

Besides the test cases, UEFI SCT source package all provides SCT test services, SCT core, SCRT and other applications and tools, shown in the figure below.

```
|--Include
|            |--Application
|            |--Config
|            |--Include
|            |--Library                                  //test services
|            |--Protocol                                 //test services
|--SctPkg--|--SCRT
           |              |--RIVL
           |              |
           |--TestCase--|--UEFI--|--EFI             // UEFI Test Case
           |                          |--IHV             // IHV Test Case
           |
           |--TestInfrastructure                      // SCT core
           |--Tools
           |--UEFI
```

2.3 System Requirements

Several Microsoft tools are used to build the UEFI SCT source package. The following tools must be installed on the development system.

- Microsoft Windows XP operation system
- Microsoft Visual Studio. NET 2003 Professional (7.1)
- Microsoft Windows Server 2003 Driver Development Kit (DDK), build 3790.1830
- SubVersion, an open source version control system, http://subversion.tigris.org/

Note: *Refer to https://edk2.tianocore.org/subversion_setup.html for the usage of subversion.*

2.4 Other Required Source for SCT Build

As mentioned in overview part, besides test case, SCT source package only includes SCT core, some APIs of test service. To build UEFI SCT source package, some other packages are needed to provide more libraries, or others.

2.4.1 Required Source:

- A stable version of BaseTools binary

Download BaseTools binary stable version - BaseTools(Binaries).zip from https://edk2.tianocore.org/servlets/ProjectDocumentList?folderID=103&expandFolder=103&folderID=102

- A stable version of EdkCompatibilityPkg

Check out one EdkCompatibilityPkg stable version 8621 from

https://svn.tianocore.org/svn/edk2/trunk/edk2/EdkCompatibilityPkg

- A stable version of Shellpkg

Check out one Shellpkg stable version 33 from

https://efi-shell.tianocore.org/svn/efi-shell/trunk/Shell

2.5 Build Process

According to the selected build target, two build processes are provided here.

2.5.1 Build UEFI SCT

2.5.1.1 Prepare the build directory

1. Extract UefiSctEdkII-Dev.zip to the UEFI SCT directory, for example, extract to C:\UefiSct, then two folders named as SctPkg and Include will be located in C:\ UefiSct

2. Extract BaseTools(Binaries).zip to C:\UefiSct, then two folders named as BaseTools and Conf, one batch file named as edksetup.bat will be located in C:\UefiSct

3. Check out one EdkCompatibilityPkg stable version 8621 to C:\UefiSct\EdkCompatibilityPkg

4. Check out one Shell stable version 33 to C:\UefiSct\EdkCompatibilityPkg\Other\Maintained\Application\Shell

Note: *Since the build tools of EFKII are friendly and flexible to support more compilers, we only need modify the C:\UefiSct\BaseTools\Conf\tools_def.template and make it support UEFI SCT. Before the first build process of UEFI SCT, please replace the section between line 819 and line 927 with the content of SctPkg\DDK3790.txt.*

2.5.1.2 Build Step

1. Run Visual Studio .NET 2003 Command Prompt to go to the command line environment. Let's make UEFI SCT X64 tip as an example.

2. Execute follow commands in turn.

```
cd C:\UefiSct

edksetup.bat

set efi_source=C:\UefiSct

set edk_source= C:\UefiSct\EdkCompatibilityPkg

copy SctPkg\Tools\Bin\GenBin.exe BaseTools\Bin\Win32\

build -p SctPkg\UEFI\UEFI_SCT.dsc -t DDK3790 -a X64

cd Build\UefiSct\DEBUG_DDK3790

..\..\..\SctPkg\CommonGenFramework.bat uefi_sct X64 InstallX64.efi
```

3. The target subdirectory named as SctPackageX64 which includes test cases and UEFI SCT applications will be generated and located at C:\UefiSct\Build\UefiSct\DEBUG_DDK3790

Note: *The example shows the build process of X64 only. For other architecture, please change the keyword, X64, which is highlighted in build step to IA32 or IPF.*

2.5.2 Build IHV SCT

Compared with the build process of UEFI SCT, to support IHV SCT, only several mini-changes need be updated for IHV SCT build.

2.5.3 Build Step

1. Run Visual Studio .NET 2003 Command Prompt to go to the command line environment. Let's make IHV SCT X64 tip as an example.
2. Execute follow commands in turn.
   a. cd C:\UefiSct
   b. edksetup.bat
   c. set efi_source=C:\UefiSct
   d. set edk_source= C:\UefiSct\EdkCompatibilityPkg
   e. copy SctPkg\Tools\Bin\GenBin.exe BaseTools\Bin\Win32\
   f. build -p SctPkg\UEFI\IHV_SCT.dsc -t DDK3790 -a X64
   g. cd Build\IHVSct\DEBUG_DDK3790
   h. ..\..\..\SctPkg\CommonGenFramework.bat ihv_sct X64 InstallX64.efi
3. The target subdirectory named as SctPackageX64 which includes test cases and UEFI SCT applications will be generated and located at C:\UefiSct\Build\IHVSct\DEBUG_DDK3790

3
Test Case Quick Started

3.1 3.1 Overview

This section describes how to create a simple test case to work within SCT infrastructure quickly. We will make BlockIo protocol test case as an example in this chapter. Generally, according to the execution mode, UEFI SCT test cases are divided into 2 types, native mode test case and passive mode test case. In this chapter, native mode test case will be introduced only. In native mode, test case can be divided into EFI/IHV for IBV/IHV too. The difference between them is very little, and will be introduced in the last section of this chapter.

3.2 The Components of Test Case

A SCT test case is generally composed of three types of files:

- INF: Test Case description file
- H: C source header file.
- C: C source code file

3.3 Test Case Description File

The INF file is the description file of test case. It is mainly used to generate the makefile of each component.

The following is the INF file for BlockIo protocol test case, BlockIoBBTest.inf

```
[defines]
BASE_NAME            = BlockIoBBTest
FILE_GUID            = A7DF1948-4CD3-4d24-B373-8B701C8AB9A4
COMPONENT_TYPE       = BS_DRIVER

[sources.common]
  BlockIoBBTestMain.c
  BlockIoBBTestMain.h
  BlockIoBBTestConformance.c
  BlockIoBBTestFunction.c
  BlockIoBBTestStress.c
  BlockIoProtocol.c
  Guid.c

[includes.common]
  .
  $(EDK_SOURCE)\Foundation
  $(EDK_SOURCE)\Foundation\Efi
  $(EDK_SOURCE)\Foundation\Include
  $(EDK_SOURCE)\Foundation\Efi\Include
  $(EDK_SOURCE)\Foundation\Framework\Include
  $(EDK_SOURCE)\Foundation\Include\IndustryStandard
  $(EDK_SOURCE)\Foundation\Library\Dxe\Include
  $(EFI_SOURCE)\
  $(EFI_SOURCE)\Include
  $(EFI_SOURCE)\SctPkg\Include
  $(EFI_SOURCE)\SctPkg\Library\EfiTestLib
  $(EFI_SOURCE)\SctPkg\Library\EfiTestUtilityLib

[libraries.common]
  EfiProtocolLib
  TestProtocolLib
  EfiCommonLib
  EfiTestLib
  EfiTestUtilityLib

[nmake.common]
  IMAGE_ENTRY_POINT = InitializeBBTestBlockIo
```

3.4 Header Files

Header files are used to store the global definitions, GUIDs, Function prototypes, header file inclusions and macros/definitions. Generally, there are 3 header files in the source code of test case.

- ProtocolNameBBTestMain.h, BlockIoBBTestMain.h here.
- ProtocolName.h, BlockIoProtocol.h here.
- Guid.h

3.4.1 ProtocolNameBBTestMain.h

It contains GUIDs for each test entry, and also specifies the set of UEFI SCT include files that the test case is dependent on.

```
#ifndef _BLOCK_IO_BBTEST_H_
#define _BLOCK_IO_BBTEST_H_

//
// Includes
//

#include "Efi.h"
#include "BlockIoProtocol.h"
#include "Guid.h"
#include "EfiTestUtilityLib.h"
#include "EfiTestLib.h"

//
// Definitions
//
#define BLOCK_IO_PROTOCOL_TEST_REVISION     0x00010000

#define MAX_NUMBER_OF_READ_BLOCK_BUFFER     20
#define MAX_DIFFERENT_BUFFERSIZE_FOR_TEST   4
#define MAX_DIFFERENT_LBA_FOR_TEST          4
#define MAX_DIFFERENT_IOALIGN_FOR_TEST      4
#define MAX_REPEAT_OF_STRESS_TEST           20
```

```
#define MAXIMUM(a,b)                          ((a)>(b)?(a):(b))
#define MINIMUM(a,b)                          ((a)<(b)?(a):(b))

//
// Entry GUIDs
//

//
// Conformance
//
#define BLOCK_IO_PROTOCOL_READBLOCKS_CONFORMANCE_AUTO_GUID \
  { 0x826159d3, 0x4a5, 0x4cce, \
  { 0x84, 0x31, 0x34, 0x47, 0x7, 0xa8, 0xb5, 0x7a } }

#define BLOCK_IO_PROTOCOL_READBLOCKS_CONFORMANCE_MANUAL_GUID \
  { 0xd743e08, 0xe3fd, 0x401b, \
  { 0x95, 0xc4, 0x98, 0x61, 0x1f, 0x65, 0xf2, 0xee } }

#define BLOCK_IO_PROTOCOL_WRITEBLOCKS_CONFORMANCE_AUTO_GUID \
  { 0x75961463, 0x882b, 0x4afd, \
  { 0x87, 0xcc, 0x7a, 0xfb, 0xd4, 0xd8, 0xff, 0x32 } }

#define BLOCK_IO_PROTOCOL_WRITEBLOCKS_CONFORMANCE_MANUAL_GUID \
  { 0x8b983f70, 0x7478, 0x4f59, \
  { 0xbe, 0xad, 0x16, 0xd1, 0x89, 0x0, 0xa3, 0xd9 } }

#define BLOCK_IO_PROTOCOL_FLUSHBLOCKS_CONFORMANCE_AUTO_GUID \
  { 0x759c5229, 0x272a, 0x45aa, \
  { 0xa6, 0x16, 0xb5, 0x62, 0x26, 0x19, 0xeb, 0x6 } }

#define BLOCK_IO_PROTOCOL_FLUSHBLOCKS_CONFORMANCE_MANUAL_GUID \
  { 0x5c484ad1, 0xa3bc, 0x495a, \
  { 0xa8, 0x82, 0xd6, 0xd8, 0x9d, 0x76, 0xf2, 0xa } }

//
// Function
//
#define BLOCK_IO_PROTOCOL_RESET_FUNCTION_AUTO_GUID \
  { 0xb0501c52, 0x1564, 0x4945, \
  { 0xb5, 0x7b, 0x1f, 0xd2, 0xc8, 0x82, 0xff, 0x91 } }

#define BLOCK_IO_PROTOCOL_READBLOCKS_FUNCTION_AUTO_GUID \
  { 0xa4284ae9, 0xc988, 0x4230, \
  { 0x97, 0x3a, 0x5c, 0x19, 0x39, 0x8c, 0x82, 0x54 } }
```

```
#define BLOCK_IO_PROTOCOL_WRITEBLOCKS_FUNCTION_AUTO_GUID \
  { 0xb7e32978, 0x4a8, 0x414d, \
  { 0x99, 0x7b, 0x59, 0x62, 0x8d, 0x73, 0xbf, 0x0 } }

#define BLOCK_IO_PROTOCOL_MEDIA_INTEGRITY_MANUAL_GUID \
  { 0x616fdaf9, 0x413f, 0x4a8c, \
  { 0xac, 0xd5, 0xa5, 0xac, 0x72, 0x71, 0x2c, 0xf2 } }

#define BLOCK_IO_PROTOCOL_FLUSHBLOCKS_FUNCTION_AUTO_GUID \
  { 0x7c947313, 0xc8a3, 0x4abd, \
  { 0xa1, 0xf2, 0x86, 0x70, 0xee, 0x57, 0x4b, 0xe5 } }

//
// Combination
//
#define BLOCK_IO_PROTOCOL_RESET_STRESS_AUTO_GUID \
  { 0xc29d4482, 0x3341, 0x4bbb, \
  { 0x8c, 0xa3, 0x82, 0xf2, 0x77, 0x2d, 0x98, 0x80 } }

#define BLOCK_IO_PROTOCOL_READBLOCKS_STRESS_AUTO_GUID \
  { 0x4fdea1da, 0x643d, 0x4e71, \
  { 0x82, 0xd7, 0xad, 0xd1, 0xc6, 0x4d, 0x70, 0x9c } }

#define BLOCK_IO_PROTOCOL_WRITEBLOCKS_STRESS_AUTO_GUID \
  { 0x8619522b, 0xc0bb, 0x48ea, \
  { 0xb9, 0xa2, 0x23, 0x0, 0x69, 0xfd, 0x4, 0xa9 } }

//
// Global variables
//

extern   EFI_EVENT               TimerEvent;

//
// Prototypes
//

EFI_STATUS
InitializeBBTestBlockIo (
  IN EFI_HANDLE                 ImageHandle,
  IN EFI_SYSTEM_TABLE           *SystemTable
  );
```

```
EFI_STATUS
BBTestBlockIoProtocolUnload (
  IN EFI_HANDLE                 ImageHandle
  );

//
// Conformance test case prototypes
//

EFI_STATUS
BBTestReadBlocksConformanceAutoTest (
  IN EFI_BB_TEST_PROTOCOL       *This,
  IN VOID                       *ClientInterface,
  IN EFI_TEST_LEVEL             TestLevel,
  IN EFI_HANDLE                 SupportHandle
  );

EFI_STATUS
BBTestReadBlocksConformanceManualTest (
  IN EFI_BB_TEST_PROTOCOL       *This,
  IN VOID                       *ClientInterface,
  IN EFI_TEST_LEVEL             TestLevel,
  IN EFI_HANDLE                 SupportHandle
  );

EFI_STATUS
BBTestWriteBlocksConformanceAutoTest (
  IN EFI_BB_TEST_PROTOCOL       *This,
  IN VOID                       *ClientInterface,
  IN EFI_TEST_LEVEL             TestLevel,
  IN EFI_HANDLE                 SupportHandle
  );

EFI_STATUS
BBTestWriteBlocksConformanceManualTest (
  IN EFI_BB_TEST_PROTOCOL       *This,
  IN VOID                       *ClientInterface,
  IN EFI_TEST_LEVEL             TestLevel,
  IN EFI_HANDLE                 SupportHandle
  );

EFI_STATUS
BBTestFlushBlocksConformanceAutoTest (
  IN EFI_BB_TEST_PROTOCOL       *This,
```

```
  IN VOID                       *ClientInterface,
  IN EFI_TEST_LEVEL             TestLevel,
  IN EFI_HANDLE                 SupportHandle
  );


EFI_STATUS
BBTestFlushBlocksConformanceManualTest (
  IN EFI_BB_TEST_PROTOCOL       *This,
  IN VOID                       *ClientInterface,
  IN EFI_TEST_LEVEL             TestLevel,
  IN EFI_HANDLE                 SupportHandle
  );

//
// Function test case prototypes
//

EFI_STATUS
BBTestResetFunctionAutoTest (
  IN EFI_BB_TEST_PROTOCOL       *This,
  IN VOID                       *ClientInterface,
  IN EFI_TEST_LEVEL             TestLevel,
  IN EFI_HANDLE                 SupportHandle
  );

EFI_STATUS
BBTestReadBlocksFunctionAutoTest (
  IN EFI_BB_TEST_PROTOCOL       *This,
  IN VOID                       *ClientInterface,
  IN EFI_TEST_LEVEL             TestLevel,
  IN EFI_HANDLE                 SupportHandle
  );

EFI_STATUS
BBTestWriteBlocksFunctionAutoTest (
  IN EFI_BB_TEST_PROTOCOL       *This,
  IN VOID                       *ClientInterface,
  IN EFI_TEST_LEVEL             TestLevel,
  IN EFI_HANDLE                 SupportHandle
  );

EFI_STATUS
BBTestFlushBlocksFunctionAutoTest (
```

```
  IN EFI_BB_TEST_PROTOCOL       *This,
  IN VOID                       *ClientInterface,
  IN EFI_TEST_LEVEL             TestLevel,
  IN EFI_HANDLE                 SupportHandle
  );
```

3.4.2 ProtocolName.h

The second one is ProtocolName.h, BlockIoProtocol.h here, which provides the prototype definitions for the specified protocol. Generally, these definitions can be found in UEFI Spec, and all UEFI compatible implementation should to follow these definitions.

```
#ifndef _EFI_BLOCK_IO_FOR_TEST_H_
#define _EFI_BLOCK_IO_FOR_TEST_H_

#define EFI_BLOCK_IO_PROTOCOL_GUID \
  { 0x964e5b21, 0x6459, 0x11d2, \
  0x8e, 0x39, 0x0, 0xa0, 0xc9, 0x69, 0x72, 0x3b }

EFI_FORWARD_DECLARATION(EFI_BLOCK_IO);

typedef struct {
  UINT32            MediaId;
  BOOLEAN           RemovableMedia;
  BOOLEAN           MediaPresent;
  BOOLEAN           LogicalPartition;
  BOOLEAN           ReadOnly;
  BOOLEAN           WriteCaching;
  UINT32            BlockSize;
  UINT32            IoAlign;
  EFI_LBA           LastBlock;
} EFI_BLOCK_IO_MEDIA;

typedef UINT64          EFI_LBA;

typedef
EFI_STATUS
(EFIAPI *EFI_BLOCK_RESET) (
  IN EFI_BLOCK_IO    *This,
  IN BOOLEAN         ExtendedVerification
  );
```

```
typedef
EFI_STATUS
(EFIAPI *EFI_BLOCK_READ) (
  IN EFI_BLOCK_IO    *This,
  IN UINT32           MediaId,
  IN EFI_LBA          LBA,
  IN UINTN            BufferSize,
  OUT VOID            *Buffer
  );

typedef
EFI_STATUS
(EFIAPI *EFI_BLOCK_WRITE) (
  IN EFI_BLOCK_IO    *This,
  IN UINT32           MediaId,
  IN EFI_LBA          LBA,
  IN UINTN            BufferSize,
  IN VOID             *Buffer
  );

typedef
EFI_STATUS
(EFIAPI *EFI_BLOCK_FLUSH) (
  IN EFI_BLOCK_IO     *This
  );

typedef struct _EFI_BLOCK_IO {
  UINT64                  Revision;
  EFI_BLOCK_IO_MEDIA      *Media;
  EFI_BLOCK_RESET         Reset;
  EFI_BLOCK_READ          ReadBlocks;
  EFI_BLOCK_WRITE         WriteBlocks;
  EFI_BLOCK_FLUSH         FlushBlocks;
} EFI_BLOCK_IO_PROTOCOL;

extern EFI_GUID                          gEfiBlockIoProtocolGuid;

#endif
```

3.4.3 Guid.h

The third one is Guid.h, which defines the globally unique identifiers(GUID) value for all test assertions. The GUID value is generated by Microsoft Visual Studio.

3.5 Source Files

Source files contain the code for a test case. Generally, there are five source files at least.

- ProtocolNameBBTestMain.c, BlockIoBBTestMain.c here.
- ProtocolName.c, BlockIoProtocol.c here.
- ProtocolNameBBTestConformance.c, BlockIoBBTestConformance.c here.
- ProtocolNameBBTestFunction.c, BlockIoBBTestFunction.c here
- Guid.c

There are several important pieces in a source file for a test case.

3.5.1 ProtocolNameBBTestMain.c

In this file, an individual protocol test is defined by the UEFI Black-Box Test Protocol structure. This structure contains meta-data regarding the test, including revision, category, and human readable test descriptions. Each test is required to have a protocol field.

```
EFI_BB_TEST_PROTOCOL_FIELD gBBTestProtocolField = {
  BLOCK_IO_PROTOCOL_TEST_REVISION,
  EFI_BLOCK_IO_PROTOCOL_GUID,
  L"Block I/O Protocol Test",
  L"UEFI 2.0 Block I/O Protocol Test"
};
```

Each test is required to have a GUID, which is included in the test protocol field as a structure of this form:

```
EFI_GUID gSupportProtocolGuid1[2] = {
  EFI_STANDARD_TEST_LIBRARY_GUID,
  EFI_NULL_GUID
};
```

An individual protocol test may be composed of several test cases, which are specified by the Test Entry Field.

```
EFI_BB_TEST_ENTRY_FIELD gBBTestEntryField[] = {
  //
  // Conformance test section
  //
  {
    BLOCK_IO_PROTOCOL_READBLOCKS_CONFORMANCE_AUTO_GUID,
    L"ReadBlocks_Conf",
    L"Perform auto consistency checkes on the ReadBlocks interface",
    EFI_TEST_LEVEL_MINIMAL,
    gSupportProtocolGuid1,
    EFI_TEST_CASE_AUTO,
    BBTestReadBlocksConformanceAutoTest
  },
  ……
  //
  // Function test section
  //
  {
    BLOCK_IO_PROTOCOL_RESET_FUNCTION_AUTO_GUID,
    L"Reset_Func",
    L"Invoke Reset and verify interface correctness within test case",
    EFI_TEST_LEVEL_DEFAULT,
    gSupportProtocolGuid1,
    EFI_TEST_CASE_AUTO,
    BBTestResetFunctionAutoTest
  },
  ……
}
```

The entry and exit point of an individual protocol test are also defined in this file.

```
EFI_DRIVER_ENTRY_POINT(InitializeBBTestBlockIo)

EFI_STATUS
InitializeBBTestBlockIo (
  IN EFI_HANDLE           ImageHandle,
  IN EFI_SYSTEM_TABLE    *SystemTable
  )
{
  EfiInitializeTestLib (ImageHandle, SystemTable);
  InitializeLib (ImageHandle, SystemTable);

  gtBS->CreateEvent (EFI_EVENT_TIMER, 0, NULL, NULL, &TimerEvent);

  return EfiInitAndInstallBBTestInterface (
             &ImageHandle,
             &gBBTestProtocolField,
             gBBTestEntryField,
             BBTestBlockIoProtocolUnload,
             &gBBTestProtocolInterface
             );
}

EFI_STATUS
BBTestBlockIoProtocolUnload (
  IN EFI_HANDLE       ImageHandle
  )
{
  EFI_STATUS          Status;

  gtBS->CloseEvent (TimerEvent);

  Status = EfiUninstallAndFreeBBTestInterface (
               ImageHandle,
               gBBTestProtocolInterface
               );

  return Status;
}
```

3.5.2 <ProtocolName>Protocol.c

Only gProtocolNameGuid is defined in this file.

```
#include "Efi.h"

#include "BlockIoProtocol.h"

EFI_GUID gEfiBlockIoProtocolGuid = EFI_BLOCK_IO_PROTOCOL_GUID;

EFI_GUID_STRING(&gEfiBlockIoProtocolGuid, "BlockIo Protocol", "EFI 1.0 Block IO
protocol");
```

3.5.3 ProtocolNameBBTestFunction.c

In this file, all test cases are designed to call the API of Protocol with correct arguments or available resource. The API should return EFI_SUCCESS in most cases.

```
EFI_STATUS
BBTestResetFunctionAutoTest (
  IN EFI_BB_TEST_PROTOCOL              *This,
  IN VOID                              *ClientInterface,
  IN EFI_TEST_LEVEL                    TestLevel,
  IN EFI_HANDLE                        SupportHandle
  )
{
  EFI_STANDARD_TEST_LIBRARY_PROTOCOL   *StandardLib;
  EFI_STATUS                           Status;
  EFI_BLOCK_IO_PROTOCOL                 *BlockIo;
  EFI_TEST_ASSERTION                    AssertionType;

  //
  // Get the Standard Library Interface
  //
  Status = gtBS->HandleProtocol (
                     SupportHandle,
                     &gEfiStandardTestLibraryGuid,
                     &StandardLib
                     );
```

```
  if (EFI_ERROR(Status)) {
    StandardLib->RecordAssertion (
                        StandardLib,
                        EFI_TEST_ASSERTION_FAILED,
                        gTestGenericFailureGuid,
                        L"BS.HandleProtocol - Handle standard test library",
                        L"%a:%d:Status - %r",
                        __FILE__,
                        (UINTN)__LINE__,
                        Status
                        );
    return Status;
  }
  //The protocol want to be tested is found by SCT infrastructure and transferred
as a parameter.
  BlockIo = (EFI_BLOCK_IO_PROTOCOL *)ClientInterface;

  //
  // Reset must succeed to reset the block device with extended verification
  //
  Status = BlockIo->Reset (BlockIo, TRUE);
  if (EFI_ERROR(Status)) {
    AssertionType = EFI_TEST_ASSERTION_FAILED;
  } else {
    AssertionType = EFI_TEST_ASSERTION_PASSED;
  }

  Print (L"Wait 5 seconds for the block device resetting...");
  gtBS->Stall (5000000);

  StandardLib->RecordAssertion (
                    StandardLib,
                    AssertionType,
                    gBlockIoFunctionTestAssertionGuid001,
                    L"EFI_BLOCK_IO_PROTOCOL.Reset - Reset the block device with
extended verification",
                    L"%a:%d:Status=%r",
                    __FILE__,
                    (UINTN)__LINE__,
                    Status
                    );

  //
// Reset must succeed to reset the block device without extended verification
```

```
  //
  Status = BlockIo->Reset (BlockIo, FALSE);
  if (EFI_ERROR(Status)) {
    AssertionType = EFI_TEST_ASSERTION_FAILED;
  } else {
    AssertionType = EFI_TEST_ASSERTION_PASSED;
  }

  Print (L"Wait 5 seconds for the block device resetting...");
  gtBS->Stall (5000000);

  StandardLib->RecordAssertion (
                   StandardLib,
                   AssertionType,
                   gBlockIoFunctionTestAssertionGuid002,
                   L"EFI_BLOCK_IO_PROTOCOL.Reset - Reset the block device
without extended verification",
                   L"%a:%d:Status=%r",
                   __FILE__,
                  (UINTN)__LINE__,
                   Status
                   );

  return EFI_SUCCESS;
}
```

For **gTestGenericFailureGuid**, it is not test case/test assertion fail, and just means test environment is not ready for test case. For one test case, it may be composed of several test assertion points at most case (2 assertion points are tested in this example). You can implement all test assertions in one function as this instance, or you can implement one test assertion as one sub function and test case call all sub functions in turn.

3.5.4 ProtocolNameBBTestConformance.c

In this file, all test cases are designed to use the API of Protocol with various combinations of invalid arguments or unavailable resource. The API should fail and should not return EFI_SUCCESS. Each invalid parameter combination, all but one of its parameters is set to an inadmissible value, need be verified.

3.5.5 Guid.c

This file defines the globally unique identifiers (GUID) for all test assertions.

3.6 IHV Test Case

IHV test cases are designed for IHV, and user not only can specify the device from the device list, but also can specify the test case thru menu. This design gives IHV more convenience and freedom. For some

In essence, IHV test cases are similar with EFI test cases, and only 3 points need be updated.

3.6.1 ProtocolNameBBTestMain.h

- **IHV_COMPONENT_NAME2_PROTOCOL_GUID** need to be defined.
- **IHV_COMPONENT_NAME2_TEST_REVISION** need to be defined.

3.6.2 ProtocolName.c

- The definition of **gEfiComponentName2ProtocolGuid** need to be removed.

3.6.3 ProtocolNameBBTestMain.c

- The member of structure gBBTestProtocolField need to be updated with **IHV_COMPONENT_NAME2_PROTOCOL_GUID** and **IHV_COMPONENT_NAME2_TEST_REVISION**
- The entry point need be updated with **EfiInitAndInstallIHVBBTestInterface**
- The exit point need be updated with **EfiUninstallAndFreeIHVBBTestInterface**

4 Work Items for Build and Execution

After the new test case has been developed, some Meta files need be modified to include new test case into build system and let UEFI SCT infrastructure detect new test case while executing.

4.1 Preparation for Build

In order to add the new test case into build system, the content in "\SctPkg\UEFI\UEFI_SCT.dsc" or "\SctPkg\UEFI\IHV_SCT.dsc" need be updated. The example below shows a new highlighted test case is inserted in the DSC file.

Example

```
SctPkg\TestCase\UEFI\EFI\Protocol\DriverSupportedEfiVersion\BlackBoxTest\Dri
verSupportedEfiVersionBBTest.inf
SctPkg\TestCase\UEFI\EFI\Protocol\AbsolutePointer\BlackBoxTest\AbsolutePoint
erBBTest.inf
SctPkg\TestCase\UEFI\EFI\Protocol\PlatformToDriverConfiguration\BlackBoxTest
\PlatformToDriverConfigurationBBTest.inf

SctPkg\TestCase\UEFI\EFI\Protocol\SpecifiedFolderforSpecifiedProtocol\BlackB
oxTest\SpecifedProtocolBBTest.inf

#
# Dependency files for UEFI\EFI Compliant Test
#
```

4.2 Preparation for Execution

After build successfully, several file need be updated which make the new test case be detected by UEFI SCT infrastructure while running.

4.2.1 SctPkg\Config\Category.ini

The new test case information need be inserted in this INI file which constructs the relationship between the catalogs in UI execution mode with actual test case. The example below shows a new test case for EFI Component Name2 Protocol is inserted in the INI file. Here, **CategoryGuid/InterfaceGuid** is **6a7a5cff-e8d9-4f70-bada-75ab3025ce14**, which is specified in UEFI Spec and defined

SctPkg\TestCase\UEFI\EFI\Protocol\ComponentName2\BlackBoxTest\ComponentName2BB TestMain.h in **EFI_COMPONENT_NAME2_PROTOCOL_GUID**. The value of *Name* is 'Driver Model Test\ Component Name2 Protocol Test', since EFI Component Name2 Protocol is classified to UEFI Driver Model by UEFI Spec. The value of *Description* is blank here.

Note: *For new IHV test case, corresponding content need be updated in this file too. The difference is CategoryGuid value,* ***764165ed-a8ae-4dff-a3f2-6e99c5565690****, which is defined in SctPkg\TestCase\UEFI\IHV\Protocol\ComponentName2\BlackBoxTest\ComponentName 2BBTestMain.h with* ***IHV_COMPONENT_NAME2_PROTOCOL_GUID****. Besides this, Name need be updated with* ***IHV\Driver Model Test\Component Name2 Protocol Test*** *too.*

Example

```
[Category Data]
Revision      = 0x00010000
CategoryGuid  = 6a7a5cff-e8d9-4f70-bada-75ab3025ce14
InterfaceGuid = 6a7a5cff-e8d9-4f70-bada-75ab3025ce14
Name          = Driver Model Test\Component Name2 Protocol Test
Description   =

[Category Data]
Revision      = 0x00010000
CategoryGuid  = 764165ed-a8ae-4dff-a3f2-6e99c5565690
InterfaceGuid = 6a7a5cff-e8d9-4f70-bada-75ab3025ce14
Name          = IHV\Driver Model Test\Component Name2 Protocol Test
Description   = Component Name2 Protocol Test on IHV Drivers
```

4.2.2 SctPkg\Config\Data\GuidFile.txt

All assertions information is stored in this TXT file. Thru this file, UEFI SCT provides the 'report generation' feature which retrieves all log files, matches them with this file by Guid, and generate one Excel compatible report file. Report file gives an overall result and user can find the failure rapidly. Moreover, user can find the test description in the *UEFI SCT Case Specification*'by searching the index.

Note: *Developers need to update this TXT file carefully and the information should be consistent with the implementation of the corresponding test case. In the example,* ***31518904-1307-4BEF-84E6-66FF76A78FF4*** *is one assertion Guid of assertion point in the test case,* ***COMPONENT_NAME2_PROTOCOL.GetDriverName - GetDriverName() returns EFI_INVALID_PARAMETER with NULL Language*** *is the assertion description of the assertion point,* ***5.5.8.1.1*** *is the index which should be exclusive in this file.*

Example

```
#
31518904-1307-4BEF-84E6-66FF76A78FF4
COMPONENT_NAME2_PROTOCOL.GetDriverName - GetDriverName() returns
EFI_INVALID_PARAMETER with NULL Language
5.5.8.1.1
#
7B478492-53C0-4748-A244-60F3F2D0EE5A
COMPONENT_NAME2_PROTOCOL.GetDriverName - GetDriverName() returns
EFI_INVALID_PARAMETER with NULL DriverName
5.5.8.1.2
#
36E0A7E5-BFC8-4AB9-B41A-9D6925436AD2
COMPONENT_NAME2_PROTOCOL.GetDriverName - GetDriverName() returns
EFI_UNSUPPORTED with unsupported language
5.5.8.1.3
```

4.2.3 SctPkg\CommonGenFramework.bat

After a successful build, this batch file must be updated with an inserted line to ensure the binary of new test case will be integrated into the binary package.

Example

```
copy %ProcessorType%\ComponentName2BbTest.efi       %Framework%\Test\ > NUL
```

4.3 Required Work Items after Execution - Case Spec update

After the new test case is validated, some work items need be completed as the integrality criteria for a new test case. Besides the works mentioned in this chapter above, test case developer need update ‘UEFI SCT Case Spec’ with the test case implementation, and ensure it is consistent with GuidFile.txt. This work is very important and will help SCT user to locate failure reason rapidly with ‘UEFI SCT Case Spec’. The table below is abstracted from ‘UEFI SCT Case Spec’ as a template.

Example

4.4 EFI_COMPONENT_NAME2_PROTOCOL Test

4.4.1 GetDriverName()

No.	GUID	Assertion	Test Description
5.5.8.1.1	0x31518904, 0x1307, 0x4bef, 0x84, 0xe6, 0x66, 0xff, 0x76, 0xa7, 0x8f, 0xf4	COMPONENT_NAME2_PROTOCOL. GetDriverName - GetDriverName() returns EFI_INVALID_PARAMETER with NULL Language	Call GetDriverName() with Language being NULL. The returned status should be EFI_INVALID_PARAMETER
5.5.8.1.2	0x7b478492, 0x53c0, 0x4748, 0xa2, 0x44, 0x60, 0xf3, 0xf2, 0xd0, 0xee, 0x5a	COMPONENT_NAME2_PROTOCOL. GetDriverName - GetDriverName() returns EFI_INVALID_PARAMETER with NULL DriverName	Call GetDriverName() with DriverName being NULL. The returned status should be EFI_INVALID_PARAMETER
5.5.8.1.3	0x36e0a7e5, 0xbfc8, 0x4ab9, 0xb4, 0x1a, 0x9d, 0x69, 0x25, 0x43, 0x6a, 0xd2	COMPONENT_NAME2_PROTOCOL. GetDriverName - GetDriverName() returns EFI_UNSUPPORTED with unsupported language	Call GetDriverName() with unsupported Language. The returned status should be EFI_UNSUPPORTED
5.5.8.1.4	0x327aa49d, 0x4a8b, 0x4101, 0x8b, 0x0d, 0x92, 0x32, 0x33, 0xfc, 0x09, 0xe5	COMPONENT_NAME2_PROTOCOL. GetDriverName - GetDriverName() returns EFI_SUCCESS with supported languange	Call GetDriverName() with supported Language. The returned status should be EFI_SUCCESS

Note: *Developer need filled in one table which will be integrated into the UEFI SCT Case Specification and the test case information in the table should be consistent with both the implementation of corresponding test case and the related content in GuidFile.txt.*

4.5 Other Required Work Items

Currently, as one separated application, SCRT is included in SCT official release package, so *UEFI SCT Getting Started* and *UEFI SCT User Guide* are updated to add corresponding description, such as build process and usage model, which give the user a good experience. So, if SCT developer wants to add some applications or makes some changes about usage model, please update these two documents too.

Appendix A
Test Support Services

Besides the test case, the UEFI SCT Package also provides some test service Protocols that test writers can use to develop test case that work within the UEFI SCT Harness.

A.1 Standard Test Services

The standard test services are provided by an EFI Standard Test Library Protocol. This protocol is responsible for recording/reporting test results and test messages, and tracking the number of test results of each type.

Protocol Interface Structure

```
typedef struct _EFI_STANDARD_TEST_LIBRARY_PROTOCOL {
  UINT64                                    LibraryRevision;
  CHAR16                                    *Name;
  CHAR16                                    *Description;
  EFI_RECORD_ASSERTION                      RecordAssertion;
  EFI_RECORD_MESSAGE                        RecordMessage;
  EFI_GET_RESULT_COUNT                      GetResultCount;
} EFI_STANDARD_TEST_LIBRARY_PROTOCOL;

typedef
EFI_STATUS
(EFIAPI *EFI_RECORD_ASSERTION) (
  IN  EFI_STANDARD_TEST_LIBRARY_PROTOCOL      *This,
  IN  EFI_TEST_ASSERTION                      Type,
  IN  EFI_GUID                                EventId,
  IN  CHAR16                                  *Description,
  IN  CHAR16                                  *Detail,
  ...
  )
```

```
typedef
EFI_STATUS
(EFIAPI *EFI_RECORD_MESSAGE) (
  IN  EFI_STANDARD_TEST_LIBRARY_PROTOCOL         *This,
  IN  EFI_VERBOSE_LEVEL                          VerboseLevel,
  IN  CHAR16                                     *Message,
  ...
  )

typedef
EFI_STATUS
(EFIAPI *EFI_GET_RESULT_COUNT) (
  IN  EFI_STANDARD_TEST_LIBRARY_PROTOCOL          *This,
  IN  EFI_TEST_ASSERTION                          Type,
  OUT UINT32                                      *Count
  )
```

Usage Model

```
{
  EFI_STATUS                                 Status;
  EFI_STANDARD_TEST_LIBRARY_PROTOCOL         *StandardLib;
  UINT32                                     PassedCount;

//
// Locate the test support libraries
//
Status = BS->LocateProtocol (
                 &gEfiStandardTestLibraryGuid,
                 NULL,
                 &StandardLib
                   );
  If (EFI_ERROR (Status)) {
  return EFI_UNSUPPORTED;
}

//
// Record assertion
//
StandardLib->RecordAssertion (
StandardLib,
EFI_TEST_ASSERTION_PASSED,
gSampleAppTestAssertionGuid001,
L"Sample Assertion 001",
L"%a:%d:put the detail information at here",
__FILE__,
```

```
__LINE__
);
  //
// Record message
//
StandardLib->RecordMessage (
StandardLib,
EFI_VERBOSE_LEVEL_DEFAULT,
L" %a:%d:put the detail information at here",
__FILE__,
__LINE__
);
  //
// Get result count
//
StandardLib->GetResultCount (
StandardLib,
EFI_TEST_ASSERTION_PASSED,
&PassedCount
);

Print (L"Passed Count: %d\n", PassedCount);

return EFI_SUCCESS;
}
```

A.2 Test Recovery Services

The test recovery services are provided by an EFI Test Recovery Library Protocol. The protocol is responsible for handling the system reset in the test execution. All test cases that are required to reset the test system in the test execution could use the test recovery services to set and get the recovery data.

Protocol Interface Structure

```
typedef struct _EFI_TEST_RECOVERY_LIBRARY_PROTOCOL {
  UINT64                                    LibraryRevision;
  CHAR16                                    *Name;
  CHAR16                                    *Description;
  EFI_TRL_READ_RESET_RECORD                 ReadResetRecord;
  EFI_TRL_WRITE_RESET_RECORD                WriteResetRecord;
} EFI_TEST_RECOVERY_LIBRARY_PROTOCOL;

typedef
EFI_STATUS
(EFIAPI *EFI_TRL_READ_RESET_RECORD) (
  IN  EFI_TEST_RECOVERY_LIBRARY_PROTOCOL          *This,
  OUT UINTN                                       *Size,
  OUT VOID                                        *Buffer
  )

typedef
EFI_STATUS
(EFIAPI *EFI_TRL_WRITE_RESET_RECORD) (
  IN  EFI_TEST_RECOVERY_LIBRARY_PROTOCOL          *This,
  IN  UINTN                                       Size,
  IN  VOID                                        *Buffer
  )
```

Usage Model

The function below uses the test recovery service to verify the variable still exists across a power cycle.

```
{
  EFI_STATUS                                  Status;
  EFI_ TEST_RECOVERY_LIBRARY_PROTOCOL         *StandardLib;
  UINT32                                      Attributes;
  UINTN                                       BufferSize;
  UINT8                                       BufferSize;
  UINTN                                       DataSize;
  UINT8                                       Data[10] = {0};
```

```
//
// Locate the test support libraries
//
Status = BS->LocateProtocol (
             &gEfiTestRecoveryLibraryGuid,
             NULL,
             &RecoveryLib
               );
  If (EFI_ERROR (Status)) {
  return EFI_UNSUPPORTED;
}

//
// Read the recovery record
//
Status = RecoveryLib->ReadResetRecord (
RecoveryLib,
&BufferSize,
Buffer
);
  If (EFI_ERROR (Status)) {
  return EFI_UNSUPPORTED;
}
If (BufferSize == 0) || (Buffer[0] == 0)) {
  //
// Set a variable
  //
  Attribute = EFI_VARIABLE_NON_VOLATILE        |
               EFI_VARIABLE_BOOTSERVICE_ACCESS |
               EFI_VARIABLE_RUNTIME_ACCESS;
  DataSize = 10;

  Status = RT->SetVariable (
                   L"Test Variable",
                   &gEfiTestVariableGuid,
                   Attributes,
                   DataSize,
                   Data
);
    If (EFI_ERROR (Status)) {
    return EFI_UNSUPPORTED;
}

  //
```

```
// Write the recovery record
//
  Buffer[0] = 1;
  Status = RecoveryLib->WriteResetRecord (
RecoveryLib,
1,
Buffer
);
    If (EFI_ERROR (Status)) {
    return EFI_UNSUPPORTED;
}

//
  // Reset the system
//
  RT->ResetSystem (EfiResetCold, 0, 0, NULL);
} else {
  //
  // Clear the recovery record
//
  Status = RecoveryLib->WriteResetRecord (
                               RecoveryLib,
                               0,
                               Buffer
);
    If (EFI_ERROR (Status)) {
    return EFI_UNSUPPORTED;
}

//
// Get the variable
  //
  DataSize = 10;

  Status = RT->GetVariable (
                   L”Test Variable”,
                   &gEfiTestVariableGuid,
                   &Attributes,
                   &DataSize,
                   Data
);
  //
// Here ship the validation code
  //
```

```
}

return EFI_SUCCESS;
}
```

A.3 Test Logging Services

The test logging services are provided by an EFI Test Logging Library Protocol. This protocol is responsible for reporting test messages. It provide more control over output than the service RecordMessage() in the Standard Test Library Protocol..

Protocol Interface Structure

```
typedef struct _EFI_TEST_LOGGING_LIBRARY_PROTOCOL {
  UINT64                                    LibraryRevision;
  CHAR16                                    *Name;
  CHAR16                                    *Description;
  EFI_TLL_LINE                              Line;
  EFI_TLL_ENTER_FUNCTION                    EnterFunction;
  EFI_TLL_EXIT_FUNCTION                     ExitFunction;
  EFI_TLL_DUMP_MASK                         DumpMask;
  EFI_TLL_DUMP_BUF                          DumpBuf;
} EFI_TEST_LOGGING_LIBRARY_PROTOCOL;

typedef
EFI_STATUS
(EFIAPI *EFI_TLL_LINE) (
  IN  EFI_TEST_LOGGING_LIBRARY_PROTOCOL        *This,
  IN  EFI_VERBOSE_LEVEL                        VerboseLevel,
  IN  UINT32                                   Length
  )
```

```
typedef
EFI_STATUS
(EFIAPI *EFI_TLL_ENTER_FUNCTION) (
  IN  EFI_TEST_LOGGING_LIBRARY_PROTOCOL         *This,
  IN  CHAR16                                    *FunctionName,
  IN  CHAR16                                    *Format,
  ...
  )

typedef
EFI_STATUS
(EFIAPI *EFI_TLL_EXIT_FUNCTION) (
  IN  EFI_TEST_LOGGING_LIBRARY_PROTOCOL         *This,
  IN  CHAR16                                    *FunctionName,
  IN  CHAR16                                    *Format,
  ...
  )

typedef
EFI_STATUS
(EFIAPI *EFI_TLL_DUMP_MASK) (
  IN  EFI_TEST_LOGGING_LIBRARY_PROTOCOL         *This,
  IN  EFI_VERBOSE_LEVEL                         VerboseLevel,
  IN  UINT32                                    BitMask,
  IN  UINT32                                     Length
  )

typedef
EFI_STATUS
(EFIAPI *EFI_TLL_DUMP_BUF) (
  IN  EFI_TEST_LOGGING_LIBRARY_PROTOCOL           *This,
  IN  EFI_VERBOSE_LEVEL                           VerboseLevel,
  IN  CHAR16                                      *Buffer,
  IN  UINT32                                      Length,
  IN  UINT32                                      Flags
  )
```

Usage Model

The function below invokes the five test logging services. But in SCT, a filter is created for all recorded messages. It is set to **EFI_VERBOSE_LEVEL_DEFAULT**. So the output of two special services EnterFunction() and ExitFunction() will not be recorded.

```
{
  EFI_STATUS                                    Status;
  EFI_TEST_LOGGING_LIBRARY_PROTOCOL             *LoggingLib;

  //
  // Locate the test support libraries
  //
  Status = BS->LocateProtocol (
&gEfiTestLoggingLibraryGuid,
NULL,
&LoggingLib
);
  if (EFI_ERROR (Status)) {
    return EFI_UNSUPPORTED;
}
```

```
//
// Insert a line “---” into a log file
//
LoggingLib->Line (
                LoggingLib,
                EFI_VERBOSE_LEVEL_DEFAULT,
                80
);
  //
// Enter a function. It is put at the entry point of a function.
//
LoggingLib->EnterFunction (
                LoggingLib,
                L”Function Name”,
L”%a:%d:put the detail information here”,
__FILE__,
__LINE__
);
  //
// Exit a function. It is put at the exit point of a function.
//
LoggingLib->ExitFunction (
                LoggingLib,
                L”Function Name”,
L”%a:%d:put the detail information here”,
__FILE__,
__LINE__
);

//
// Dump a bitmap buffer
//
LoggingLib->DumpMask (
                LoggingLib,
                EFI_VERBOSE_LEVEL_DEFAULT,
                0x00F0FF00,
                32
);
//
// Dump a buffer
//
LoggingLib->DumpMask (
                LoggingLib,
                EFI_VERBOSE_LEVEL_DEFAULT,
```

```
                L"Test Data",
                Sizeof(L"Test Data"),
                EFI_DUMP_HEX | EFI_DUMP_ASCII
);

return EFI_SUCCESS;
}
```

A.4 Test Profile Services

The test profile services are provided by an EFI Test Profile Library Protocol. This protocol is responsible for handling the file in INI format.

Protocol Interface Structure

```
typedef struct _EFI_TEST_PROFILE_LIBRARY_PROTOCOL {
  UINT64                                LibraryRevision;
  CHAR16                                *Name;
  CHAR16                                *Description;
  EFI_TPL_INI_OPEN                      EfiIniOpen;
  EFI_TPL_INI_CREATE                    EfiIniCreate;
  EFI_TPL_INI_CLOSE                     EfiIniClose;
  EFI_TPL_GET_SYSTEM_DEVICE_PATH        EfiGetSystemDevicePath;
  EFI_TPL_SET_SYSTEM_DEVICE_PATH        EfiSetSystemDevicePath;
} EFI_TEST_PROFILE_LIBRARY_PROTOCOL;

typedef
EFI_STATUS
(EFIAPI *EFI_TPL_INI_OPEN) (
  IN  EFI_TEST_PROFILE_LIBRARY_PROTOCOL     *This,
  IN  EFI_DEVICE_PATH_PROTOCOL              *DevicePath,
  IN  CHAR16                                *FileName,
  OUT EFI_INI_FILE                          **FileHandle
  )

typedef
EFI_STATUS
(EFIAPI *EFI_TPL_INI_CREATE) (
  IN  EFI_TEST_PROFILE_LIBRARY_PROTOCOL     *This,
  IN  EFI_DEVICE_PATH_PROTOCOL              *DevicePath,
  IN  CHAR16                                *FileName,
  OUT EFI_INI_FILE                          **FileHandle
  )

typedef
EFI_STATUS
(EFIAPI *EFI_TPL_INI_CLOSE) (
  IN  EFI_TEST_PROFILE_LIBRARY_PROTOCOL     *This,
  IN  EFI_INI_FILE                          *FileHandle
  )

typedef
EFI_STATUS
(EFIAPI *EFI_TPL_GET_SYSTEM_DEVICE_PATH) (
  IN  EFI_TEST_PROFILE_LIBRARY_PROTOCOL      *This,
  OUT EFI_DEVICE_PATH_PROTOCOL               **DevicePath,
  OUT CHAR16                                 **FilePath
  )
```

```
typedef
EFI_STATUS
(EFIAPI *EFI_TPL_SET_SYSTEM_DEVICE_PATH) (
  IN  EFI_TEST_PROFILE_LIBRARY_PROTOCOL    *This,
  IN  EFI_DEVICE_PATH_PROTOCOL             *DevicePath,
  IN  CHAR16                               *FilePath
  )

typedef struct _EFI_INI_FILE {
  UINT64                                   Revision;
  EFI_INI_GETSTRING                        GetString;
  EFI_INI_SETSTRING                        SetString;
  EFI_INI_RMSECTION                        RmSection;
  EFI_INI_GETSTRING_BYORDER                GetStringByOrder;
  EFI_INI_SETSTRING_BYORDER                SetStringByOrder;
  EFI_INI_RMSECTION_BYORDER                RmSectionByOrder;
  EFI_INI_GETORDERNUM                      GetOrderNum;
  EFI_INI_FLUSH                            Flush;
} EFI_INI_FILE, *EFI_INI_FILE_HANDLE;

typedef
EFI_STATUS
(EFIAPI *EFI_INI_FLUSH) (
  IN  EFI_INI_FILE                         *This
  )

typedef
EFI_STATUS
(EFIAPI *EFI_INI_GETSTRING) (
  IN     EFI_INI_FILE                    *This,
  IN     CHAR16                          *Section,
  IN     CHAR16                          *Entry,
     OUT CHAR16                          *String,
  IN OUT UINT32                          *MaxLength
  )

typedef
EFI_STATUS
(EFIAPI *EFI_INI_SETSTRING) (
  IN  EFI_INI_FILE                         *This,
  IN  CHAR16                               *Section,
  IN  CHAR16                               *Entry,
  IN  CHAR16                               *String
```

```
  )

typedef
EFI_STATUS
(EFIAPI *EFI_INI_RMSECTION) (
  IN  EFI_INI_FILE                        *This,
  IN  CHAR16                              *Section
  )

typedef
EFI_STATUS
(EFIAPI *EFI_INI_GETSTRING_BYORDER) (
  IN     EFI_INI_FILE                     *This,
  IN     UINT32                           Order,
  IN     CHAR16                           *Section,
  IN     CHAR16                           *Entry,
     OUT CHAR16                           *String,
  IN OUT UINT32                           *MaxLength
  )

typedef
EFI_STATUS
(EFIAPI *EFI_INI_SETSTRING_BYORDER) (
  IN  EFI_INI_FILE                        *This,
  IN  UINT32                              Order,
  IN  CHAR16                              *Section,
  IN  CHAR16                              *Entry,
  IN  CHAR16                              *String
  )

typedef
EFI_STATUS
(EFIAPI *EFI_INI_RMSECTION_BYORDER) (
  IN  EFI_INI_FILE                        *This,
  IN  UINT32                              Order,
  IN  CHAR16                              *Section
  )

typedef
EFI_STATUS
(EFIAPI *EFI_INI_GETORDERNUM) (
  IN  EFI_INI_FILE                       *This,
  IN  CHAR16                             *Section,
  OUT UINT32                             *OrderNum
```

```
)
```

Usage Model

Assume an INI file, named as test.ini, is put in the root directory of UEFI SCT folder

```
[Test Data1]
Key1 = Value1
Key2 = Value2

[Test Data2]
Key1 = Value3
Key2 = Value4
```

The function below reads the data in the INI file.

```
{
  EFI_STATUS                                Status;
  EFI_TEST_PROFILE_LIBRARY_PROTOCOL         *ProfileLib;
  EFI_DEVICE_PATH                           *SystemDevicePath;
  CHAR16                                    *SystemFilePath;
  CHAR16                                    *FileName;
  EFI_INI_FILE_HANDLE                       IniFileHandle;
  UINT32                                    OrderNum;
  UINT32                                    ValueSize;
  CHAR16                                    Value[20];

  //
  // Locate the test support libraries
  //
  Status = BS->LocateProtocol (
&gEfiTestProfileLibraryGuid,
NULL,
&ProfileLib
);
  if (EFI_ERROR (Status)) {
    return EFI_UNSUPPORTED;
}

  //
// Get the system device path and file path
//
Status = ProfileLib->EfiGetSystemDevicePath (
                            ProfileLib,
                            &SystemDevicePath,
                            &SystemFilePath
);
  if (EFI_ERROR (Status)) {
    return EFI_UNSUPPORTED;
}

  //
// Create the path of the INI file
//
  FileName = PoolPrint (L"%s\\%s", SystemFilePath, L"test.ini");
  If (FileName == NULL) {
  BS->FreePool (SystemDevicePath);
  BS->FreePool (SystemFilePath);
  return EFI_UNSUPPORTED;
}
```

```
BS->FreePool (SystemFilePath);

//
// Open the INI file
//
Status = ProfileLib->EfiIniOpen (
                            ProfileLib,
                            SystemDevicePath,
                            FileName,
                            &IniFileHandle
);
  If (EFI_ERROR (Status)) {
  BS->FreePool (SystemDevicePath);
  BS->FreePool (FileName);
  return EFI_UNSUPPORTED;
}

BS->FreePool (SystemDevicePath);
BS->FreePool (FileName);

//
// Get the number of required sections
//
Status = IniFileHandle->GetOrderNum (
                                IniFileHandle,
                                L"Test Data",
                                &OrderNum
);
  if (EFI_ERROR (Status) || (OrderNum < 2)) {
ProfileLib->EfiIniClose (ProfileLib, IniFileHandle);
return EFI_UNSUPPORTED;
}

//
// Read the Key1's value in the second section (start from 0)
//
ValueSize = 20;
Status = IniFileHandle->GetStringByOrder (
                                IniFileHandle,
                                (UINT32)1,
                                L"Test Data",
                                L"Key1",
                                Value,
```

```
                                &ValueSize
);
  if (EFI_ERROR (Status)) {
ProfileLib->EfiIniClose (ProfileLib, IniFileHandle);
Return EFI_UNSUPPORTED;
}

Print (L”Key1’s value in the second section: %s”, Value);

//
// Close the INI file
//
ProfileLib->EfiIniClose (ProfileLib, IniFileHandle);

return EFI_SUCCESS;
}
```