Self Certification Test (SCT) II
Case Specification

June 2017

Revision History

Version	Revision Notes	Release Date
2.1	Initial Release	May 2009
2.3	Add chapter for Vlan and EAP. Additional materials to EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL test, also DHCPv6, TCP6, IPv6, IP6Config, IPsecConfig, UDPv6 and MTFTPv6	June 2010
2.3	Mantis 618 content integration 643 Add test case description of Firmware Management Protocol 646 ATA Pass Thru Protocol 699 SetInfo and StringToImage Integration 672 InstallSCT application need to support on NT32 673 New feature request for Verbose function in SCT	Jan. 2011
2.3.1 C	Mantis 643 Firmware Management Protocol Test Case 710 Match SCT Case Specification 2.3 to UEFI spec 2.3 827 Char issues in Guid format corrected and missed Index appended 832 Some index numbers in spec corrected 835 Guid Format corrections 841 Corrected some duplicated Index/Guid 843 Case Spec refreshed to include new items of UEFI 2.3.1 Spec	Oct. 2011
2.3.1 C	Mantis 939 Update to align with UEFI Spec 2.3.1 Errata C	Aug. 2012
2.4 B	Mantis 1295 Update to align with UEFI Spec 2.4 Errata B	Dec. 2014
2.5 A	Mantis 1733 Update to align with UEFI Spec 2.5 Errata A	Jan. 2017
2.6 A	Mantis 1807 Update to align with UEFI Spec 2.6 Errata A	June 2017

Contents

Tables

1 Introduction

1.1 Overview

This document provides detailed information for each assertion in the UEFI SCT fundamental service and protocol tests. This document can be used as a reference on case assertion for UEFI SCT users.

Reference Documents

- *UEFI Specification* -- indicates current and past UEFI specifications, unless specific versions are noted
- *UEFI SCT Getting Started*
- *UEFI SCT User Guide*
-

5.26.2.7.71	0x732738e8, 0x1ff1, 0x4f3a, 0xa0,0xc8, 0x38,0x81, 0x1d,0x15, 0x92,0x83	**EFI_MTFTP4_PROTOCOL.ReadFile()** - **ReadFile()** must return **EFI_PROTOCOL_UNREACHABLE** when receive an ICMP protocol unreachable packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure Host side to send back an ICMP protocol unreachable packet and the return status should be **EFI_PROTOCOL_UNREACHABLE**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

5.26.2.7.72	0xd1c4e1e8, 0x1099, 0x4646, 0xb7,0xc9, 0x64,0x7e, 0x65,0xc3, 0x82,0x30	**EFI_MTFTP4_PROTOCOL.ReadFile()** - **ReadFile()** must return **EFI_PORT_UNREACHABLE** when receive an ICMP port unreachable packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure Host side to send back an ICMP port unreachable packet and the return status should be **EFI_PORT_UNREACHABLE**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

1.2 System Hang

If the system hangs in any of tests, the UEFI SCT framework records a failure assertion in the test report and skips this test after a system restart.

Number	GUID	Assertion	Test Description
5.20.1.1.1	0xde687a18, 0x0bbd, 0x4396, 0x85, 0x09, 0x49, 0x8f, 0xf2, 0x32, 0x34, 0xf1	System hangs or stops..	The name of the test which causes the system hang can be found in the test report.

2 EFI Compliance Test

2.1 EFI Requirements Test

Reference Document:

UEFI Specification, Requirements Section.

Configuration

Configuration is a checkpoint in the EFI Requirements Test. If the you need to check the platform-specific protocols, the related profile needs to be updated.

For the correct formatting of profiles, refer to Appendix section A.1, EFI Requirements Test Profile.

2.1.1 Required Elements

Number	GUID	Assertion	Test Description
5.22.1.1.1	0xf6a871e3, 0xef8a, 0x420f, 0x82, 0x01, 0x35, 0xb6, 0x1c, 0xe2, 0xe8, 0xdb	EFI-Compliant - EFI System Table must be implemented.	1. The *Signature* of EFI System Table should be 0x5453595320494249. 2. The *Revision* of EFI System Table should be equal to or larger than 0x00020000. 3. The *Reserved* field in EFI System Table should be 0. 4. The *RuntimeServices* and *BootServices* pointers of EFI System Table should not be **NULL**. 5. The *CRC32* of EFI System Table must be correct.
5.22.1.1.2	0xaddab6ed, 0x5a17, 0x4327, 0x8f, 0xb1, 0x72, 0x93, 0x3d, 0x1a, 0x7b, 0xba	EFI-Compliant - EFI Boot Services Table must be implemented.	1. The *Signature* of EFI Boot Services Table should be 0x56524553544f4f42. 2. The *Revision* of EFI Boot Services Table should be equal to or larger than 0x00020000. 3. The *Reserved* field in EFI Boot Services Table should be 0. 4. No function pointers in EFI Boot Services Table should be **NULL**.
5.22.1.1.3	0x13a20958, 0xc860, 0x452f, 0xb9, 0xa2, 0xe6, 0xd9, 0x96, 0x41, 0x92, 0x24	EFI-Compliant - EFI Runtime Services Table must be implemented.	1. The *Signature* of EFI Runtime Services Table should be 0x56524553544e5552. 2. The *Revision* of EFI Runtime Services Table should be equal to or larger than 0x00020000. 3. The *Reserved* field in EFI Runtime Services Table should be 0. 4. No function pointers in EFI Runtime Services Table should be **NULL**.
5.22.1.1.4	0xa82f8d56, 0x1476, 0x41f1, 0xba, 0xc4, 0x97, 0x59, 0x79, 0x9f, 0x97, 0xf3	EFI-Compliant – **EFI_LOADED_IMAGE_PROTOCOL** must exist.	1. Call **LocateProtocol()** to find the **LOADED_IMAGE_PROTOCOL**. The return status should be **EFI_SUCCESS**.
5.22.1.1.5	0xf61f0f0a, 0x64fe, 0x40a6, 0x9d, 0x7c, 0x07, 0x46, 0xa2, 0x30, 0x24, 0x5f	EFI-Compliant – **EFI_DEVICE_PATH_PROTOCOL** must exist.	1. Call **LocateProtocol()** to find the **DEVICE_PATH_PROTOCOL**. The return status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.22.1.1.6	0x02c017d7, 0x1557, 0x47d9, 0xbc, 0xe9, 0x87, 0x18, 0x2d, 0x07, 0x91, 0x0c	EFI-Compliant – EFI_DECOMPRESS_PROTOCOL must exist.	1. Call LocateProtocol() to find the DECOMPRESS_PROTOCOL. The return status should be EFI_SUCCESS. 2. No function pointers in DECOMPRESS_PROTOCOL should be NULL.
5.22.1.1.7	0x3a07dc1b, 0x53d1, 0x4fac, 0x88, 0xaf, 0xc7, 0x25, 0x79, 0xeb, 0x07, 0xf2	UEFI-Compliant- EFI_DEVICE_PATH_UTILITIES_PROTOCOL must exist	1. Call LocateProtocol() to find the EFI_DEVICE_PATH_UTILITIES_PROTOCOL, the return status should be EFI_SUCCESS 2. No function pointer in Device Path Utility protocol should be NULL
5.22.1.1.8	0xf6334f9b, 0xb930, 0x4adb, 0xa5, 0x3b, 0x76, 0xfa, 0x7b, 0x4c, 0x27, 0x62	UEFI-Compliant The EFI_GLOBAL_VARIABLE guid should be used by the globally defined variables only, and the attributes of the variables should be same with the definition in the Specification.	1. Locate all variables with EFI_GLOBAL_VARIABLE guid, check the variable name is in the pre-defined globally variable list. 2. Check the variable attribute.

2.1.2 Platform-Specific Elements

Number	GUID	Assertion	Test Description
5.22.1.2.1	0x8f7556c2, 0x4665, 0x4353, 0xa3, 0xaf, 0x9c, 0x00, 0x5a, 0x1e, 0x63, 0xe1	EFI-Compliant - `EFI_SIMPLE_TEXT_INPUT_PROTOCOL, EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL` and `EFI_SIMPLE_TEXT_OUT_PROTOCOL` must be implemented if a platform includes console devices.	1. Call `LocateProtocol()` to find the `EFI_SIMPLE_TEXT_INPUT_PROTOCOL`. 2. Call `LocateProtocol()` to find the `EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL`. 3. Call `LocateProtocol()` to find the `EFI_SIMPLE_TEXT_OUT_PROTOCOL`. 4. If the INI file indicates that the platform includes console devices, the return status in steps 1, 2 and 3 should be `EFI_SUCCESS`. If not, the return status in steps 1, 2 and 3 should be `EFI_SUCCESS` or `EFI_ERROR`.
5.22.1.2.2	0x72ba0e86, 0x58e5, 0x48dd, 0x85, 0x29, 0x88, 0xc6, 0x83, 0x83, 0x11, 0x8d	UEFI-Compliant - `EFI_GRAPHICS_OUTPUT_PROTOCOL`, `EFI_EDID_ACTIVE_PROTOCOL`, `EFI_EDID_DISCOVERED_PROTOCOL` must be implemented if a platform includes graphical console devices.	1. Call `LocateProtocol()` to find the `EFI_GRAPHICS_OUTPUT_PROTOCOL` 2. Call `LocateProtocol()` to find the `EFI_EDID_ACTIVE_PROTOCOL. ,` 3. Call `LocateProtocol()` to find the `EFI_EDID_DISCOVERED_PROTOCOL.` 4. If the INI file indicates that the platform includes graphical console devices, the return status in all steps 1, 2 and 3 should be `EFI_SUCCESS`. 5. If the INI file doesn't indicate that the platform includes graphical console devices, the return status in all steps 1, 2 and 3 could be either `EFI_SUCCESS` or `EFI_ERROR`.
5.22.1.2.3	0x18670db1, 0x89fb, 0x4de4, 0xb1, 0x0f, 0x89, 0x8e, 0x04, 0x7d, 0x95, 0x2a	UEFI-Compliant – `EFI_SIMPLE_POINTER_PROTOCOL` must be implemented if a platform includes a pointer device as part of its console support.	1. Call `LocateProtocol()` to find the `EFI_SIMPLE_POINTER_PROTOCOL`. 2. If the INI file indicates that the platform includes a pointer device, the return status in step 1 should be `EFI_SUCCESS`. 3. If the INI file doesn't indicate that the platform includes a pointer device, the return status in step 1 could be either `EFI_SUCCESS` or `EFI_ERROR`.

Number	GUID	Assertion	Test Description
5.22.1.2.4	0xbf38a3fd, 0x58ac, 0x419a, 0xab, 0xc2, 0xc6, 0x0b, 0xae, 0x9c, 0xfe, 0x67	UEFI-Compliant – EFI_BLOCK_IO_PROTOCOL, EFI_DISK_IO_PROTOCOL, EFI_SIMPLE_FILE_SYSTEM; EFI_UNICODE_COLLATION_PROTOCOL must be implemented if a platform supports booting from a disk.	1. Call LocateProtocol() to find the EFI_BLOCK_IO_PROTOCOL protocol. 2. Call LocateProtocol() to find the EFI_DISK_IO_PROTOCOL. 3. Call LocateProtocol() to find the EFI_SIMPLE_FILE_SYSTEM_PROTOCOL. 4. Call LocateProtocol() to find the EFI_UNICODE_COLLATION_PROTOCOL. 5. If the INI file indicates that the platform supports booting from a disk, the return status in steps 1, 2, 3, and 4 all should be EFI_SUCCESS. 6. If the INI file doesn't indicate that the platform supports booting from a disk, the return status in steps 1, 2, 3, and 4 all should be EFI_SUCCESS or EFI_ERROR.
5.22.1.2.5	0x98551ae7, 0x5020, 0x4ddd, 0x86, 0x1a, 0xcf, 0xff, 0xb4, 0xd6, 0x03, 0x82	UEFI-Compliant – EFI_PXE_BASE_CODE_PROTOCOL must be implemented if a platform supports TFTP-based booting from a network device. And platform must be prepared to produce this protocol on any of EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL (UNDI), EFI_SIMPLE_NETWORK_PROTOCOL, or the EFI_MANAGED_NETWORK_PROTOCOL. If platform supports validating the image received from network device, SetupMode equal zero.	1. Call LocateProtocol() to find the EFI_PXE_BASE_CODE_PROTOCOL. 2. Call LocateProtocol() to find the EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL, EFI_SIMPLE_NETWORK_PROTOCOL, EFI_MANAGED_NETWORK_PROTOCOL. 3. If the INI file indicates that the platform supports TFTP-based booting from a network device, the return status in step 1 should be EFI_SUCCESS. And one of the step 2 should be EFI_SUCCESS at least. 4. If the INI file doesn't indicate that the platform supports TFTP-based booting from a network device, the return status in both step 1 and step 2 step should be EFI_SUCCESS or EFI_ERROR. 5. If the INI file indicates that the platform supports validating the image received from a network device, SetupMode equal zero.
5.22.1.2.6	0x517bcbeb, 0x4982, 0x4a7e, 0x85, 0x51, 0xca, 0x84, 0x7d, 0xdc, 0x21, 0xc2	UEFI-Compliant – EFI_SERIAL_IO_PROTOCOL must be implemented if a platform includes a byte stream device.	1. Call LocateProtocol() to find the EFI_SERIAL_IO_PROTOCOL. 2. If the INI file indicates that the platform includes a byte-stream device, the return status in step 1 should be EFI_SUCCESS. 3. If the INI file doesn't indicate that the platform includes a byte-stream device, the return status in step 1 step could be either EFI_SUCCESS or EFI_ERROR.

Number	GUID	Assertion	Test Description
5.22.1.2.7	0x213a75c9, 0x7f3d, 0x42db, 0xb3, 0x2a, 0x02, 0xdb, 0xd6, 0x98, 0x31, 0x9d	UEFI-Compliant – EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL and EFI_PCI_IO_PROTOCOL must be implemented if a platform includes PCI bus support.	1. Call LocateProtocol() to find the EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL. 2. Call LocateProtocol() to find the EFI_PCI_IO_PROTOCOL. 3. If the INI file indicates that the platform includes PCI bus support, the return status in both steps 1 and 2 should be EFI_SUCCESS. 4. If the INI file doesn't indicate that the platform includes PCI bus support, the return status in both steps 1 and 2 steps could be EFI_SUCCESS or EFI_ERROR.
5.22.1.2.8	0x0ccd5843, 0x5bb5, 0x4fc2, 0xa7, 0x32, 0xdb, 0x17, 0xc4, 0x14, 0xa4, 0x3d	UEFI-Compliant – EFI_USB_HC2_PROTOCOL and EFI_USB_IO_PROTOCOL must be implemented if a platform includes USB bus support.	1. Call LocateProtocol() to find the EFI_USB_HC2_PROTOCOL. 2. Call LocateProtocol() to find the EFI_USB_IO_PROTOCOL. 3. If INI file indicates the platform includes USB bus support, the return status in 1 and 2 steps should be both EFI_SUCCESS. 4. If INI file doesn't indicate the platform includes USB bus support, the return status in 1 and 2 steps should be both EFI_SUCCESS or both EFI_ERROR.
5.22.1.2.9	0x2b83418f, 0xe7fb, 0x4528, 0xb6, 0xff, 0xc9, 0xd4, 0x87, 0xae, 0x2e, 0xff	UEFI-Compliant – EFI_EXT_SCSI_PASS_THRU_PROTOCOL must be implemented if a platform includes an I/O system that uses SCSI command packets.	1. Call LocateProtocol() to find the EFI_EXT_SCSI_PASS_THRU_PROTOCOL. 2. If INI file indicates the platform includes an I/O system that uses SCSI command packets, the return status in 1 step should be EFI_SUCCESS. 3. If INI file doesn't indicate the platform includes an I/O system that uses SCSI command packets, the return status in 1 step could be EFI_SUCCESS or EFI_ERROR.

Number	GUID	Assertion	Test Description
5.22.1.2.10	0x3ee22696, 0x0875, 0x46f4, 0x88, 0x84, 0xba, 0x12, 0x4c, 0x7e, 0xaf, 0xf0	UEFI-Compliant – `EFI_DEBUG_SUPPORT_PROTOCOL` and `EFI_DEBUG_PORT_PROTOCOL` must be implemented if a platform supports debugging capabilities.	1. Call `LocateProtocol()` to find the `EFI_DEBUG_SUPPORT_PROTOCOL`. 2. Call `LocateProtocol()` to find the `EFI_DEBUG_PORT_PROTOCOL`. 3. If INI file indicates the platform supports debugging capabilities, the return status in 1 and 2 steps should be both `EFI_SUCCESS`. 4. If INI file doesn't indicate the platform supports debugging capabilities, the return status in 1 and 2 steps should be both `EFI_SUCCESS` or both `EFI_ERROR`.
5.22.1.2.11	0x329027ce, 0x406e, 0x48c8, 0x8a, 0xc1, 0xa0, 0x2c, 0x1a, 0x6e, 0x39, 0x83	UEFI-Compliant – `EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL` must be implemented if a platform includes the ability to override the default driver.	1. Call `LocateProtocol()` to find the `EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL`. 2. If INI file indicates the platform includes the ability to override the default driver, the return status in 1 step should be `EFI_SUCCESS`. 3. If INI file doesn't indicate the platform includes the ability to override the default driver, the return status in 1 step could be `EFI_SUCCESS` or `EFI_ERROR`.

Number	GUID	Assertion	Test Description
5.22.1.2.12	0x76a6a1b0, 0x8c53, 0x407d, 0x84, 0x86, 0x9a, 0x6e, 0x63, 0x32, 0xd3, 0xce	UEFI-Compliant – **EFI_MANAGED_NETWORK_PROTOCOL, EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL, EFI_ARP_PROTOCOL, EFI_ARP_SERVICE_BINDING_PROTOCOL, EFI_DHCP4_PROTOCOL, EFI_DHCP4_SERVICE_BINDING_PROTOCOL, EFI_TCP4_PROTOCOL, EFI_TCP4_SERVICE_BINDING_PROTOCOL, EFI_IP4_PROTOCOL, EFI_IP4_SERVICE_BINDING_PROTOCOL, EFI_IP4_CONFIG2_PROTOCOL, EFI_UDP4_PROTOCOL, EFI_UDP4_SERVICE_BINDING_PROTOCOL, EFI_MTFTP4_PROTOCOL**, and **EFI_MTFTP4_SERVICE_BINIING_PROTOCOL** are required for general network application	1. Call **LocateProtocol()** to find the **EFI_MANAGED_NETWORK_PROTOCOL**, **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL, EFI_ARP_PROTOCOL, EFI_ARP_SERVICE_BINDING_PROTOCOL, EFI_DHCP4_PROTOCOL, EFI_DHCP4_SERVICE_BINDING_PROTOCOL, EFI_TCP4_PROTOCOL, EFI_TCP4_SERVICE_BINDING_PROTOCOL, EFI_IP4_PROTOCOL, EFI_IP4_SERVICE_BINDING_PROTOCOL, EFI_IP4_CONFIG2_PROTOCOL, EFI_UDP4_PROTOCOL, EFI_UDP4_SERVICE_BINDING_PROTOCOL, EFI_MTFTP4_PROTOCOL**, and **EFI_MTFTP4_SERVICE_BINIING_PROTOCOL** 2. If INI file indicates the platform includes the ability to general network application, the return status for locating all protocols described in step 1 should be **EFI_SUCCESS** 3. If INI file doesn't indicate the platform includes the ability for general network application, the return status for locating all protocols described in step 1 could

Number	GUID	Assertion	Test Description
5.22.1.2.13	0x28c068f2, 0xf398, 0x488a, 0xb0, 0x59, 0x53, 0x4e, 0x98, 0x2d, 0x9c, 0x85	UEFI-Compliant – EFI_SCSI IO_PROTOCOL, EFI_Block IO_PROTOCOL and EFI_EXT_SCSI_PASS_THRU_PROTOCOL must be implemented if a platform supports booting from a SCSI peripheral device.	1. Call LocateProtocol() to find the EFI_SCSI IO_PROTOCOL. 2. Call LocateProtocol() to find the EFI_Block IO_PROTOCOL protocol. 3. Call LocateProtocol() to find the EFI_EXT_SCSI_PASS_THRU_PROTOCOL. 4. If the INI file indicates that the platform supports booting from a network device, the return status in all steps 1, 2 and 3 should be EFI_SUCCESS. 5. If the INI file doesn't indicate that the platform supports booting from a network device, the return status in all steps 1, 2 and 3 should be EFI_SUCCESS or EFI_ERROR.
5.22.1.2.14	0x6b7077a6, 0x4b13, 0x4e13, 0x9b, 0x1f, 0x0c, 0x4b, 0x3a, 0x86, 0x69, 0xe2	UEFI-Compliant – EFI_ISCSI_INITIATOR_NAME_PROTOCOL and EFI_AUTHENTICATION_INFO_PROTOCOL must be implemented if a platform supports booting from a ISCSI peripheral device.	1. Call LocateProtocol() to find the EFI_ISCSI_INITIATOR_NAME_PROTOCOL and EFI_AUTHENTICATION_INFO_PROTOCOL. 2. If the INI file indicates that the platform supports booting from a iSCSI peripheral, the return status in both steps 1 should be EFI_SUCCESS. 3. If the INI file doesn't indicate that the platform supports booting from iSCSI peripheral, the return status in steps 1 should be EFI_SUCCESS or EFI_ERROR.
5.22.1.2.15	0x4c82eb2d, 0xc785, 0x410c, 0x95, 0xd1, 0xae, 0x27, 0x12, 0x21, 0x44, 0xc8	UEFI Compliant –UEFI V6 General Network Driver Dhcp6SB, Tcp6SB, Ip6SB, Udp6SB, Ip6Config, Vlan must exist if a platform supports V6 network stack	1. Call LocateProtocol() to find the V6 network stack. 2. If the INI file indicates that the platform supports v6 stack, the return status in step 1 should be EFI_SUCCESS. 3. If the INI file doesn't indicate that the platform supports v6 network stack, the return status in steps1 should be EFI_SUCCESS or EFI_ERROR.

Number	GUID	Assertion	Test Description
5.22.1.2.16	0x1d0a2f2a, 0x924, 0x4b8c, 0x9f, 0xc7, 0xb1, 0x85, 0xcc, 0x22, 0xe1, 0x18	UEFI Compliant –UEFI EBC interpreter must exist if a platform supports EBC image	1. Call **LocateProtocol()** to find the **EFI_EBC_PROTOCOL**. 2. If the INI file indicates that the platform supports EBC image, the return status in step 1 should be **EFI_SUCCESS**. 3. If the INI file doesn't indicate that the platform supports EBC image, the return status in step 1 should be **EFI_SUCCESS** or **EFI_ERROR**.
5.22.1.2.17	0xb7cd2d76, 0xea43, 0x4013, 0xb7, 0xd1, 0x59, 0xeb, 0x2e, 0xc9, 0xbf, 0x1b	UEFI Compliant –UEFI **HiiDatabase, HiiString, HiiConfigRouting, HiiConfigAccess** must be existed if the platform supports HII. If it supports bitmapped fonts, then **HiiFont** must exist also.	1. Call **LocateProtocol()** to find **HiiDatabase, HiiString, HiiConfigRouting, HiiConfigAccess**. 2. If the INI file indicates that the platform supports HII all return statuses in step 1 should be **EFI_SUCCESS**. 3. If the INI file doesn't indicate that the platform supports HII, the return status in step1 should be **EFI_SUCCESS** or **EFI_ERROR**. 4. If step 2 is true, and the INI file indicates the platform support bitmapped font, call **LocateProtocol()** to find **HiiFont,** and the return status should **EFI_SUCCESS**.
5.22.1.2.18	0x5aea7246, 0xbcf9, 0x4ba4, 0x81, 0xd2, 0x83, 0x2c, 0x98, 0x41, 0x46, 0xf3	UEFI-Compliant – **EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL** must be implemented if a platform includes an NVM Express controller	1. Call **LocateProtocol()** to find the **EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL**. 2. If the INI file indicates that the platform includes an NVM Express controller, the return status in steps 1 should be **EFI_SUCCESS**. 3. If the INI file doesn't indicate that the platform includes an NVM Express controller, the return status in steps 1 should be **EFI_SUCCESS** or **EFI_ERROR**.

Number	GUID	Assertion	Test Description
5.22.1.2.19	0x5cb0cdb5, 0xac80, 0x4983, 0xb7, 0x10, 0x4b, 0xb, 0xf0, 0x19, 0x15, 0x63	UEFI Compliant – EFI_BLOCK_IO_PROTOCOL must be existed if the platform supports booting from a block-oriented NVM Express controller. EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL may be required.	1. Call LocateProtocol() to find EFI_BLOCK_IO_PROTOCOL. 2. If the INI file indicates that the platform supports booting from a block-oriented NVM Express controller, all return statuses in step 1 should be EFI_SUCCESS. 3. If the INI file doesn't indicate that the platform supports booting from a block-oriented NVM Express controller, the return status in step1 should be EFI_SUCCESS or EFI_ERROR. 4. If step 2 is true, and the INI file indicates the platform support EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL, call LocateProtocol() to find it, and the return status should be EFI_SUCCESS.
5.22.1.2.20	0x563f654f, 0xaba8, 0x4539, 0x80, 0x4b, 0x50, 0x63, 0x5, 0x7, 0x26, 0x23	UEFI-Compliant – EFI_ATA_PASS_THRU_PROTOCOL must be implemented if a platform includes an I/O subsystem that utilizes ATA command packets.	1. Call LocateProtocol() to find the EFI_ATA_PASS_THRU_PROTOCOL. 2. If the INI file indicates that the platform includes an I/O subsystem that utilizes ATA command packets, the return status in steps 1 should be EFI_SUCCESS. 3. If the INI file doesn't indicate that the platform includes an I/O subsystem that utilizes ATA command packets, the return status in steps 1 should be EFI_SUCCESS or EFI_ERROR.
5.22.1.2.21	0x2e6d1733, 0x6d39, 0x49ab, 0xa8, 0x86, 0x1b, 0x6d, 0xe4, 0x45, 0x66, 0xa8	UEFI Compliant – EFI_DNS4_PROTOCOL, EFI_DNS4_SERVICE_BINDING_PROTOCOL must be existed if the platform supports DNS for IPv4 stack.	1. Call LocateProtocol() to find the EFI_DNS4_PROTOCOL and EFI_DNS4_SERVICE_BINDING_PROTOCOL. 2. If the INI file indicates that the platform supports DNS for IPv4 stack, the return status in steps 1 should be EFI_SUCCESS. 3. If the INI file doesn't indicate this capability, the return status in steps 1 should be EFI_SUCCESS or EFI_ERROR.

Number	GUID	Assertion	Test Description
5.22.1.2.22	0xe02a6ef3, 0x4b70, 0x40ec, 0xaa, 0x23, 0x50, 0xb7, 0xb9, 0x72, 0xb0, 0x65	UEFI Compliant – EFI_DNS6_PROTOCOL, EFI_DNS6_SERVICE_BINDING_PROTOCOL must be existed if the platform supports DNS for IPv6 stack.	1. Call LocateProtocol() to find the EFI_DNS6_PROTOCOL and EFI_DNS6_SERVICE_BINDING_PROTOCOL. 2. If the INI file indicates that the platform supports DNS for IPv6 stack, the return status in steps 1 should be EFI_SUCCESS. 3. If the INI file doesn't indicate this capability, the return status in steps 1 should be EFI_SUCCESS or EFI_ERROR.
5.22.1.2.23	0xcb6f7b77, 0xb15, 0x43f7, 0xa9, 0x5b, 0x8c, 0x7f, 0x9f, 0xd7, 0xb, 0x21	UEFI Compliant – EFI_TLS_PROTOCOL, EFI_TLS_SERVICE_BINDING_PROTOCOL, EFI_TLS_CONFIGURATION_PROTOCOL must be existed if the platform supports TLS feature.	1. Call LocateProtocol() to find the EFI_TLS_PROTOCOL, EFI_TLS_SERVICE_BINDING_PROTOCOL and EFI_TLS_CONFIGURATION_PROTOCOL. 2. If the INI file indicates that the platform supports TLS feature, the return status in steps 1 should be EFI_SUCCESS. 3. If the INI file doesn't indicate this capability, the return status in steps 1 should be EFI_SUCCESS or EFI_ERROR.
5.22.1.2.24	0x77fddb95, 0x5969, 0x4fb4, 0xa2, 0x18, 0x5c, 0xc, 0x76, 0xb, 0x5, 0x64	UEFI Compliant – EFI_HTTP_PROTOCOL, EFI_HTTP_SERVICE_BINDING_PROTOCOL, EFI_HTTP_UTILITIES_PROTOCOL must be existed if the platform includes the ability to perform a HTTP-based boot from a network device.	1. Call LocateProtocol() to find the EFI_HTTP_PROTOCOL, EFI_HTTP_SERVICE_BINDING_PROTOCOL and EFI_HTTP_UTILITIES_PROTOCOL. 2. If the INI file indicates that the platform includes the ability to perform a HTTP-based boot from a network device, the return status in steps 1 should be EFI_SUCCESS. 3. If the INI file doesn't indicate this capability, the return status in steps 1 should be EFI_SUCCESS or EFI_ERROR.

Number	GUID	Assertion	Test Description
5.22.1.2.25	0xf0dc12fa, 0x3c4b, 0x43f7, 0xa6, 0x9e, 0xa5, 0xbe, 0x6f, 0xcc, 0x90, 0xa1	UEFI Compliant – EFI_EAP_PROTOCOL, EFI_EAP_CONFIGURATION_PROTOCOL, EFI_EAP_MANAGEMENT2_PROTOCOL must be existed if the platform includes the ability to perform a wireless boot from a network device with EAP feature, and if this platform provides a standalone wireless EAP driver.	1. Call LocateProtocol() to find the EFI_EAP_PROTOCOL, EFI_EAP_CONFIGURATION_PROTOCOL and EFI_EAP_MANAGEMENT2_PROTOCOL. 2. If the INI file indicates that the platform includes the ability to perform a wireless boot from a network device with EAP feature, and if this platform provides a standalone wireless EAP driver, the return status in steps 1 should be EFI_SUCCESS. 3. If the INI file doesn't indicate this capability, the return status in steps 1 should be EFI_SUCCESS or EFI_ERROR.
5.22.1.2.26	0x87e50392, 0xf5a2, 0x42b8, 0x81, 0x12, 0x68, 0xbe, 0xc9, 0x2, 0xb9, 0xbc	UEFI Compliant – EFI_BLUETOOTH_HC_PROTOCOL, EFI_BLUETOOTH_IO_PROTOCOL, EFI_BLUETOOTH_CONFIG_PROTOCOL must be existed if the platform supports classic Bluetooth.	1. Call LocateProtocol() to find the UEFI Compliant –UEFI EFI_BLUETOOTH_HC_PROTOCOL, EFI_BLUETOOTH_IO_PROTOCOL and EFI_BLUETOOTH_CONFIG_PROTOCOL. 2. If the INI file indicates that the platform supports classic Bluetooth, the return status in steps 1 should be EFI_SUCCESS. 3. If the INI file doesn't indicate this capability, the return status in steps 1 should be EFI_SUCCESS or EFI_ERROR.

3 Services Boot Services

3.1 Event, Timer, and Task Priority Services Test

Reference Document:

UEFI Specification, Event, Timer, and Task Priority Services Section.

Table 1. Event, Timer, and Task Priority Functions

Name	Type	Description
CreateEvent()	Boot	Creates a general-purpose event structure.
CloseEvent()t	Boot	Closes and frees an event structure.
SignalEvent()	Boot	Signals an event.
WaitForEvent()	Boot	Stops execution until an event is signaled.
CheckEvent()	Boot	Checks whether an event is in the signaled state.
SetTimer()	Boot	Sets an event to be signaled at a particular time.
RaiseTPL()	Boot	Raises the task priority level.
RestoreTPL()	Boot	Restores/lowers the task priority level.
CreateEventEx()	Boot	Creates an event in a group.

3.1.1 CreateEvent()

Number	GUID	Assertion	Test Description
5.1.1.1.1	0xa2a285eb, 0x1c60, 0x42d2, 0xa3, 0x2c, 0x74, 0x61, 0x5f, 0x1f, 0x76, 0x50	`BS.CreateEvent –` `CreateEvent()` returns `EFI_INVALID_PARAMETER` with invalid event type.	1. Call `CreateEvent()` with invalid event type. The return status must be `EFI_INVALID_PARAMETER`.
5.1.1.1.2	0xbd6d4465, 0xaee3, 0x4a07, 0x84, 0x70, 0x2a, 0xba, 0x24, 0x7b, 0xc8, 0x65	`BS.CreateEvent –` `CreateEvent()` returns `EFI_INVALID_PARAMETER` with invalid notify TPL.	1. Call `CreateEvent()` with invalid notification function TPLs. The return status must be `EFI_INVALID_PARAMETER`.
5.1.1.1.3	0x587ecd61, 0x0af3, 0x442d, 0xb9, 0xa5, 0x0a, 0xdd, 0x02, 0x57, 0x5b, 0x7b	`BS.CreateEvent –` `CreateEvent()` returns `EFI_INVALID_PARAMETER` with an *Event* value of `NULL`.	1. Call `CreateEvent()` with an *Event* value of `NULL`. The return status must be `EFI_INVALID_PARAMETER`.
5.1.1.1.4	0xef317ade, 0x8668, 0x456f, 0xbe, 0xd9, 0x76, 0x60, 0x56, 0x67, 0x2d, 0xff	`BS.CreateEvent –` `CreateEvent()` returns `EFI_SUCCESS` with all valid parameters.	1. Call `CreateEvent()` with all valid parameters. The return status must be `EFI_SUCCESS`. 2. Call `CloseEvent()` with the created event.
5.1.1.1.5	0x8759ef89, 0xbc76, 0x4fc1, 0xb8, 0x64, 0x91, 0x9d, 0x33, 0xa9, 0xb3, 0x91	`BS.CreateEvent –` The events created by `CreateEvent()` are invoked in order of each specified notifyTPL.	1. Call `CreateEvent()` to create events with different notification TPLs. 2. Call `RaiseTPL()` to the highest TPL. 3. Call `SignalEvent()` with each created event. 4. Call `RestoreTPL()` to the original TPL. The notification functions of the created event must be invoked in order of each specified notification TPL. 5. Call `CloseEvent()` with each created event.
5.1.1.1.6	0xd4d37597, 0x6367,0x4f9d, 0xad, 0xac, 0x0f, 0xab, 0xe5, 0xb8, 0x3f, 0x2e	`BS.CreateEvent -` `Create event` with `NotifyFunction` being `NULL` and `Type` is `EFI_EVENT_NOTIFY_WAIT` or `EFI_EVENT_NOTIFY_SIGNAL.`	Call `CreateEvent()` with `NotifyFunction` being `NULL` and `EventType` is `EFI_EVENT_NOTIFY_WAIT` or `EFI_EVENT_NOTIFY_SIGNAL`. The return status should be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.1.1.1.7	0x48342406, 0xf478, 0x409e, 0x85, 0xa2, 0xca, 0x65, 0xad, 0xa6, 0xcd, 0xb8	`BS.CreateEvent - Create event` with neither `EVENT_NOTIFY_WAIT` nor `EVENT_NOTIFY_SIGNAL event types and unsupported notify TPLs`	Call `CreateEvent` with neither `EVENT_NOTIFY_WAIT` nor `EVENT_NOTIFY_SIGNAL` event type and unsupported notify TPLs. The return status should be `EFI_SUCCESS`.

3.1.2 CloseEvent()

Number	GUID	Assertion	Test Description
5.1.1.2.1	0xa4f5922e, 0x26f8, 0x4591, 0xbb, 0x2e, 0xba, 0xf8, 0xdc, 0xc1, 0xcd, 0x93	`BS.CloseEvent – CloseEvents()` returns `EFI_SUCCESS` with all valid parameters.	1. Call `CreateEvent()` with all valid parameters. 2. Call `RaiseTPL()` to the highest TPL. 3. Call `SignalEvent()` with the created event. 4. Call `CloseEvent()` with all valid parameters. The return status must be `EFI_SUCCESS`. 5. Call `RestoreTPL()` to the original TPL. The notification function should not be invoked.

3.1.3 SignalEvent()

Number	GUID	Assertion	Test Description
5.1.1.3.1	0x397ab206, 0x7270, 0x484d, 0x8b, 0x2c, 0xd9, 0x0a, 0xeb, 0xe5, 0xad, 0x90	`BS.SignalEvent – SignalEvent()` returns `EFI_SUCCESS` with all valid parameters.	1. Call `CreateEvent()` with all valid parameters. 2. Call `RaiseTPL()` to a TPL lower than the notifification TPL. 3. Call `SignalEvent()` with the created event X times. The notification function will be invoked X times. 4. Call `RaiseTPL()` to a TPL higher than the notification TPL. 5. Call `SignalEvent()` with the created event X times. 6. Call `RestoreTPL()` to the original TPL. The notification function will be invoked once. 7. Call `CloseEvent()` with the created event.

3.1.4 WaitForEvent()

Number	GUID	Assertion	Test Description
5.1.1.4.1	0x8dfd27a6, 0xa43c, 0x4443, 0x92, 0x2a, 0x34, 0x3a, 0x36, 0xee, 0xb9, 0x80	`BS.WaitForEvent – WaitForEvent()` returns `EFI_UNSUPPORTED` from an invalid TPL.	1. Call `CreateEvent()` with all valid parameters. 2. Call `RaiseTPL()` to a TPL higher than `TPL_APPLICATION`. 3. Call `WaitForEvent()` with the created event. The return status must be `EFI_UNSUPPORTED`, and the notification function should not be invoked. 4. Call `CloseEvent()` with the created event.
5.1.1.4.2	0xe38e1362, 0xbf34, 0x4947, 0xa4, 0xf5, 0x39, 0xce, 0xa9, 0x3a, 0xcb, 0x0d	`BS.WaitForEvent – WaitForEvent()` returns `EFI_INVALID_PARAMETER` with an event of type `EVT_NOTIFY_SIGNAL`.	1. Call `CreateEvent()` with the type `EVT_NOTIFY_SIGNAL`. 2. Call `WaitForEvent()` with the created event. The return status must be `EFI_INVALID_PARAMETER`, and the return index must be the index of the created event. 3. Call `CloseEvent()` with the created event.

Number	GUID	Assertion	Test Description
5.1.1.4.3	0xe1e27d6e, 0x1130, 0x475b, 0xb0, 0xaf, 0xa0, 0xa8, 0x10, 0x48, 0xb2, 0xba	**`BS.WaitForEvent – WaitForEvent()`** returns **`EFI_INVALID_PARAMETER`** with a *`NumberOfEvents`* value of 0.	1. Call **`WaitForEvent()`** with a *`NumberOfEvents`* value of 0. The return status must be **`EFI_INVALID_PARAMETER`**.
5.1.1.4.4	0x65657374, 0xc1a4, 0x424d, 0xb5, 0xa6, 0x85, 0x03, 0xf5, 0xb9, 0x75, 0x8d	**`BS.WaitForEvent – WaitForEvent()`** gets the correct index with a signaled event.	1. Call **`CreateEvent()`** with all valid parameters to create a list of events. 2. Call **`SignalEvent()`** with one of created events. 3. Call **`WaitForEvent()`** with the list of events. The return status must be **`EFI_SUCCESS`**, and the output index must be the index of the signaled event. 4. Call **`CloseEvent()`** with each created event.
5.1.1.4.5	0x129c34d4, 0x1045, 0x4fd2, 0x80, 0x57, 0x92, 0x14, 0x1d, 0x63, 0xb8, 0xdc	**`BS.WaitForEvent – WaitForEvent()`** gets the correct index with an un-signaled event.	1. Call **`CreateEvent()`** and **`SetTimer()`** to create a timer to signal the event created in the next step. 2. Call **`CreateEvent()`** with all valid parameters. 3. Call **`WaitForEvent()`** with the created event. The return status must be **`EFI_SUCCESS`**, and the output index must be the index of the event signaled by the timer. 4. Call **`CloseEvent()`** with each created event.

3.1.5 CheckEvent()

Number	GUID	Assertion	Test Description
5.1.1.5.1	0xe69c54f3, 0x5a97, 0x4e09, 0x8f, 0x4b, 0xf3, 0x0f, 0xf1, 0x96, 0x4e, 0x0d	`BS.CheckEvent – CheckEvent()` returns `EFI_INVALID_PARAMETER` with an event of type `EVT_NOTIFY_SIGNAL`.	1. Call `CreateEvent()` with the type `EVT_NOTIFY_SIGNAL`. 2. Call `CheckEvent()` with the created event. The return status must be `EFI_INVALID_PARAMETER`. 3. Call `CloseEvent()` with the created event.
5.1.1.5.2	0x3cb51863, 0x1181, 0x49e5, 0x82, 0xa6, 0x66, 0x70, 0x90, 0x08, 0x81, 0x69	`BS.CheckEvent – CheckEvent()` returns `EFI_NOT_READY` with an event that does not have the notification function.	1. Call `CreateEvent()` without the notification function. 2. Call `CheckEvent()` with the created event. The return status must be `EFI_NOT_READY`. 3. Call `CloseEvent()` with the created event.
5.1.1.5.3	0x4e9aa877, 0x2672, 0x4f8c, 0xba, 0x3c, 0xc0, 0x2f, 0x49, 0xa6, 0x89, 0x11	`BS.CheckEvent – CheckEvent()` returns `EFI_NOT_READY` with an event that has a notification function that does not signal itself.	1. Call `CreateEvent()` with a notification function that does not signal itself. 2. Call `CheckEvent()` with the created event. The return status must be `EFI_NOT_READY`. 3. Call `CloseEvent()` with the created event.
5.1.1.5.4	0x060234f5, 0xa84a, 0x4dd7, 0xad, 0x5b, 0x64, 0x99, 0x62, 0x50, 0xf2, 0x16	`BS.CheckEvent – CheckEvent()` returns `EFI_SUCCESS` with a signaled event.	1. Call `CreateEvent()` with all valid parameters. 2. Call `SignalEvent()` with the created event. 3. Call `CheckEvent()` with the signaled event. The return status must be `EFI_SUCCESS`, and the notification function must not be invoked. 4. Call `CloseEvent()` with the created event.

Number	GUID	Assertion	Test Description
5.1.1.5.5	0xfa181d1b, 0x9fda, 0x4405, 0xb3, 0xb0, 0xf3, 0xfe, 0xdd, 0x30, 0x3e, 0xbe	`BS.CheckEvent – CheckEvent()` returns `EFI_SUCCESS` with an event that has a notification function that signals itself.	1. Call `CreateEvent()` with a notification function that signals itself. 2. Call `CheckEvent()` with the created event. The return status must be `EFI_SUCCESS`, and the notification function must be invoked. 3. Call `CloseEvent()` with the created event.

3.1.6 SetTimer()

Number	GUID	Assertion	Test Description
5.1.1.6.1	0x80bbd29e, 0x0c5b, 0x4f5b, 0xa2, 0x46, 0xdb, 0xea, 0xde, 0xf1, 0x59, 0x9c	`BS.SetTimer – SetTimer()` returns `EFI_INVALID_PARAMETER` with an event that does not include `EVT_TIMER`.	1. Call `CreateEvent()` without including the type `EVT_TIMER`. 2. Call `SetTimer()` with the created event. The return status must be `EFI_INVALID_PARAMETER`. 3. Call `CloseEvent()` with the created event.
5.1.1.6.2	0x16418244, 0x71a4, 0x4e4d, 0x86, 0x62, 0x43, 0xff, 0xf1, 0xac, 0x5e, 0xd7	`BS.SetTimer – SetTimer()` returns `EFI_INVALID_PARAMETER` with an invalid timer type.	1. Call `CreateEvent()` with all valid parameters. 2. Call `SetTimer()` with an invalid timer type. The return status must be `EFI_INVALID_PARAMETER`. 3. Call `CloseEvent()` with the created event.
5.1.1.6.3	0x918f9f6c, 0x5072, 0x41a6, 0x95, 0xec, 0x81, 0x84, 0xaf, 0x57, 0x4e, 0xd1	`BS.SetTimer – SetTimer()` with the type `TimerRelative`; the notification function will be invoked once.	1. Call `CreateEvent()` with all valid parameters. 2. Call `SetTimer()` with the type `TimerRelative`. The return status must be `EFI_SUCCESS`, and the notification function will be invoked once. 3. Call `CloseEvent()` with the created event.
5.1.1.6.4	0x989ba6bc, 0x08eb, 0x4e98, 0xae, 0xa6, 0x9f, 0xe8, 0xe8, 0x73, 0x74, 0xa8	`BS.SetTimer – SetTimer()` with the type `TimerRelative`; the notification function will be invoked more than once.	1. Call `CreateEvent()` with all valid parameters. 2. Call `SetTimer()` with the type `TimerRelative`. The return status must be `EFI_SUCCESS`, and the notification function will be invoked more than once. 3. Call `CloseEvent()` with the created event.

Number	GUID	Assertion	Test Description
5.1.1.6.5	0xbd333dd3, 0x62b2, 0x46eb, 0xbb, 0x4a, 0xa6, 0xb7, 0xb3, 0xde, 0xe2, 0x5f	**BS.SetTimer – SetTimer()** with type of *TimerCancel*; the notification function will not be invoked.	1. Call **CreateEvent()** with all valid parameters. 2. Call **SetTimer()** with the type *TimerCancel*. The return status must be **EFI_SUCCESS**, and the notification function will not be invoked. 3. Call **CloseEvent()** with the created event.
5.1.1.6.6	0xdea3cb68, 0xdc79, 0x4b91, 0x91, 0x34, 0x64, 0xfb, 0x3e, 0xa2, 0x92, 0x03	**BS.SetTimer –** The notification function will be invoked correctly after the timer type is changed by **SetTimer()**.	1. Call **CreateEvent()** with all valid parameters. 2. Call **SetTimer()** with the type **TimerRelative**. 3. Call **SetTimer()** with the type **TimerRelative**. The return status must be **EFI_SUCCESS**, and the notification function will be invoked once. 4. Call **CloseEvent()** with the created event.
5.1.1.6.7	0xe866f000, 0xb5e6, 0x4d29, 0xab, 0xdd, 0x5d, 0xbb, 0x11, 0x8d, 0xc2, 0xc0	**BS.SetTimer – SetTimer()** returns **EFI_SUCCESS** with a *TriggerTime* of 0.	1. Call **CreateEvent()** with all valid parameters. 2. Call **SetTimer()** with a *TriggerTime* of 0. The return status must be **EFI_SUCCESS**, and the notification function will be invoked immediately. 3. Call **CloseEvent()** with the created event.

3.1.7 RaiseTPL()

Number	GUID	Assertion	Test Description
5.1.1.7.1	0x94fff736, 0xc5df, 0x40a6, 0xaa, 0x4f, 0x88, 0x1c, 0x38, 0x0f, 0x78, 0x84	`BS.RaiseTPL – RaiseTPL()` returns the correct TPL with valid parameters.	1. Get the original TPL via `RaiseTPL()` and `RestoreTPL()`. 2. Call `RaiseTPL()` with all valid TPLs. The return TPL must be the same as the original TPL. 3. Call `RaiseTPL()` with the highest TPL. The return TPL must be the same as the TPL passed by the previous `RaiseTPL()`. 4. Call `RestoreTPL()` to the original TPL.

3.1.8 RestoreTPL()

Number	GUID	Assertion	Test Description
5.1.1.8.1	0x08bcd6be, 0x9808, 0x4417, 0x88, 0x3a, 0x5e, 0x54, 0xd3, 0x9f, 0xc3, 0xa8	`BS.RestoreTPL – RestoreTPL()` sets the correct TPL with valid parameters.	1. Get the original TPL via `RaiseTPL()` and `RestoreTPL()`. 2. Call `RaiseTPL()` with all valid TPLs. 3. Call `RestoreTPL()` to the original TPL. 4. Get the current TPL via `RaiseTPL()` and `RestoreTPL()`. This TPL must be the same as the original TPL.

3.1.9 CreateEventEx()

Number	GUID	Assertion	Test Description
5.1.1.9.1	0xd68d782c, 0xc59d, 0x4acb, 0x98, 0x33, 0xdc, 0x5c, 0xad, 0x20, 0xfd, 0x38	**BS.CreateEventEx – CreateEventEx()** returns **EFI_INVALID_PARAMETER** with all invalid event types.	1. Call **CreateEventEx()** with all invalid event types. The return status should be **EFI_INVALID_PARAMETER**.
5.1.1.9.2	0xa74a802f, 0xd632, 0x49f0, 0xa3, 0xde, 0x13, 0xc5, 0x5d, 0x9c, 0x9e, 0x06	**BS.CreateEventEx – CreateEventEx()** returns **EFI_INVALID_PARAMETER** with an invalid notification TPL function.	1. Call **CreateEventEx()** with the notification TPL function **EFI_TPL_APPLICATION**. The return status should be **EFI_INVALID_PARAMETER**.
5.1.1.9.3	0xff0e6747, 0x80b6, 0x4168, 0xa6, 0x6b, 0x66, 0x94, 0xa7, 0x88, 0x10, 0x59	**BS.CreateEventEx – CreateEventEx()** returns **EFI_INVALID_PARAMETER** with an *Event* value of **NULL**.	1. Call **CreateEventEx()** with an *Event* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.1.1.9.4	0x40f0e21f, 0x2ffe, 0x43ca, 0xa0, 0x25, 0x78, 0x32, 0x83, 0xf1, 0xc3, 0x0b	**BS.CreateEventEx – CreateEventEx()** returns **EFI_INVALID_PARAMETER** when either **EFI_EVENT_NOTIFY_WAIT** or **EFI_EVENT_NOTIFY_SIGNAL** is set and **NotifyFunction** is **NULL**.	Call **CreateEventEx()** with a **NotifyTpl** value of: **EFI_EVENT_NOTIFY_WAIT** or **EFI_EVENT_NOTIFY_SIGNAL** or **EFI_EVENT_TIMER** | **EFI_EVENT_NOTIFY_SIGNAL** or **EFI_EVENT_TIMER** | **EFI_EVENT_NOTIFY_WAIT** In addition, a **NotifyFunction** value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.1.1.9.5	0x3e26a97e, 0xda03, 0x4409, 0x98, 0xa1, 0x93, 0x12, 0xbe, 0xb2, 0x8c, 0x43	`BS.CreateEventEx –` Creates an event with valid parameters. Once an event in an event group is signaled, all the events in this group are signaled, and the notification functions are called in the proper order.	1. Call `CreateEventEx()` to create three events with notification functions. Among them, Event0 and Event1 are created with `NotifyTpl` set to `EFI_TPL_CALL_BACK` and `EventGroup` set to TestEventGroup1. Event2 is created with `NotifyTpl` set to `EFI_TPL_NOTIFY` and `EventGroup` set to `NULL`. 2. Call `RaiseTPL()` to raise the current TPL to `TPL_HIGH_LEVEL`,and call `SignalEvent()` to signal all the events in the order of Event0,Event2. 3. Call `RestoreTPL()` to restore the current TPL to the original level. The return status of `CreateEventEx()` should be `EFI_SUCCESS`. After the execution of `RestoreTPL()`, the notification functions of the 3 events should be invoked in the order of Event2, Event1, Event0.

Number	GUID	Assertion	Test Description
5.1.1.9.6	0xf2eb0902, 0x3192, 0x4026,0x89, 0x2e,0x83, 0xa3,0x5b, 0x43,0x27, 0x9c	`BS.CreateEventEx` - Creates an event with valid parameters and Check the notification of the EventGroup and the notify order when call `InstallConfigurationTable`.	1. Call `CreateEventEx()` to create 3 events with the same notification function and same event group. Among them, Event0 and Event1 are created with `NotifyTpl` set to `EFI_TPL_CALL_BACK`. Event2 is created with `NotifyTpl` set to `EFI_TPL_NOTIFY.` 2. Call `RaiseTPL()` to raise the current TPL to `EFI_TPL_HIGH_LEVEL.` Call `InstallConfigurationTable()` to signal all events in the same group. 3. Call `RestoreTPL()` to restore the current `TPL` to the original level. Close all events and remove the newly installed configuration table. After the execution, the notification function of Event3 should be invoked in first.

Number	GUID	Assertion	Test Description
5.1.1.9.7	0xba3d7e17, 0x7ee1, 0x4a0f, 0xaa, 0x99, 0x3c, 0x49, 0x23, 0x3d, 0x6c, 0x36	**BS.CreateEventEx -** Check the notification of the EFI_EVENT_GROUP_MEMORY_MAP_CHANGE and the notify order when Memory Allocation Services is called	1. Call **CreateEventEx()** to create 3 events in **EVT_NOTIFY_SIGNAL** type with the same notification function. Event1 and Event2 are **CALLBACK TPL** and Event3 is **NOTIFY TPL**. They are registered in the **gEfiEventMemoryMapChangeGuid**. 2. Call **RaiseTPL()** to raise the current **TPL** to **EFI_TPL_NOTIFY_LEVEL.** Call **AllocatePages()** to signal all events. 3. Call **RestoreTPL()** to restore the current **TPL** to the original level. Close all events and free the newly allocated pages. After the execution, the notification order should be correct.

3.2 Memory Allocation Services Test

Reference Document:

UEFI Specification, Memory Allocation Services Section.

Table 2. Memory Allocation Functions

Name	Type	Description
AllocatePages()	Boot	Allocates pages of a particular type.
FreePages()	Boot	Frees allocated pages.
GetMemoryMap()	Boot	Returns the current boot services memory map and memory map key.
AllocatePool()	Boot	Allocates a pool of a particular type.
FreePool()	Boot	Frees allocated pool.

3.2.1 AllocatePages()

Number	GUID	Assertion	Test Description
5.1.2.1.1	0x7c9075d2, 0xcbf1, 0x4b57, 0x86, 0x30, 0xde, 0x34, 0xb9, 0xcc, 0x11, 0x90	**BS.AllocatePages – AllocatePages()** returns **EFI_INVALID_PARAMETER** with a *Type* value of **MaxAllocateType**	1. Call **AllocatePages()** with a *Type* value of **MaxAllocateType**. The return code must be **EFI_INVALID_PARAMETER**.
5.1.2.1.2	0x224a63b3, 0x1e41, 0x47b7, 0xa8, 0xdc, 0x82, 0x3d, 0xe4, 0x0d, 0x00, 0xd5	**BS.AllocatePages – AllocatePages()** returns **EFI_INVALID_PARAMETER** with a *Type* value of **MaxAllocateType** + 1.	1. Call **AllocatePages()** with a *Type* value of **MaxAllocateType** + 1. The return code must be **EFI_INVALID_PARAMETER**.
5.1.2.1.3	0x6c330112, 0x24cb, 0x48f2, 0x9e, 0x68, 0x6a, 0xcf, 0x80, 0x7b, 0x40, 0xc4	**BS.AllocatePages – AllocatePages()** returns **EFI_INVALID_PARAMETER** with a *Type* value of –1.	1. Call **AllocatePages()** with a *Type* value of –1. The return code must be **EFI_INVALID_PARAMETER**.
5.1.2.1.4	0x6f5ca3fc, 0x9893, 0x42da, 0xb1, 0x4f, 0x8d, 0x24, 0xf3, 0x49, 0x14, 0x4a	**BS.AllocatePages – AllocatePages()** returns **EFI_INVALID_PARAMETER** with a *MemoryType* value of **EfiMaxMemoryType**.	1. Call **AllocatePages()** with a *MemoryType* value of **EfiMaxMemoryType**. The return code must be **EFI_INVALID_PARAMETER**.
5.1.2.1.5	0x2ca3999f, 0x70a7, 0x4a2a, 0x96, 0x62, 0xf1, 0x42, 0x1a, 0x10, 0x36, 0x89	**BS.AllocatePages – AllocatePages()** returns **EFI_INVALID_PARAMETER** with a *MemoryType* value of **EfiMaxMemoryType** + 1.	1. Call **AllocatePages()** with a *MemoryType* value of **EfiMaxMemoryType** + 1. The return code must be **EFI_INVALID_PARAMETER**.
5.1.2.1.6	0xd26a1cfc, 0x51ef, 0x42c6, 0x99, 0x07, 0x13, 0x72, 0xde, 0xc6, 0xce, 0x80	**BS.AllocatePages – AllocatePages()** returns **EFI_INVALID_PARAMETER** with a *MemoryType* value of 0x6FFFFFFE.	1. Call **AllocatePages()** with a *MemoryType* value of 0x6FFFFFFE. The return code must be **EFI_INVALID_PARAMETER**.
5.1.2.1.7	0xee820dab, 0xf589, 0x49e9, 0xbd, 0xec, 0x84, 0x19, 0x75, 0x44, 0x7e, 0xcd	**BS.AllocatePages – AllocatePages()** returns **EFI_INVALID_PARAMETER** with a *MemoryType* value of 0x6FFFFFFF.	1. Call **AllocatePages()** with a *MemoryType* value of 0x6FFFFFFF. The return code must be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.1.2.1.8	0x9b0c2857, 0x4116, 0x4890, 0xac, 0x8f, 0x61, 0xef, 0x02, 0xbc, 0x2d, 0x75	**BS.AllocatePages – AllocatePages()** returns **EFI_OUT_OF_RESOURCES** with a **Pages** value of *MaxFreePages* + 1.	1. Call **GetMemoryMap()** to get the memory map. Get the page number of the biggest contiguous free memory. 2. Call **AllocatePages()** with a **Pages** value of *MaxFreePages* + 1. The return status must be **EFI_OUT_OF_RESOURCES**.
5.1.2.1.9	0x382e4ce7, 0x81d9, 0x479b, 0xa5, 0xf5, 0x55, 0x80, 0x8e, 0xe7, 0xb7, 0x06	**BS.AllocatePages – AllocatePages()** returns **EFI_NOT_FOUND** with non-existent memory.	1. Call **GetMemoryMap()** to get the memory map. Find a physical address that is not in the range of any memory descriptor. 2. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and Memory containing non-existent memory. The return status must be **EFI_NOT_FOUND**.
5.1.2.1.10	0x69663454, 0x635d, 0x48f8, 0x8e, 0x9a, 0x8b, 0x3f, 0x28, 0xc8, 0x42, 0xc2	**BS.AllocatePages – AllocatePages()** returns **EFI_NOT_FOUND** with allocated memory.	1. Call **GetMemoryMap()** to get the memory map. Find a physical address that has been allocated. 2. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and Memory containing allocated memory. The return status must be **EFI_NOT_FOUND**.
5.1.2.1.11	0x501a28d8, 0x4d4f, 0x4f56, 0x99, 0xa4, 0x45, 0x11, 0xb5, 0xe3, 0x31, 0x9b	**BS.AllocatePages – AllocatePages()** allocates memory with a *Type* value of *AllocateAnyPages* at **EFI_TPL_APPLICATION**.	1. Raise to **EFI_TPL_APPLICATION** via **RaiseTPL()**. 2. Call **AllocatePages()** with a *Type* value of *AllocateAnyPages*. The return status must be **EFI_SUCCESS**. 3. Restore to the previous TPL. 4. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.12	0xb7f8a839, 0xc3bf, 0x4967, 0x85, 0x7f, 0x4a, 0x23, 0xe6, 0x1a, 0x52, 0x4c	**BS.AllocatePages – AllocatePages()** allocates memory with a *Type* value of *AllocateAnyPages* at **EFI_TPL_CALLBACK**.	1. Raise to **EFI_TPL_CALLBACK** via **RaiseTPL()**. 2. Call **AllocatePages()** with a *Type* value of *AllocateAnyPages*. The return status must be **EFI_SUCCESS**. 3. Restore to the previous TPL. 4. Call **FreePages()** to free the allocated memory.
5.1.2.1.13	0x9ba3d098, 0x6457, 0x4287, 0xb7, 0x3c, 0x1c, 0x1a, 0xcb, 0x70, 0xf0, 0x2f	**BS.AllocatePages – AllocatePages()** allocates memory with a *Type* value of *AllocateAnyPages* at **EFI_TPL_NOTIFY**.	1. Raise to **EFI_TPL_NOTIFY** via **RaiseTPL()**. 2. Call **AllocatePages()** with a *Type* value of *AllocateAnyPages*. The return status must be **EFI_SUCCESS**. 3. Restore to the previous TPL. 4. Call **FreePages()** to free the allocated memory.
5.1.2.1.14	0xfcbf390b, 0xf2d3, 0x47ea, 0xb0, 0x60, 0xca, 0x49, 0xcc, 0xb3, 0x42, 0x75	**BS.AllocatePages – AllocatePages()** allocates page-aligned memory with a *Type* value of *AllocateAnyPages* at **EFI_TPL_APPLICATION**.	1. Raise to **EFI_TPL_APPLICATION** via **RaiseTPL()**. 2. Call **AllocatePages()** with a *Type* value of *AllocateAnyPages*. The return Memory must be page-aligned. 3. Restore to the previous TPL. 4. Call **FreePages()** to free the allocated memory.
5.1.2.1.15	0x24e4d5c2, 0x2295, 0x48d2, 0xa5, 0x4e, 0x35, 0x83, 0xa0, 0xf8, 0x67, 0x67	**BS.AllocatePages – AllocatePages()** allocates page-aligned memory with a *Type* value of *AllocateAnyPages* at **EFI_TPL_CALLBACK**.	1. Raise to **EFI_TPL_CALLBACK** via **RaiseTPL()**. 2. Call **AllocatePages()** with a *Type* value of *AllocateAnyPages*. The return Memory must be page-aligned. 3. Restore to the previous TPL. 4. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.16	0x41a830a7, 0x88b8, 0x42a5, 0xb9, 0xb6, 0x71, 0xe8, 0x9d, 0x38, 0x2f, 0x95	**BS.AllocatePages – AllocatePages()** allocates page-aligned memory with a *Type* value of *AllocateAnyPages* at **EFI_TPL_NOTIFY**.	1. Raise to **EFI_TPL_NOTIFY** via **RaiseTPL()**. 2. Call **AllocatePages()** with a *Type* value of *AllocateAnyPages*. The return Memory must be page-aligned. 3. Restore to the previous TPL. 4. Call **FreePages()** to free the allocated memory.
5.1.2.1.17	0x4035dc76, 0xae10, 0x4964, 0x94, 0x06, 0x07, 0x30, 0x68, 0x4c, 0xc3, 0xd7	**BS.AllocatePages – AllocatePages()** allocates memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**, Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return code must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.18	0xa1834910, 0x5c26, 0x4c62, 0x92, 0xa0, 0xad, 0xd0, 0xf4, 0x35, 0x4c, 0x35	**BS.AllocatePages – AllocatePages()** allocates memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**, Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return code must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.19	0xca4d6c22, 0xb382, 0x4546, 0x97, 0xd7, 0x4c, 0x14, 0x72, 0x61, 0xbb, 0x16	**BS.AllocatePages – AllocatePages()** allocates memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**, Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is / 3. Restore to the previous TPL. The return code must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.20	0x3dcb261f, 0x75ec, 0x4384, 0xa1, 0x74, 0x21, 0xff, 0x5c, 0xf1, 0x03, 0x98	**BS.AllocatePages – AllocatePages()** allocates page-aligned memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**, Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.21	0x5f41e4f3, 0x8b1c, 0x4329, 0x97, 0x50, 0xd1, 0x21, 0x89, 0xea, 0x2e, 0x7f	**BS.AllocatePages – AllocatePages()** allocates page-aligned memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**, Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.22	0x7dcdedeb, 0xf204, 0x40c4, 0x8a, 0x84, 0x0f, 0x90, 0x93, 0x90, 0xcf, 0xd0	**BS.AllocatePages – AllocatePages()** allocates page-aligned memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**, Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.23	0xa99d8b50, 0xb10f, 0x4fbb, 0xb7, 0x23, 0x89, 0x54, 0xdf, 0x9f, 0x7e, 0x57	**BS.AllocatePages –** **AllocatePages()** allocates specified memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**, Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be less than or equal to *MaxFreeAddress* – *MaxFreePages* / 3. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.24	0x921d4b59, 0xb5a7, 0x4cff, 0xb1, 0x11, 0x24, 0xd5, 0xdb, 0xdc, 0xda, 0x15	**BS.AllocatePages –** **AllocatePages()** allocates specified memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**, Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be less than or equal to *MaxFreeAddress* – *MaxFreePages* / 3. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.25	0x6a06e702, 0x8564, 0x48d6, 0xbd, 0x05, 0x87, 0xe7, 0x16, 0xc4, 0x25, 0x49	**BS.AllocatePages – AllocatePages()** allocates specified memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**, Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be less than or equal to *MaxFreeAddress* – *MaxFreePages* / 3. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.26	0x97b0a334, 0xe68d, 0x4f6d, 0xb8, 0x63, 0xb5, 0x98, 0x13, 0x01, 0x09, 0x5b	**BS.AllocatePages – AllocatePages()** skips the allocated memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. 3. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 4. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.27	0x41e801c5, 0x9f47, 0x4d2d, 0xb0, 0x11, 0x0c, 0xa0, 0x74, 0x43, 0x57, 0x66	**BS.AllocatePages – AllocatePages()** skips the allocated memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. 3. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 4. Call **FreePages()** to free the allocated memory.
5.1.2.1.28	0xc0f7ee56, 0x8c2f, 0x4bc9, 0x9d, 0xcf, 0x1f, 0x74, 0x36, 0x5e, 0x29, 0xba	**BS.AllocatePages – AllocatePages()** skips the allocated memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. 3. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 4. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.29	0x36b82136, 0xa336, 0x4f34, 0xbb, 0x65, 0xd9, 0xab, 0x57, 0x45, 0xba, 0x24	**BS.AllocatePages – AllocatePages()** allocates page-aligned with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. 3. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 4. Call **FreePages()** to free the allocated memory.
5.1.2.1.30	0x914a008f, 0xfef7, 0x4550, 0x85, 0xf4, 0x81, 0x8d, 0xdb, 0x9c, 0x7e, 0x81	**BS.AllocatePages – AllocatePages()** allocates page-aligned with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. 3. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 4. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.31	0xe3e584d5, 0x4724, 0x4489, 0xb8, 0xa0, 0x0f, 0x0c, 0x88, 0xbb, 0x4a, 0xb9	**`BS.AllocatePages –`** **`AllocatePages()`** allocates page-aligned with a *`Type`* value of **`AllocateMaxAddress`** at **`EFI_TPL_NOTIFY`**.	1. Call **`GetMemoryMap()`** to get the memory map. Find the max free memory address and page number. 2. Call **`AllocatePages()`** with a *`Type`* value of **`AllocateMaxAddress`** and the max free memory address, the required Pages is *`MaxFreePages`* / 3. 3. Raise to **`EFI_TPL_NOTIFY`**. Call **`AllocatePages()`** with a *`Type`* value of **`AllocateMaxAddress`** and the max free memory address, the required Pages is *`MaxFreePages`* / 3. Restore to the previous TPL. The return memory must be page-aligned. 4. Call **`FreePages()`** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.32	0x07042b86, 0xdc99, 0x49a5, 0xa7, 0x99, 0x7a, 0xc8, 0x29, 0xb5, 0xa8, 0xfa	**BS.AllocatePages – AllocatePages()** skips the allocated memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. 3. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return memory must less than or equal to *MaxFreeAddress* – *MaxFreePages* * 2 / 3. 4. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.33	0x87cb26a9, 0xd9d7, 0x4e94, 0x85, 0x9d, 0x18, 0x75, 0x20, 0x8e, 0xfa, 0x3b	**BS.AllocatePages – AllocatePages()** skips the allocated memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. 3. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return memory must less than or equal to *MaxFreeAddress* – *MaxFreePages* * 2 / 3. 4. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.34	0x1020847c, 0xcec5, 0x4201, 0x97, 0x39, 0x10, 0xe6, 0xb8, 0x54, 0xfc, 0xea	**BS.AllocatePages – AllocatePages()** skips the allocated memory with a *Type* value of **AllocateMaxAddress** at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMap()** to get the memory map. Find the max free memory address and page number. 2. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. 3. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateMaxAddress** and the max free memory address, the required Pages is *MaxFreePages* / 3. Restore to the previous TPL. The return memory must less than or equal to *MaxFreeAddress* – *MaxFreePages* * 2 / 3. 4. Call **FreePages()** to free the allocated memory.
5.1.2.1.35	0xc660bfb9, 0x0f5a, 0x4379, 0xad, 0x60, 0x94, 0x94, 0x56, 0x10, 0x76, 0xdb	**BS.AllocatePages – AllocatePages()** allocates memory with a *Type* value of **AllocateAddress** at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.36	0xed56052c, 0x876e, 0x499d, 0xbd, 0xd0, 0x93, 0x9d, 0xd1, 0x72, 0x00, 0x25	**BS.AllocatePages – AllocatePages()** allocates memory with a *Type* value of **AllocateAddress** at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMap()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.37	0x5202b52b, 0x215f, 0x4638, 0x99, 0x32, 0x4a, 0x55, 0x05, 0x84, 0xe9, 0x7d	**BS.AllocatePages – AllocatePages()** allocates memory with a *Type* value of **AllocateAddress** at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMap()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.38	0x75150eec, 0xcc62, 0x47c7, 0xaf, 0x09, 0x47, 0xb8, 0xaa, 0x3f, 0xdb, 0xee	**BS.AllocatePages – AllocatePages()** allocates page-aligned memory with a *Type* value of **AllocateAddress** at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.39	0xca38bfcb, 0x036f, 0x4f3b, 0x89, 0x21, 0xe7, 0x27, 0x6c, 0x91, 0x45, 0x2e	**BS.AllocatePages – AllocatePages()** allocates page-aligned memory with a *Type* value of **AllocateAddress** at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMap()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.40	0xe6e7432c, 0x679d, 0x40da, 0xbd, 0xce, 0xf0, 0xba, 0xb6, 0x9d, 0x21, 0x55	**BS.AllocatePages – AllocatePages()** allocates page-aligned memory with a *Type* value of **AllocateAddress** at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMap()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.41	0x26d0d6aa, 0x49ca, 0x434b, 0x8c, 0x2b, 0xa9, 0x0f, 0x31, 0x2e, 0x6f, 0x5a	**BS.AllocatePages – AllocatePages()** allocates specified memory with a *Type* value of **AllocateAddress** at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address. Restore to the previous TPL. The return memory must be the specified address. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.42	0xbd3eaba7, 0x8c6d, 0x420c, 0x84, 0x56, 0x9d, 0x37, 0x61, 0x7c, 0x8e, 0xcb	**BS.AllocatePages – AllocatePages()** allocates specified memory with a *Type* value of **AllocateAddress** at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMap()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address. Restore to the previous TPL. The return memory must be the specified address. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.43	0x36f46d2d, 0xe1c6, 0x45e2, 0xaa, 0x46, 0x6e, 0x12, 0x18, 0x11, 0x65, 0xd3	**BS.AllocatePages – AllocatePages()** allocates specified memory with a *Type* value of **AllocateAddress** at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMap()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address. Restore to the previous TPL. The return memory must be the specified address. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.44	0x795de369, 0x3491, 0x44f9, 0x9c, 0x4f, 0xcf, 0x9a, 0x2e, 0x46, 0xf4, 0xbc	**BS.AllocatePages – AllocatePages()** allocates the front range memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.45	0xa1c0ad17, 0x6437, 0x404d, 0xbf, 0x96, 0x68, 0xa5, 0x6e, 0x89, 0x3e, 0xff	**BS.AllocatePages – AllocatePages()** allocates the front range memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.46	0xb06f5d52, 0x3e4c, 0x480a, 0xa9, 0x58, 0x4a, 0x96, 0x25, 0x68, 0x5f, 0xbb	**BS.AllocatePages – AllocatePages()** allocates the front range memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.47	0x314ca190, 0x0b96, 0x4485, 0x80, 0x14, 0xbd, 0x99, 0x06, 0x01, 0x05, 0x45	**BS.AllocatePages – AllocatePages()** allocates the front range page-aligned memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.48	0xeb6fb13f, 0x175e, 0x454a, 0x88, 0x0b, 0x1d, 0x6d, 0xc1, 0xb1, 0x3b, 0x98	**BS.AllocatePages – AllocatePages()** allocates the front range page-aligned memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.49	0x3f710c4c, 0x1b2a, 0x4fff, 0x95, 0x23, 0x28, 0x2c, 0x60, 0x89, 0x49, 0x96	`BS.AllocatePages – AllocatePages()` allocates the front range page-aligned memory at `EFI_TPL_NOTIFY`.	1. Call `GetMemoryMep()` to get the memory map. Find the free memory address and page number. 2. Raise to `EFI_TPL_NOTIFY`. Call `AllocatePages()` with a *Type* value of `AllocateAddress` and the free memory address, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call `FreePages()` to free the allocated memory.
5.1.2.1.50	0xa95be66c, 0xc41a, 0x46d5, 0x81, 0xfe, 0x4c, 0xa2, 0x0f, 0xf5, 0x61, 0x76	`BS.AllocatePages – AllocatePages()` allocates the front range specified memory at `EFI_TPL_APPLICATION`.	1. Call `GetMemoryMep()` to get the memory map. Find the free memory address and page number. 2. Raise to `EFI_TPL_APPLICATION`. Call `AllocatePages()` with a *Type* value of `AllocateAddress` and the free memory address, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be the *MaxFreeAddress*. 3. Call `FreePages()` to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.51	0x524a404b, 0xf888, 0x4ce0, 0xb5, 0xec, 0xcd, 0xe5, 0x35, 0x5a, 0xc3, 0xc2	**BS.AllocatePages – AllocatePages()** allocates the front range specified memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be the *MaxFreeAddress*. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.52	0x5417ba5c, 0x3fdd, 0x47ab, 0xa3, 0xfd, 0x37, 0x11, 0x12, 0xeb, 0x81, 0x60	**BS.AllocatePages – AllocatePages()** allocates the front range specified memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be the *MaxFreeAddress*. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.53	0xcc5fe3de, 0x5df7, 0x4430, 0x8e, 0xd6, 0xfb, 0x0f, 0xf3, 0xcf, 0x80, 0xa9	**BS.AllocatePages – AllocatePages()** allocates the middle range memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.54	0xf2308944, 0xd010, 0x401f, 0x84, 0xa5, 0xb2, 0x6a, 0xe0, 0x95, 0x3f, 0x2c	**BS.AllocatePages – AllocatePages()** allocates the middle range memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.55	0x4ce5e0ba, 0x1830, 0x463e, 0x99, 0xd0, 0x11, 0x60, 0xa9, 0xdf, 0x5e, 0xac	**BS.AllocatePages – AllocatePages()** allocates the middle range memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.56	0x42a635a5, 0x60c6, 0x492a, 0x80, 0x6d, 0x17, 0x58, 0x54, 0x35, 0x48, 0xba	**BS.AllocatePages – AllocatePages()** allocates the middle range page-aligned memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.57	0x2dcc2be2, 0x6474, 0x48c9, 0xba, 0xbc, 0x88, 0xf4, 0xe6, 0x7b, 0xad, 0x9d	**BS.AllocatePages – AllocatePages()** allocates the middle range page-aligned memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.58	0xbe11065a, 0x6b98, 0x4713, 0x8d, 0xc6, 0xd9, 0x4c, 0xb2, 0x42, 0xcd, 0xc7	**BS.AllocatePages – AllocatePages()** allocates the middle range page-aligned memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.59	0x38c4fb2a, 0xfc38, 0x48dc, 0xa8, 0x71, 0xe9, 0xba, 0x67, 0x5b, 0x5d, 0x67	**BS.AllocatePages – AllocatePages()** allocates the middle range specified memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be the *MaxFreeAddress* + *MaxFreePages* / 3. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.60	0xb2ce5fd6, 0x6651, 0x4a7e, 0x8a, 0x78, 0x1a, 0x30, 0xf9, 0xfb, 0x37, 0xef	**BS.AllocatePages – AllocatePages()** allocates the middle range specified memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be the *MaxFreeAddress* + *MaxFreePages* / 3. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.61	0x1818d9da, 0x4c0d, 0x4024, 0xaa, 0x2a, 0xd1, 0x64, 0xbb, 0xda, 0x61, 0x0a	**BS.AllocatePages –** **AllocatePages()** allocates the middle range specified memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be the *MaxFreeAddress* + *MaxFreePages* / 3. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.62	0x3e0a81a9, 0x3670, 0x4239, 0x8c, 0x91, 0x5d, 0x99, 0x61, 0x3d, 0x96, 0x44	**BS.AllocatePages –** **AllocatePages()** allocates the back range memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* * 2 / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.63	0x34b922f1, 0x69eb, 0x4ebf, 0x96, 0xb8, 0x88, 0xf2, 0x90, 0x8c, 0x78, 0x9d	**BS.AllocatePages – AllocatePages()** allocates the back range memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* * 2 / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.64	0x716ed29e, 0xc942, 0x4768, 0x9b, 0xc4, 0x2c, 0xcf, 0x8a, 0x27, 0x7e, 0x52	**BS.AllocatePages – AllocatePages()** allocates the back range memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* * 2 / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.65	0xba6c792f, 0xc50a, 0x41ce, 0x97, 0xfa, 0x72, 0xde, 0x0b, 0xb0, 0x7c, 0xda	**BS.AllocatePages – AllocatePages()** allocates the back range page-aligned memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* * 2 / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.66	0x91c452d2, 0x452a, 0x4d7f, 0xbc, 0x7a, 0x9b, 0xc1, 0xb9, 0x00, 0x9b, 0x4e	**BS.AllocatePages – AllocatePages()** allocates the back range page-aligned memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* * 2 / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.67	0x4707f413, 0xd4fe, 0x4f6b, 0x83, 0x11, 0x2a, 0x99, 0x3c, 0x66, 0x4f, 0xc7	**BS.AllocatePages – AllocatePages()** allocates the back range page-aligned memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* * 2 / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.68	0x0016743c, 0x47d3, 0x46ef, 0xaa, 0xc6, 0x3b, 0x53, 0x87, 0x27, 0x03, 0xb1	**BS.AllocatePages – AllocatePages()** allocates the back range specified memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* * 2 / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be the *MaxFreeAddress* + *MaxFreePages* * 2 / 3. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.69	0xcd59e7d8, 0x2f94, 0x43e1, 0xb3, 0x47, 0x56, 0x0f, 0xc9, 0x38, 0x48, 0x9d	**BS.AllocatePages – AllocatePages()** allocates the back range specified memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* * 2 / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be the *MaxFreeAddress* + *MaxFreePages* * 2 / 3. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.70	0x24fb7551, 0xb7cb, 0x44d3, 0xbd, 0xeb, 0x83, 0x9f, 0x42, 0x29, 0x72, 0xc6	**BS.AllocatePages – AllocatePages()** allocates the front range specified memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address + *MaxFreePages* * 2 / 3, a required size value of *MaxFreePages* / 3. Restore to the previous TPL. The return memory must be the *MaxFreeAddress* + *MaxFreePages* * 2 / 3. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.71	0xb46677ff, 0x657f, 0x4ac8, 0x8c, 0x22, 0xdd, 0x18, 0xf5, 0x4d, 0x3e, 0x5b	**BS.AllocatePages – AllocatePages()** allocates 1 page memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of 1. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.72	0x24f43772, 0xb149, 0x4a1a, 0xb0, 0xee, 0x5c, 0x0d, 0x58, 0x62, 0x2c, 0xf4	**BS.AllocatePages – AllocatePages()** allocates 1 page memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of 1. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.73	0xda1285ae, 0xd920, 0x4a2b, 0xac, 0x5d, 0x6e, 0x35, 0xc9, 0xcd, 0xa7, 0x37	**BS.AllocatePages – AllocatePages()** allocates 1 page memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of 1. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.74	0xe8f44262, 0x8a44, 0x4baa, 0xa3, 0xe6, 0x08, 0x34, 0x63, 0xd5, 0xfb, 0x02	**BS.AllocatePages – AllocatePages()** allocates 1 page-aligned memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of 1. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.75	0xfea00605, 0xd3ca, 0x488d, 0xb8, 0xc3, 0xec, 0xd8, 0x2e, 0xe8, 0x13, 0x09	**BS.AllocatePages – AllocatePages()** allocates 1 page-aligned memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of 1. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.76	0x25fff7ef, 0x3c3d, 0x428a, 0x84, 0x30, 0x98, 0xed, 0x44, 0xc1, 0x32, 0xe7	**BS.AllocatePages – AllocatePages()** allocates 1 page-aligned memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of 1. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.77	0x5551cfc4, 0x69e3, 0x41ee, 0xb5, 0x7f, 0x95, 0x4a, 0x3e, 0xae, 0x41, 0x5a	**`BS.AllocatePages – AllocatePages()`** allocates 1 page specified memory at **`EFI_TPL_APPLICATION`**.	1. Call **`GetMemoryMep()`** to get the memory map. Find the free memory address and page number. 2. Raise to **`EFI_TPL_APPLICATION`**. Call **`AllocatePages()`** with a *`Type`* value of **`AllocateAddress`** and the free memory address, a required size value of 1. Restore to the previous TPL. The return memory must be the *`MaxFreeAddress`*. 3. Call **`FreePages()`** to free the allocated memory.
5.1.2.1.78	0x4207a629, 0x5dab, 0x4ec6, 0x87, 0x1e, 0xd9, 0xf7, 0xb0, 0x73, 0x8b, 0x9a	**`BS.AllocatePages – AllocatePages()`** allocates 1 page specified memory at **`EFI_TPL_CALLBACK`**.	1. Call **`GetMemoryMep()`** to get the memory map. Find the free memory address and page number. 2. Raise to **`EFI_TPL_CALLBACK`**. Call **`AllocatePages()`** with a *`Type`* value of **`AllocateAddress`** and the free memory address, a required size value of 1. Restore to the previous TPL. The return memory must be the *`MaxFreeAddress`*. 3. Call **`FreePages()`** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.79	0xe1f99cec, 0xa0f6, 0x4faa, 0xb6, 0xd4, 0x59, 0x5b, 0x46, 0x54, 0x6f, 0xe7	**BS.AllocatePages – AllocatePages()** allocates 1 page specified memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of 1. Restore to the previous TPL. The return memory must be the *MaxFreeAddress*. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.80	0x74333bdf, 0x4ae6, 0x4251, 0x86, 0xc8, 0x7e, 0x13, 0xf4, 0x43, 0xef, 0x46	**BS.AllocatePages – AllocatePages()** allocates (num – 1) pages memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value ofg **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* - 1. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.81	0x7a4005b5, 0xdb06, 0x436b, 0xbe, 0x70, 0xf3, 0x6b, 0x8e, 0x27, 0xac, 0xa0	**BS.AllocatePages – AllocatePages()** allocates (num – 1) pages memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* - 1. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.82	0xb2942967, 0x5d94, 0x4d0a, 0xb9, 0x00, 0x6e, 0xc2, 0x92, 0x04, 0xac, 0x70	**BS.AllocatePages – AllocatePages()** allocates (num – 1) pages memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* - 1. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.83	0x9881d7df, 0x6c22, 0x4062, 0xbe, 0x67, 0xda, 0x8c, 0xa5, 0xd5, 0xfa, 0x61	**BS.AllocatePages – AllocatePages()** allocates (num – 1) pages aligned memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* - 1. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.84	0xed0d3c6f, 0xb9e8, 0x4713, 0xba, 0x6f, 0x04, 0xf2, 0xaa, 0x8a, 0xc5, 0x45	**BS.AllocatePages – AllocatePages()** allocates (num – 1) pages aligned memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* - 1. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.85	0xaeca503a, 0x4948, 0x4014, 0x85, 0x5c, 0x16, 0xc7, 0xd0, 0x95, 0xfa, 0xbb	**BS.AllocatePages –** **AllocatePages()** allocates (num – 1) pages aligned memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* - 1. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.86	0xa9edd440, 0x6d31, 0x49c9, 0x84, 0x3e, 0x76, 0x08, 0x3e, 0xdf, 0x12, 0x22	**BS.AllocatePages –** **AllocatePages()** allocates (num –1) pages specified memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* - 1. Restore to the previous TPL. The return memory must be the *MaxFreeAddress*. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.87	0xfb85b1c9, 0x74a8, 0x41cb, 0xac, 0xed, 0x0f, 0xf4, 0x11, 0x1a, 0xf5, 0x2f	**BS.AllocatePages – AllocatePages()** allocates (num –1) pages specified memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* - 1. Restore to the previous TPL. The return memory must be the *MaxFreeAddress*. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.88	0x1b0d694f, 0x61c6, 0x4d16, 0xae, 0x5d, 0xa7, 0xb1, 0x24, 0x60, 0xed, 0x50	**BS.AllocatePages – AllocatePages()** allocates (num –1) pages specified memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages* - 1. Restore to the previous TPL. The return memory must be the *MaxFreeAddress*. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.89	0x04ffd118, 0xa284, 0x4dda, 0xb5, 0x8f, 0x63, 0xb6, 0x12, 0xe2, 0xab, 0xe6	**BS.AllocatePages – AllocatePages()** allocates num pages memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages*. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.90	0x78cdeb2f, 0x492b, 0x49b5, 0x83, 0x82, 0x18, 0x63, 0xac, 0xe9, 0xa9, 0xa4	**BS.AllocatePages – AllocatePages()** allocates num pages memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages*. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.91	0x32901e32, 0xa85a, 0x4230, 0x99, 0x14, 0xfa, 0xa6, 0xd4, 0x33, 0xa8, 0x13	**BS.AllocatePages – AllocatePages()** allocates num pages memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages*. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.92	0x89e723c7, 0x0b2f, 0x4751, 0xac, 0xc5, 0xe1, 0xba, 0xa6, 0x28, 0xcd, 0x54	**BS.AllocatePages – AllocatePages()** allocates num pages aligned memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages*. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.93	0xa81cb559, 0xdc0c, 0x4893, 0xbb, 0xbd, 0xa4, 0x30, 0xe4, 0x07, 0x8b, 0xb3	**BS.AllocatePages – AllocatePages()** allocates num pages aligned memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages*. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.94	0x2d655fc1, 0x98c3, 0x405e, 0x9a, 0x62, 0x5b, 0xdb, 0x24, 0xa0, 0xd9, 0xc0	**BS.AllocatePages – AllocatePages()** allocates num pages aligned memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages*. Restore to the previous TPL. The return memory must be page-aligned. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.95	0xc1b252ad, 0x2652, 0x4368, 0xb6, 0x75, 0xe4, 0x73, 0x90, 0xef, 0x7a, 0x47	**BS.AllocatePages – AllocatePages()** allocates num pages specified memory at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages*. Restore to the previous TPL. The return memory must be the *MaxFreeAddress*. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.96	0x749fd711, 0x393a, 0x4dee, 0x85, 0xbf, 0xe4, 0xee, 0xf2, 0x69, 0x89, 0xa0	**BS.AllocatePages – AllocatePages()** allocates num pages specified memory at **EFI_TPL_CALLBACK**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages*. Restore to the previous TPL. The return memory must be the *MaxFreeAddress*. 3. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.1.97	0x117696f6, 0xb7f9, 0x41c7, 0xa8, 0x5b, 0xb5, 0xf0, 0x55, 0xfd, 0x96, 0x32	**BS.AllocatePages – AllocatePages()** allocates num pages specified memory at **EFI_TPL_NOTIFY**.	1. Call **GetMemoryMep()** to get the memory map. Find the free memory address and page number. 2. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePages()** with a *Type* value of **AllocateAddress** and the free memory address, a required size value of *MaxFreePages*. Restore to the previous TPL. The return memory must be the *MaxFreeAddress*. 3. Call **FreePages()** to free the allocated memory.
5.1.2.1.98	0xa49b9e70, 0x956a, 0x4f29, 0xbb, 0x7f, 0x37, 0x5a, 0xc0, 0xa7, 0x29, 0x30	**BS.AllocatePages** - **AllocatePages()** returns **EFI_INVALID_PARAMETER** with **NULL** Memory.	1. Call **AllocatePages()** with **NULL** Memory. The return code must be **EFI_INVALID_PARAMETE R**
5.1.2.1.99	0x2d261231, 0xc694, 0x4dbb, 0x83, 0xd0, 0x1d, 0xc8, 0xd3, 0x89, 0x44, 0x5f	**BS.AllocatePages – AllocatePages()** returns **EFI_INVALID_PARAMETER** when *MemoryType* is *EfiPersistentMemory*.	1. Call **AllocatePages()** when *MemoryType* is *EfiPersistentMemory*. The return code must be **EFI_INVALID_PARAMETE R**.

3.2.2 FreePages()

Number	GUID	Assertion	Test Description
5.1.2.2.1	0x3c73e240, 0xe73b, 0x4163, 0x93, 0x72, 0x80, 0x50, 0x61, 0x73, 0xc4, 0x35	BS.FreePages – FreePages() returns EFI_NOT_FOUND with non-existent memory.	1. Call GetMemoryMap() to get the memory map. Find a physical address that is not in the range of any memory descriptor. 2. Call FreePages() with the Memory containing non-existent memory. The return status must be EFI_NOT_FOUND.
5.1.2.2.2	0x0a2e4eb5, 0x1197, 0x41eb, 0xa3, 0x89, 0x15, 0xf7, 0x56, 0x3a, 0xf6, 0xf6	BS.FreePages – FreePages() returns EFI_NOT_FOUND with conventional memory.	1. Call GetMemoryMap() to get the memory map. Find a physical address whose type is *EfiConventionalMemory*. 2. Call FreePages() with the Memory containing conventional memory. The return status must be EFI_NOT_FOUND.
5.1.2.2.3	0x42b2869e, 0xe546, 0x4302, 0x83, 0xb3, 0x39, 0xf1, 0xad, 0x8d, 0x0f, 0x85	BS.FreePages – FreePages() returns EFI_INVALID_PARAMETER with non page-aligned memory.	1. Call FreePages() with the Memory is not a 4KB aligned address. The return status must be EFI_INVALID_PARAMETER.
5.1.2.2.4	0x089cfb08, 0x2990, 0x4f44, 0xb6, 0xa1, 0x4c, 0x73, 0xa5, 0x3e, 0x30, 0xba	BS.FreePages – FreePages() returns EFI_INVALID_PARAMETER with a Pages value of 0.	1. Call AllocatePages() to allocate a block of memory. 2. Call FreePages() with the allocated memory but a Pages value of 0. The return Status code must be EFI_INVALID_PARAMETER. 3. Call FreePages() to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.2.5	0xc5484c8d, 0xc84d, 0x485d, 0x8c, 0x22, 0x46, 0xa1, 0x16, 0xc1, 0x44, 0x1d	BS.FreePages – FreePages() frees 1 page at EFI_TPL_APPLICATION.	1. Call AllocatePages() to allocate 1 page memory. 2. Raise to EFI_TPL_APPLICATION. Call FreePages() to free the allocated memory. Restore to the previous TPL. The return status must be EFI_SUCCESS.
5.1.2.2.6	0x54166362, 0xcd1f, 0x44d5, 0xb5, 0xf1, 0x73, 0x71, 0xc7, 0x91, 0x2b, 0x58	BS.FreePages – FreePages() frees 1 page at EFI_TPL_CALLBACK.	1. Call AllocatePages() to allocate 1 page memory. 2. Raise to EFI_TPL_CALLBACK. Call FreePages() to free the allocated memory. Restore to the previous TPL. The return status must be EFI_SUCCESS.
5.1.2.2.7	0xa46f5e7b, 0x462d, 0x40e0, 0x99, 0x1a, 0x2d, 0xc6, 0x46, 0xc2, 0x31, 0x24	BS.FreePages – FreePages() frees 1 page at EFI_TPL_NOTIFY.	1. Call AllocatePages() to allocate 1 page memory. 2. Raise to EFI_TPL_NOTIFY. Call FreePages() to free the allocated memory. Restore to the previous TPL. The return status must be EFI_SUCCESS.

3.2.3 GetMemoryMap()

Number	GUID	Assertion	Test Description
5.1.2.3.1	0x55a9228e, 0x9960, 0x4558, 0x83, 0xb0, 0x99, 0xdc, 0xf0, 0x7c, 0x4f, 0x56	**BS.GetMemoryMap –** **GetMemoryMap()** returns **EFI_INVALID_PARAMETER** with a *MemoryMapSize* value of **NULL**.	1. Call **GetMemoryMap()** with a *MemoryMapSize* value of **NULL**. The return code must be **EFI_INVALID_PARAMETER**.
5.1.2.3.2	0x1bc8f675, 0x0cbe, 0x4b7a, 0x96, 0xa4, 0x90, 0xc4, 0x19, 0x5a, 0x33, 0x20	**BS.GetMemoryMap –** **GetMemoryMap()** returns **EFI_INVALID_PARAMETER** with a *MemoryMap* value of **NULL**.	1. Call **GetMemoryMap()** with a *MemoryMap* value of **NULL**. The return code must be **EFI_INVALID_PARAMETER**.
5.1.2.3.3	0x8bf2760e, 0x99c8, 0x4c48, 0x96, 0x0c, 0x20, 0x58, 0xf0, 0xa7, 0x51, 0xb0	**BS.GetMemoryMap –** **GetMemoryMap()** returns **EFI_INVALID_PARAMETER** with a *MapKey* value of **NULL**.	1. Call **GetMemoryMap()** with a *MapKey* value of **NULL**. The return code must be **EFI_INVALID_PARAMETER**.
5.1.2.3.4	0x6b854a8c, 0x6fb3, 0x4dbc, 0x9a, 0xc9, 0x10, 0xeb, 0xa6, 0x5e, 0x68, 0x4e	**BS.GetMemoryMap –** **GetMemoryMap()** returns **EFI_INVALID_PARAMETER** with a *DescriptorSize* value of **NULL**.	1. Call **GetMemoryMap()** with a *DescriptorSize* value of **NULL**. The return code must be **EFI_INVALID_PARAMETER**.
5.1.2.3.5	0xbb16e9b8, 0x2716, 0x42de, 0x9d, 0xe0, 0x2a, 0xd4, 0x69, 0xda, 0x37, 0x91	**BS.GetMemoryMap –** **GetMemoryMap()** returns **EFI_INVALID_PARAMETER** with a *DescriptorVersion* value of **NULL**.	1. Call **GetMemoryMap()** with a *DescriptorVersion* value of **NULL**. The return code must be **EFI_INVALID_PARAMETER**.
5.1.2.3.6	0x65130574, 0x7a59, 0x440c, 0x95, 0xc6, 0xc1, 0x9d, 0xdd, 0x2e, 0x48, 0x28	**BS.GetMemoryMap –** **GetMemoryMap()** returns **EFI_BUFFER_TOO_SMALL** with a *MemoryMapSize* value of 0.	1. Call **GetMemoryMap()** with a *MemoryMapSize* value of 0. The return code must be **EFI_BUFFER_TOO_SMALL**.
5.1.2.3.7	0x12c75089, 0x90f6, 0x4e4b, 0xbe, 0xae, 0xa2, 0x7c, 0xde, 0x04, 0x10, 0x5c	**BS.GetMemoryMap –** **GetMemoryMap()** returns **EFI_BUFFER_TOO_SMALL** with the *MemoryMapSize* less than the required.	1. Call **GetMemoryMap()** with a *MemoryMapSize* value of 0. Record the returned *MemoryMapSize* as the required size. 2. Call **GetMemoryMap()** with the required size – 1. The return code must be **EFI_BUFFER_TOO_SMALL**.

Number	GUID	Assertion	Test Description
5.1.2.3.8	0x73225506, 0x9b48, 0x4196, 0x9f, 0x4e, 0x77, 0x4a, 0xe7, 0xfc, 0x81, 0xdf	**BS.GetMemoryMap – GetMemoryMap()** returns the current memory map at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current memory map must be returned.
5.1.2.3.9	0xfb436e4d, 0x7f39, 0x4fdf, 0xbe, 0xf8, 0x5b, 0x4f, 0x66, 0x69, 0x7d, 0x5b	**BS.GetMemoryMap – GetMemoryMap()** returns the current memory map at **EFI_TPL_CALLBACK**.	1. Raise to **EFI_TPL_CALLBACK** via **RaiseTPL()**. 2. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current memory map must be returned. 3. Restore to previous TPL via **RestoreTPL()**.
5.1.2.3.10	0x06a3b2b5, 0xfb48, 0x4b13, 0xa3, 0x80, 0x12, 0xcb, 0x9d, 0x7f, 0xdd, 0xfb	**BS.GetMemoryMap – GetMemoryMap()** returns the current memory map at **EFI_TPL_NOTIFY**.	1. Raise to **EFI_TPL_NOTIFY** via **RaiseTPL()**. 2. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current memory map must be returned. 3. Restore to previous TPL via **RestoreTPL()**.
5.1.2.3.11	0x53e08693, 0xc268, 0x4b70, 0xa0, 0x20, 0xc7, 0x8c, 0x49, 0xfa, 0xf0, 0x40	**BS.GetMemoryMap – GetMemoryMap()** returns the current *MapKey* at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current *MapKey* must be returned.
5.1.2.3.12	0x04e010ff, 0x860b, 0x40b1, 0xbe, 0x2c, 0x07, 0xdb, 0xb3, 0xf8, 0x65, 0x0a	**BS.GetMemoryMap – GetMemoryMap()** returns the current *MapKey* at **EFI_TPL_CALLBACK**.	1. Raise to **EFI_TPL_CALLBACK** via **RaiseTPL()**. 2. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current *MapKey* must be returned. 3. Restore to previous TPL via **RestoreTPL()**.

Number	GUID	Assertion	Test Description
5.1.2.3.13	0x1030be5b, 0x38bd, 0x4131, 0x97, 0x8d, 0x91, 0x98, 0xd6, 0xca, 0xd1, 0x3d	**BS.GetMemoryMap – GetMemoryMap()** returns the current *MapKey* at **EFI_TPL_NOTIFY**.	1. Raise to **EFI_TPL_NOTIFY** via **RaiseTPL()**. 2. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current *MapKey* must be returned. 3. Restore to previous TPL via **RestoreTPL()**.
5.1.2.3.14	0x007f4e8e, 0x0ed3, 0x479e, 0x8f, 0xc7, 0xcb, 0x5d, 0xf2, 0x4d, 0xd3, 0x83	**BS.GetMemoryMap – GetMemoryMap()** returns the current *MapKey* after **AllocatePages()** at **EFI_TPL_APPLICATION**.	1. Call **AllocatePages()** to allocate a block of memory. 2. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current *MapKey* must be returned. 3. Call **FreePages()** to free the allocated memory.
5.1.2.3.15	0x15255fb4, 0x7c7b, 0x488a, 0xa8, 0xe5, 0x26, 0xce, 0x95, 0xb1, 0x8b, 0xe2	**BS.GetMemoryMap – GetMemoryMap()** returns the current *MapKey* after **AllocatePages()** at **EFI_TPL_CALLBACK**.	1. Call **AllocatePages()** to allocate a block of memory. 2. Raise to **EFI_TPL_CALLBACK** via **RaiseTPL()**. 3. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current *MapKey* must be returned. 4. Restore to previous TPL via **RestoreTPL()**. 5. Call **FreePages()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.3.16	0xf069b658, 0x9196, 0x4915, 0x8e, 0x5f, 0xbb, 0xaa, 0x0f, 0x56, 0x1a, 0xa0	**BS.GetMemoryMap – GetMemoryMap()** returns the current *MapKey* after **AllocatePages()** at **EFI_TPL_NOTIFY**.	1. Call **AllocatePages()** to allocate a block of memory. 2. Raise to **EFI_TPL_NOTIFY** via **RaiseTPL()**. 3. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current *MapKey* must be returned. 4. Restore to previous TPL via **RestoreTPL()**. 5. Call **FreePages()** to free the allocated memory.
5.1.2.3.17	0xe8721bb8, 0xbefa, 0x4839, 0x84, 0x9f, 0xdb, 0xb4, 0xcf, 0x21, 0x38, 0x03	**BS.GetMemoryMap – GetMemoryMap()** returns the current *MapKey* after **FreePages()** at **EFI_TPL_APPLICATION**.	1. Call **AllocatePages()** to allocate a block of memory. 2. Call **FreePages()** to free the allocated memory. 3. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current *MapKey* must be returned.
5.1.2.3.18	0xc004a412, 0x0487, 0x49d6, 0x93, 0xe6, 0x0d, 0x6e, 0x26, 0xa5, 0x58, 0x8f	**BS.GetMemoryMap – GetMemoryMap()** returns the current *MapKey* after **FreePages()** at **EFI_TPL_CALLBACK**.	1. Call **AllocatePages()** to allocate a block of memory. 2. Call **FreePages()** to free the allocated memory. 3. Raise to **EFI_TPL_CALLBACK** via **RaiseTPL()**. 4. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current *MapKey* must be returned. 5. Restore to previous TPL via **RestoreTPL()**.

Number	GUID	Assertion	Test Description
5.1.2.3.19	0x5c536f96, 0x7a27, 0x4425, 0xba, 0x91, 0xe1, 0x10, 0x22, 0x7a, 0x07, 0xed	**BS.GetMemoryMap – GetMemoryMap()** returns the current *MapKey* after **FreePages()** at **EFI_TPL_NOTIFY**.	1. Call **AllocatePages()** to allocate a block of memory. 2. Call **FreePages()** to free the allocated memory. 3. Raise to **EFI_TPL_NOTIFY** via **RaiseTPL()**. 4. Call **GetMemoryMap()** with valid parameters. The return status must be **EFI_SUCCESS** and the current *MapKey* must be returned. 5. Restore to previous TPL via **RestoreTPL()**.
5.1.2.3.20	0xe7fe82f4, 0xc7f5, 0x4181, 0xab, 0x37, 0x20, 0xa1, 0x51, 0xfa, 0x98, 0xe6	**BS.GetMemoryMap – GetMemoryMap()** returns different *MapKey*s after **AllocatePages()** and **FreePages()** at **EFI_TPL_APPLICATION**.	1. Call **GetMemoryMap()** with valid parameters. Record the return *MapKey*. 2. Call **AllocatePages()** to allocate a block of memory. 3. Call **GetMemoryMap()** with valid parameters. Record the return *MapKey*. This *MapKey* must be different from the first one. 4. Call **FreePages()** to free the allocated memory. 5. Call **GetMemoryMap()** with valid parameters. This *MapKey* must be different from the second one.

Number	GUID	Assertion	Test Description
5.1.2.3.21	0x3093039c, 0xdff7, 0x4097, 0x9a, 0x36, 0xd7, 0x96, 0x82, 0x81, 0xc1, 0x46	**BS.GetMemoryMap – GetMemoryMap()** returns different *MapKey*s after **AllocatePages()** and **FreePages()** at **EFI_TPL_CALLBACK**.	1. Raise to **EFI_TPL_CALLBACK**, Call **GetMemoryMap()** with valid parameters. Restore to previous TPL.. Record the return *MapKey*. 2. Call **AllocatePages()** to allocate a block of memory. 3. Raise to **EFI_TPL_CALLBACK**. Call **GetMemoryMap()** with valid parameters. Restore to previous TPL. Record the return *MapKey*. This *MapKey* must be different from the first one. 4. Call **FreePages()** to free the allocated memory. 5. Raise to **EFI_TPL_CALLBACK**, Call **GetMemoryMap()** with valid parameters. Restore to previous TPL. This *MapKey* must be different from the second one.

Number	GUID	Assertion	Test Description
5.1.2.3.22	0x284e0cc8, 0x913a, 0x4e8b, 0xbd, 0x05, 0xb4, 0xc8, 0xe1, 0x95, 0xc3, 0x69	**BS.GetMemoryMap – GetMemoryMap()** returns different *MapKey*s after **AllocatePages()** and **FreePages()** at **EFI_TPL_NOTIFY**.	1. Raise to **EFI_TPL_NOTIFY**, Call **GetMemoryMap()** with valid parameters. Restore to previous TPL. Record the return *MapKey*. 2. Call **AllocatePages()** to allocate a block of memory. 3. Raise to **EFI_TPL_NOTIFY**. Call **GetMemoryMap()** with valid parameters. Restore to previous TPL. Record the return *MapKey*. This *MapKey* must be different from the first one. 4. Call **FreePages()** to free the allocated memory. 5. Raise to **EFI_TPL_NOTIFY**. Call **GetMemoryMap()** with valid parameters. Restore to previous TPL. This *MapKey* must be different from the second one.

3.2.4 AllocatePool()

Number	GUID	Assertion	Test Description
5.1.2.4.1	0x99f47ede, 0x57c9, 0x4892, 0x94, 0x3e, 0xf0, 0xf5, 0x08, 0xb2, 0x3b, 0x91	**BS.AllocatePool – AllocatePool()** returns **EFI_INVALID_PARAMETER** with a *Type* value of **EfiMaxMemoryType**.	1. Call **AllocatePool()** with a *Type* value of **EfiMaxMemoryType**. The return status must be **EFI_INVALID_PARAMETER**.
5.1.2.4.2	0xcff743c0, 0x83e6, 0x4fd2, 0x8d, 0x94, 0x9c, 0x01, 0x7b, 0x3c, 0xdf, 0x45	**BS.AllocatePool – AllocatePool()** returns **EFI_INVALID_PARAMETER** with a *Type* value of **EfiMaxMemoryType** + 1.	1. Call **AllocatePool()** with a *Type* value of **EfiMaxMemoryType** + 1. The return status must be **EFI_INVALID_PARAMETER**.
5.1.2.4.3	0xa4c46515, 0x1e87, 0x472c, 0xae, 0xac, 0x0b, 0x91, 0xf8, 0x3a, 0xcb, 0x4c	**BS.AllocatePool – AllocatePool()** returns **EFI_INVALID_PARAMETER** with a *Type* value of 0x6FFFFFFE.	1. Call **AllocatePool()** with a *Type* value of 0x6FFFFFFE. The return status must be **EFI_INVALID_PARAMETER**.
5.1.2.4.4	0xd97381cf, 0xb4d5, 0x483b, 0xa2, 0xe2, 0xdc, 0x7f, 0xb9, 0xfe, 0xe9, 0x1d	**BS.AllocatePool – AllocatePool()** returns **EFI_INVALID_PARAMETER** with a *Type* value of 0x6FFFFFFE.	1. Call **AllocatePool()** with a *Type* value of 0x6FFFFFFE. The return status must be **EFI_INVALID_PARAMETER**.
5.1.2.4.5	0xee50a1e8, 0x5adb, 0x4cba, 0xad, 0x6d, 0xcf, 0x2f, 0x90, 0x05, 0xee, 0xce	**BS.AllocatePool – AllocatePool()** returns **EFI_OUT_OF_RESOURCES** with a *Size* value of **MaxFreeMemory** + 1.	1. Call **GetMemoryMap()** to get the memory map. Get the size of the biggest contiguous free memory. 2. Call **AllocatePool()** with a *Size* value of **MaxFreeMemory** + 1. The return status must be **EFI_OUT_OF_RESOURCES**.
5.1.2.4.6	0xd60b985b, 0xa3b3, 0x4040, 0xad, 0xb6, 0xcd, 0x69, 0x20, 0xe3, 0x8e, 0xc2	**BS.AllocatePool – AllocatePool()** allocates memory at **EFI_TPL_APPLICATION**.	1. Raise to **EFI_TPL_APPLICATION**. Call **AllocatePool()** to allocate 1 byte memory. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 2. Call **FreePool()** to free the allocated memory.

Number	GUID	Assertion	Test Description
5.1.2.4.7	0x2f3a94f3, 0x95ba, 0x4d5c, 0xba, 0xcc, 0x32, 0xa3, 0xe4, 0xe9, 0x7d, 0x9e	**BS.AllocatePool – AllocatePool()** allocates memory at **EFI_TPL_CALLBACK**.	1. Raise to **EFI_TPL_CALLBACK**. Call **AllocatePool()** to allocate 1 byte memory. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 2. Call **FreePool()** to free the allocated memory.
5.1.2.4.8	0xb6666c18, 0x25c8, 0x4e93, 0x96, 0x00, 0x66, 0x48, 0x90, 0xb3, 0xaf, 0xe8	**BS.AllocatePool – AllocatePool()** allocates memory at **EFI_TPL_NOTIFY**.	1. Raise to **EFI_TPL_NOTIFY**. Call **AllocatePool()** to allocate 1 byte memory. Restore to the previous TPL. The return status must be **EFI_SUCCESS**. 2. Call **FreePool()** to free the allocated memory.
5.1.2.4.9	0xe6ee903a, 0x88a3, 0x4428, 0xb0, 0x05, 0x62, 0x59, 0x43, 0xed, 0x6e, 0x9d	**BS.AllocatePool** - **AllocatePool()** returns **EFI_INVALID_PARAMETER** with **NULL** Buffer.	1. Call **AllocatePool()** with **NULL** Buffer. The return code must be **EFI_INVALID_PARAMETE R**.
5.1.2.4.10	0x41062e36, 0x7401, 0x4b0c, 0xb4, 0xe9, 0xe7, 0xaa, 0x27, 0xcc, 0xa8, 0x8	**AllocatePool()** returns **EFI_INVALID_PARAMETER** when *MemoryType* is *EfiPersistentMemory*.	1. Call **AllocatePool()** when *MemoryType* is *EfiPersistentMemory*. The return code must be **EFI_INVALID_PARAMETE R**.

3.2.5 FreePool()

Number	GUID	Assertion	Test Description
5.1.2.5.1	0xcb7b4b1c, 0x26a1, 0x4302, 0xbd, 0x71, 0xd3, 0xf9, 0xef, 0x4e, 0x93, 0xb7	`BS.FreePool` – `FreePool()` returns `EFI_INVALID_PARAMETER` with a *`Buffer`* value of `NULL`.	1. Call `FreePool()` with a *`Buffer`* value of `NULL`. The return status must be `EFI_INVALID_PARAMETER`.
5.1.2.5.2	0xeccf8a71, 0xbd7d, 0x45f3, 0xa3, 0x70, 0xa4, 0x0f, 0xb7, 0x34, 0xac, 0xdc	`BS.FreePool` – `FreePool()` frees memory at `EFI_TPL_APPLICATION`.	1. Call `AllocatePool()` to allocate 1 byte memory. 2. Raise to `EFI_TPL_APPLICATION`. Call `FreePool()` to free the allocated memory. Restore to the previous TPL. The return status must be `EFI_SUCCESS`.
5.1.2.5.3	0x3bd08624, 0x28eb, 0x475b, 0x93, 0xfc, 0x69, 0x56, 0xaf, 0x7c, 0xc0, 0x7b	`BS.FreePool` – `FreePool()` frees memory at `EFI_TPL_CALLBACK`.	1. Call `AllocatePool()` to allocate 1 byte memory. 2. Raise to `EFI_TPL_CALLBACK`. Call `FreePool()` to free the allocated memory. Restore to the previous TPL. The return status must be `EFI_SUCCESS`.
5.1.2.5.4	0xdc1fa4f1, 0x91c5, 0x4edc, 0xa1, 0x00, 0x8a, 0x95, 0x32, 0xb8, 0x89, 0x14	`BS.FreePool` – `FreePool()` frees memory at `EFI_TPL_NOTIFY`.	1. Call `AllocatePool()` to allocate 1 byte memory. 2. Raise to `EFI_TPL_NOTIFY`. Call `FreePool()` to free the allocated memory. Restore to the previous TPL. The return status must be `EFI_SUCCESS`.

3.3 Protocol Handler Services Test

Reference Document:

UEFI Specification, Protocol Handler Services Section.

Table 3. Protocol Interface Functions

Name	Boot	Description
"InstallProtocolInterface()"	Boot	Installs a protocol interface on a device handle.
UninstallProtocolInterface()	Boot	Removes a protocol interface from a device handle.
ReinstallProtocolInterface()	Boot	Reinstalls a protocol interface on a device handle.

Name	**Boot**	**Description**
RegisterProtocolNotify()	Boot	Registers an event that is to be signaled whenever an interface is installed for a specified protocol.
LocateHandle()	Boot	Returns an array of handles that support a specified protocol.
HandleProtocol()	Boot	Queries a handle to determine if it supports a specified protocol.
LocateDevicePath()	Boot	Locates all devices on a device path that support a specified protocol and returns the handle to the device that is closest to the path.
OpenProtocol()	Boot	Adds elements to the list of agents consuming a protocol interface.
CloseProtocol()	Boot	Removes elements from the list of agents consuming a protocol interface.
OpenProtocolInformation()	Boot	Retrieve the list of agents that are currently consuming a protocol interface.
ConnectController()	Boot	Uses a set of precedence rules to find the best set of drivers to manage a controller.
DisconnectController()	Boot	Informs a set of drivers to stop managing a controller.
ProtocolsPerHandle()	Boot	Retrieves the list of protocols installed on a handle. The return buffer is automatically allocated.
LocateHandleBuffer()	Boot	Retrieves the list of handles from the handle database that meet the search criteria. The return buffer is automatically allocated.
LocateProtocol()	Boot	Finds the first handle in the handle database the supports the requested protocol.
InstallMultipleProtocolInterfaces()	Boot	Installs one or more protocol interfaces onto a handle.
UninstallMultipleProtocolInterfaces()	Boot	Uninstalls one or more protocol interfaces from a handle.

3.3.1 InstallProtocolInterface()

Number	GUID	Assertion	Test Description
5.1.3.1.1	0xd9fedaff, 0xc22b, 0x47b7, 0x86, 0xb7, 0x27, 0x0a, 0x50, 0x06, 0x86, 0x22	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_INVALID_PARAMETER with invalid interface type.	1. Call InstallProtocolInterface() with the interface type other than EFI_NATIVE_INTERFACE. The return code must be EFI_INVALID_PARAMETER.
5.1.3.1.2	0x016ba242, 0x367d, 0x4a8d, 0x8f, 0x07, 0x51, 0x7e, 0x34, 0x5c, 0x6b, 0x83	BS.InstallProtocolInterface – InstallProtolInterface() returns EFI_INVALID_PARAMETER with invalid handle.	1. Call InstallProtocolInterface() with an invalid handle (*Handle* = NULL or *Handle* is invalid). Each return code must be EFI_INVALID_PARAMETER.
5.1.3.1.3	0xf3b82a36, 0x9dc7, 0x4754, 0xb4, 0x25, 0xa9, 0xda, 0xff, 0x06, 0x94, 0xd8	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_INVALID_PARAMETER with same protocol multiple times.	1. Call InstallProtocolInterface() to install TestProtocol1 onto a new handle. 2. Call InstallProtocolInterface() again to try to install TestProtocol1 onto the same handle. The return code must be EFI_INVALID_PARAMETER.
5.1.3.1.4	0xe19b4a73, 0x7652, 0x4bf4, 0x96, 0x11, 0x16, 0xe3, 0x46, 0xe1, 0x83, 0x97	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_INVALID_PARAMETER with *Protocol* is NULL.	1. Call InstallProtocolInterface() with a *Protocol* value of NULL. The return code should be EFI_INVALID_PARAMETER.
5.1.3.1.5	0xb546a05c, 0x1cb5, 0x4c4f, 0x9e, 0x4d, 0x61, 0x30, 0x8a, 0x4c, 0x0c, 0xc5	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with a new handle at EFI_TPL_APPLICATION	1. Call InstallProtocolInterface() to install the TestProtocol1 as type EFI_NATIVE_INTERFACE onto a new handle created by this function call. The InstallProtocolInterface() return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.1.6	0x023420e7, 0x5921, 0x4d64, 0xaa, 0xc8, 0x41, 0x70, 0xf2, 0x5d, 0x21, 0x03	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with a new handle at `EFI_TPL_CALLBACK`	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto a new handle created by this function call. The `InstallProtocolInterface()` return code should be `EFI_SUCCESS`.
5.1.3.1.7	0x04399b4c, 0xd2f8, 0x44fc, 0xa0, 0x9b, 0xf2, 0xb1, 0x86, 0x77, 0x72, 0x4a	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with a new handle at `EFI_TPL_NOTIFY`	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto a new handle created by this function call. The `InstallProtocolInterface()` return code should be `EFI_SUCCESS`.
5.1.3.1.8	0x3e0c0947, 0x29f8, 0x4097, 0x82, 0x3f, 0xe6, 0x2a, 0x27, 0x45, 0xe0, 0x90	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on a new handle at `EFI_TPL_APPLICATION`	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto a new handle created by this function call. A new handle is created.
5.1.3.1.9	0x157e0e28, 0xa05f, 0x4a7e, 0x8d, 0xb0, 0xdd, 0xa8, 0x16, 0xf7, 0x2a, 0x1a	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on a new handle at `EFI_TPL_CALLBACK`.	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto a new handle created by this function call. A new handle is created.
5.1.3.1.10	0x16101f58, 0x8faf, 0x4a15, 0x82, 0x98, 0x85, 0x60, 0xad, 0x1e, 0x6c, 0x85	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on a new handle at `EFI_TPL_NOTIFY`.	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto a new handle created by this function call. A new handle is created.

Number	GUID	Assertion	Test Description
5.1.3.1.11	0xffd329d5, 0x37bc, 0x44d0, 0x83, 0x74, 0xa7, 0x5e, 0xa6, 0x79, 0xfb, 0x2a	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on a new handle at `EFI_TPL_APPLICATION`.	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto a new handle created by this function call. 2. Call `LocateHandleBuffer()` to locate the handle via the protocol. The new handle should be located.
5.1.3.1.12	0xb8798dc8, 0x257f, 0x489e, 0x8c, 0x62, 0x3a, 0xf5, 0xc3, 0x16, 0xb3, 0xf3	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on a new handle at `EFI_TPL_CALLBACK`.	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto a new handle created by this function call. 2. Call `LocateHandleBuffer()` to locate the handle via the protocol. The new handle should be located.
5.1.3.1.13	0x284345a7, 0x7041, 0x459d, 0xbd, 0xad, 0xa7, 0xcc, 0x67, 0x81, 0xdb, 0xc2	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on a new handle at `EFI_TPL_NOTIFY`.	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto a new handle created by this function call. 2. Call `LocateHandleBuffer()` to locate the handle via the protocol. The new handle should be located.

Number	GUID	Assertion	Test Description
5.1.3.1.14	0x2327caf0, 0xa5b4, 0x4234, 0x9d, 0x8d, 0x84, 0x38, 0xce, 0xa4, 0x86, 0xb3	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on a new handle at `EFI_TPL_APPLICATION`.	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto a new handle created by this function call. 2. Call `HandleProtocol()` to locate the protocol via the handle. The `TestProtocol1` should be located.
5.1.3.1.15	0x068d699f, 0xa42a, 0x47d0, 0xbb, 0xa9, 0x27, 0x2e, 0xf3, 0x36, 0x01, 0xfa	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on a new handle at `EFI_TPL_CALLBACK`.	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto a new handle created by this function call. 2. Call `HandleProtocol()` to locate the protocol via the handle. The `TestProtocol1` should be located.
5.1.3.1.16	0x6e72a454, 0x5650, 0x4d1b, 0x9a, 0x20, 0xc9, 0x9b, 0x26, 0x4c, 0x73, 0xab	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on a new handle at `EFI_TPL_NOTIFY`.	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto a new handle created by this function call. 2. Call `HandleProtocol()` to locate the protocol via the handle. The `TestProtocol1` should be located.

Number	GUID	Assertion	Test Description
5.1.3.1.17	0x539a7928, 0xd5a2, 0x400c, 0x91, 0x43, 0xe0, 0xeb, 0xe0, 0xe4, 0xf3, 0x24	BS.InstallProtocolInterface – InstallProtocolInterface() installs the protocol on a new handle at EFI_TPL_APPLICATION.	1. Call InstallProtocolInterface() to install the TestProtocol1 as type EFI_NATIVE_INTERFACE onto a new handle created by this function call. 2. Call the TestProtocol1's function. It should be accessed and be executed correctly.
5.1.3.1.18	0xfe3570b6, 0xa952, 0x4dd0, 0xa5, 0x7d, 0x45, 0x25, 0x4b, 0xde, 0x05, 0x04	BS.InstallProtocolInterface – InstallProtocolInterface() installs the protocol on a new handle at EFI_TPL_CALLBACK.	1. Call InstallProtocolInterface() to install the TestProtocol1 as type EFI_NATIVE_INTERFACE onto a new handle created by this function call. 2. Call the TestProtocol1's function. It should be accessed and be executed correctly.
5.1.3.1.19	0x202e4f04, 0x65b9, 0x4372, 0xb6, 0xf0, 0xc1, 0x54, 0x4b, 0x94, 0xdf, 0x93	BS.InstallProtocolInterface – InstallProtocolInterface() installs the protocol on a new handle at EFI_TPL_NOTIFY.	1. Call InstallProtocolInterface() to install the TestProtocol1 as type EFI_NATIVE_INTERFACE onto a new handle created by this function call. 2. Call the TestProtocol1's function. It should be accessed and be executed correctly.
5.1.3.1.20	0x1efb5778, 0xdf04, 0x4b8e, 0xa3, 0xe0, 0x89, 0xee, 0x3b, 0xc0, 0xbf, 0xd6	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with an existing handle at EFI_TPL_APPLICATION.	1. Call InstallProtocolInterface() to install the TestProtocol1 as type EFI_NATIVE_INTERFACE onto an existing handle created by this function call. The InstallProtocolInterface() return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.1.21	0xf66d17da, 0x9701, 0x4bb1, 0x82, 0x3a, 0xdb, 0x3b, 0xce, 0x93, 0xd5, 0x92	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with an existing handle at `EFI_TPL_CALLBACK`.	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto an existing handle created by this function call. The `InstallProtocolInterface()` return code should be `EFI_SUCCESS`.
5.1.3.1.22	0x244ffd78, 0x895d, 0x4924, 0xb4, 0xd2, 0x03, 0x9d, 0x78, 0x68, 0x6e, 0x47	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with an existing handle at `EFI_TPL_NOTIFY`.	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto an existing handle created by this function call. The `InstallProtocolInterface()` return code should be `EFI_SUCCESS`.
5.1.3.1.23	0x73619777, 0x3376, 0x4217, 0xa0, 0x8b, 0xde, 0x5c, 0x97, 0xb5, 0xf2, 0xd7	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on an existing handle at `EFI_TPL_APPLICATION`	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto an existing handle created by this function call. No new handle is created.
5.1.3.1.24	0x23ab54a9, 0x8165, 0x4c3f, 0x92, 0x18, 0xd2, 0x2a, 0xba, 0x3a, 0x09, 0xdc	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on an existing handle at `EFI_TPL_CALLBACK`	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto an existing handle created by this function call. No new handle is created.

Number	GUID	Assertion	Test Description
5.1.3.1.25	0x5bac7cbe, 0x62a2, 0x492d, 0x87, 0xd9, 0xf2, 0xee, 0x46, 0x67, 0x33, 0xba	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on an existing handle at `EFI_TPL_NOTIFY`	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto an existing handle created by this function call. No new handle is created.
5.1.3.1.26	0xa68ce171, 0xd077, 0x460a, 0xae, 0x94, 0x48, 0x4a, 0xfb, 0xa8, 0x4d, 0x3c	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on an existing handle at `EFI_TPL_APPLICATION`	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto an existing handle created by this function call. 2. Call `LocateHandleBuffer()` to locate the handle via the protocol. The handle should be located.
5.1.3.1.27	0xe8ad2040, 0x0241, 0x43fc, 0x99, 0xb3, 0x38, 0x7d, 0xa6, 0x6d, 0x08, 0x9f	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on an existing handle at `EFI_TPL_CALLBACK`	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto an existing handle created by this function call. 2. Call `LocateHandleBuffer()` to locate the handle via the protocol. The handle should be located.
5.1.3.1.28	0x6aa0b008, 0xc1ff, 0x4355, 0x98, 0x34, 0xab, 0xf9, 0x4d, 0x7d, 0x4e, 0x0d	`BS.InstallProtocolInterface – InstallProtocolInterface()` installs the protocol on an existing handle at `EFI_TPL_NOTIFY`	1. Call `InstallProtocolInterface()` to install the `TestProtocol1` as type `EFI_NATIVE_INTERFACE` onto an existing handle created by this function call. 2. Call `LocateHandleBuffer()` to locate the handle via the protocol. The handle should be located.

Number	GUID	Assertion	Test Description
5.1.3.1.29	0x69a0c9c5, 0xbe97, 0x4a71, 0xaf, 0xb7, 0xa2, 0xf5, 0x10, 0x70, 0x24, 0xf5	**BS.InstallProtocolInterface – InstallProtocolInterface()** installs the protocol on existing handle at **EFI_TPL_APPLICATION**	1. Call **InstallProtocolInterface()** to install the **TestProtocol1** as type **EFI_NATIVE_INTERFACE** onto an existing handle created by this function call. 2. Call **HandleProtocol()** to locate the protocol via the handle. The **TestProtocol1** should be located.
5.1.3.1.30	0x44c3605a, 0x0396, 0x4023, 0x92, 0xbd, 0x30, 0xab, 0xa5, 0x59, 0x93, 0x05	**BS.InstallProtocolInterface – InstallProtocolInterface()** installs the protocol on an existing handle at **EFI_TPL_CALLBACK**	1. Call **InstallProtocolInterface()** to install the **TestProtocol1** as type **EFI_NATIVE_INTERFACE** onto an existing handle created by this function call. 2. Call **HandleProtocol()** to locate the protocol via the handle. The **TestProtocol1** should be located.
5.1.3.1.31	0x5745edb2, 0x6384, 0x4a6b, 0xbc, 0x71, 0x71, 0x18, 0xfe, 0x0f, 0x8d, 0x48	**BS.InstallProtocolInterface – InstallProtocolInterface()** installs the protocol on an existing handle at **EFI_TPL_NOTIFY**	1. Call **InstallProtocolInterface()** to install the **TestProtocol1** as type **EFI_NATIVE_INTERFACE** onto an existing handle created by this function call. 2. Call **HandleProtocol()** to locate the protocol via the handle. The **TestProtocol1** should be located.

Number	GUID	Assertion	Test Description
5.1.3.1.32	0x1333f969, 0x957b, 0x4c96, 0x90, 0xaa, 0x06, 0x75, 0xa1, 0x61, 0x94, 0xaa	**BS.InstallProtocolInterface – InstallProtocolInterface()** installs the protocol on an existing handle at **EFI_TPL_APPLICATION**	1. Call **InstallProtocolInterface()** to install the **TestProtocol1** as type **EFI_NATIVE_INTERFACE** onto an existing handle created by this function call. 2. Call the **TestProtocol1**'s function. It should be accessed and be executed correctly.
5.1.3.1.33	0x913cbd44, 0xb381, 0x4f06, 0xbf, 0x94, 0x3d, 0xa5, 0xb0, 0x7f, 0x0d, 0xca	**BS.InstallProtocolInterface – InstallProtocolInterface()** installs the protocol on an existing handle at **EFI_TPL_CALLBACK**	1. Call **InstallProtocolInterface()** to install the **TestProtocol1** as type **EFI_NATIVE_INTERFACE** onto an existing handle created by this function call. 2. Call the **TestProtocol1**'s function. It should be accessed and be executed correctly.
5.1.3.1.34	0xf2709409, 0x4c81, 0x4942, 0xa0, 0x62, 0xdd, 0x61, 0x59, 0x63, 0x96, 0x61	**BS.InstallProtocolInterface – InstallProtocolInterface()** installs the protocol on an existing handle at **EFI_TPL_NOTIFY**	1. Call **InstallProtocolInterface()** to install the **TestProtocol1** as type **EFI_NATIVE_INTERFACE** onto an existing handle created by this function call. 2. Call the **TestProtocol1**'s function. It should be accessed and be executed correctly.
5.1.3.1.35	0x46858c39, 0x87f2, 0x444d, 0x85, 0x42, 0x48, 0xb3, 0xee, 0x60, 0xdb, 0x05	**BS.InstallProtocolInterface – InstallProtocolInterface()** returns **EFI_SUCCESS** with same protocol multiple times at **EFI_TPL_APPLICATION**	1. Call **InstallProtocolInterface()** 10 times to install the **TestProtocol1** onto 10 new handles. Each **InstallProtocolInterface()** return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.1.36	0x5470301a, 0x0e58, 0x4616, 0xa0, 0xd2, 0xce, 0xa8, 0x5f, 0x6e, 0x0b, 0x18	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with same protocol multiple times at `EFI_TPL_CALLBACK`	1. Call `InstallProtocolInterface()` 10 times to install the `TestProtocol1` onto 10 new handles. Each `InstallProtocolInterface()` return code should be `EFI_SUCCESS`.
5.1.3.1.37	0xe7417360, 0x2705, 0x4939, 0xa4, 0x86, 0x7c, 0xd9, 0x0d, 0x51, 0x4c, 0xb0	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with same protocol multiple times at `EFI_TPL_NOTIFY`	1. Call `InstallProtocolInterface()` 10 times to install the `TestProtocol1` onto 10 new handles. Each `InstallProtocolInterface()` return code should be `EFI_SUCCESS`.
5.1.3.1.38	0xde9471cf, 0xf547, 0x4940, 0x95, 0xbb, 0xb9, 0x06, 0x32, 0x54, 0xca, 0xa2	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with same protocol multiple times at `EFI_TPL_APPLICATION`	1. Call `InstallProtocolInterface()` 10 times to install the `TestProtocol1` onto 10 new handles. 10 new handles are created.
5.1.3.1.39	0xce8725eb, 0x40a8, 0x4ce2, 0x86, 0x27, 0x24, 0xe3, 0xd5, 0xfe, 0x8b, 0x72	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with same protocol multiple times at `EFI_TPL_CALLBACK`	1. Call `InstallProtocolInterface()` 10 times to install the `TestProtocol1` onto 10 new handles. 10 new handles are created.
5.1.3.1.40	0x735826c6, 0xa2b3, 0x457b, 0x88, 0x82, 0x39, 0x38, 0xcb, 0xbf, 0xf7, 0xad	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with same protocol multiple times at `EFI_TPL_NOTIFY`	1. Call `InstallProtocolInterface()` 10 times to install the `TestProtocol1` onto 10 new handles. 10 new handles are created.
5.1.3.1.41	0x4f7b61e8, 0x0777, 0x479c, 0xb3, 0x7d, 0x5b, 0xab, 0xa8, 0x2a, 0x17, 0x6c	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with same protocol multiple times at `EFI_TPL_APPLICATION`	1. Call `InstallProtocolInterface()` 10 times to install the `TestProtocol1` onto 10 new handles. 2. Call `LocateHandleBuffer()` to locate the handle via the protocol. 10 handles should be located.

Number	GUID	Assertion	Test Description
5.1.3.1.42	0xed0a8a40, 0x641f, 0x4abf, 0x9c, 0x0a, 0xae, 0xa0, 0x0e, 0xee, 0xde, 0xfb	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with same protocol multiple times at EFI_TPL_CALLBACK	1. Call InstallProtocolInterface() 10 times to install the TestProtocol1 onto 10 new handles. 2. Call LocateHandleBuffer() to locate the handle via the protocol. 10 handles should be located.
5.1.3.1.43	0x3e48a299, 0x11a8, 0x4f73, 0xb6, 0xe1, 0x40, 0x65, 0xf1, 0x8e, 0x68, 0x34	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with same protocol multiple times at EFI_TPL_NOTIFY	1. Call InstallProtocolInterface() 10 times to install the TestProtocol1 onto 10 new handles. 2. Call LocateHandleBuffer() to locate the handle via the protocol. 10 handles should be located.
5.1.3.1.44	0x2e596f06, 0x336a, 0x49a7, 0x88, 0x0e, 0x60, 0xd3, 0x68, 0x5a, 0x95, 0xa4	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with same protocol multiple times at EFI_TPL_APPLICATION	1. Call InstallProtocolInterface() 10 times to install the TestProtocol1 onto 10 new handles. 2. Call HandleProtocol() to locate the protocol via each handle. The TestProtocol1 should be located.
5.1.3.1.45	0x63a6ea07, 0xcd46, 0x40c8, 0x8a, 0x02, 0xb3, 0x36, 0xf9, 0x7d, 0x39, 0x33	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with same protocol multiple times at EFI_TPL_CALLBACK	1. Call InstallProtocolInterface() 10 times to install the TestProtocol1 onto 10 new handles. 2. Call HandleProtocol() to locate the protocol via each handle. The TestProtocol1 should be located.

Number	GUID	Assertion	Test Description
5.1.3.1.46	0x6096eff1, 0x21f0, 0x43cd, 0xb0, 0x8d, 0x88, 0xff, 0x3a, 0xd3, 0x9c, 0x28	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with same protocol multiple times at `EFI_TPL_NOTIFY`	1. Call `InstallProtocolInterface()` 10 times to install the `TestProtocol1` onto 10 new handles. 2. Call `HandleProtocol()` to locate the protocol via each handle. The `TestProtocol1` should be located.
5.1.3.1.47	0xd778b920, 0xe42b, 0x4901, 0xbc, 0x2c, 0x78, 0xea, 0x91, 0xb7, 0x91, 0xe5	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with same protocol multiple times at `EFI_TPL_APPLICATION`	1. Call `InstallProtocolInterface()` 10 times to install the `TestProtocol1` onto 10 new handles. 2. Call each `TestProtocol1`'s function. It should be accessed and be executed correctly.
5.1.3.1.48	0xf65a7dde, 0x7e46, 0x47aa, 0x9c, 0x88, 0x99, 0x5b, 0x69, 0x31, 0x24, 0x8b	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with same protocol multiple times at `EFI_TPL_CALLBACK`	1. Call `InstallProtocolInterface()` 10 times to install the `TestProtocol1` onto 10 new handles. 2. Call each `TestProtocol1`'s function. It should be accessed and be executed correctly.
5.1.3.1.49	0x06334e00, 0x03d2, 0x4406, 0x83, 0xb9, 0x66, 0x53, 0xb3, 0x41, 0x8a, 0x93	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with same protocol multiple times at `EFI_TPL_NOTIFY`	1. Call `InstallProtocolInterface()` 10 times to install the `TestProtocol1` onto 10 new handles. 2. Call each `TestProtocol1`'s function. It should be accessed and be executed correctly.

Number	GUID	Assertion	Test Description
5.1.3.1.50	0x4f229f4e, 0x64dc, 0x4a88, 0xb7, 0x77, 0xd2, 0x8d, 0xdf, 0x33, 0xac, 0x39	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with multiple protocols at `EFI_TPL_APPLICATION`	1. Call `InstallProtocolInterface()` 5 times to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3`, `TestProtocol4`, and `TestProtocol5` onto one new handle. Each `InstallProtocolInterface()` return code should be `EFI_SUCCESS`.
5.1.3.1.51	0x38deb65c, 0xf4db, 0x40c8, 0x9d, 0xea, 0xc0, 0xdf, 0xf9, 0xcc, 0x7a, 0x73	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with multiple protocols at `EFI_TPL_CALLBACK`	1. Call `InstallProtocolInterface()` 5 times to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3`, `TestProtocol4`, and `TestProtocol5` onto one new handle. Each `InstallProtocolInterface()` return code should be `EFI_SUCCESS`.
5.1.3.1.52	0x341714e5, 0xa4ce, 0x4f4a, 0x94, 0x54, 0x7b, 0xde, 0x9a, 0xb2, 0x14, 0x58	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with multiple protocols at `EFI_TPL_NOTIFY`	1. Call `InstallProtocolInterface()` 5 times to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3`, `TestProtocol4`, and `TestProtocol5` onto one new handle. Each `InstallProtocolInterface()` return code should be `EFI_SUCCESS`.
5.1.3.1.53	0x1eb05a66, 0x3ded, 0x440e, 0xa6, 0xcf, 0x72, 0x05, 0x62, 0x21, 0x48, 0xe0	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with multiple protocols at `EFI_TPL_APPLICATION`	1. Call `InstallProtocolInterface()` 5 times to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3`, `TestProtocol4`, and `TestProtocol5` onto one new handle. The new handle should be created.

Number	GUID	Assertion	Test Description
5.1.3.1.54	0x0133559d, 0x4a88, 0x41d0, 0x8b, 0x32, 0x6b, 0x87, 0x24, 0xd0, 0xcc, 0xcb	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with multiple protocols at EFI_TPL_CALLBACK	1. Call InstallProtocolInterface() 5 times to install TestProtocol1, TestProtocol2, TestProtocol3, TestProtocol4, and TestProtocol5 onto one new handle. The new handle should be created.
5.1.3.1.55	0x16ce2f4e, 0xc303, 0x49f6, 0x89, 0x94, 0x26, 0x19, 0xfd, 0x4b, 0x67, 0xf8	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with multiple protocols at EFI_TPL_NOTIFY	1. Call InstallProtocolInterface() 5 times to install TestProtocol1, TestProtocol2, TestProtocol3, TestProtocol4, and TestProtocol5 onto one new handle. The new handle should be created.
5.1.3.1.56	0x280062c1, 0x1685, 0x4307, 0x95, 0xca, 0x12, 0x07, 0x38, 0x2c, 0x0d, 0xa0	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with multiple protocols at EFI_TPL_APPLICATION	1. Call InstallProtocolInterface() 5 times to install TestProtocol1, TestProtocol2, TestProtocol3, TestProtocol4, and TestProtocol5 onto one new handle. 2. Call LocateHandleBuffer() to locate the handle via each protocol. The new handles should be located.
5.1.3.1.57	0x3b119ca5, 0x8c66, 0x4158, 0xb6, 0x8c, 0xb9, 0x43, 0x81, 0x97, 0x77, 0xdc	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with multiple protocols at EFI_TPL_CALLBACK	1. Call InstallProtocolInterface() 5 times to install TestProtocol1, TestProtocol2, TestProtocol3, TestProtocol4, and TestProtocol5 onto one new handle. 2. Call LocateHandleBuffer() to locate the handle via each protocol.The new handles should be located.

Number	GUID	Assertion	Test Description
5.1.3.1.58	0x57b88782, 0x960e, 0x4aaf, 0xbf, 0xef, 0xc9, 0xbf, 0xf1, 0xe0, 0x9c, 0x6d	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with multiple protocols at `EFI_TPL_NOTIFY`	1. Call `InstallProtocolInterface()` 5 times to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3`, `TestProtocol4`, and `TestProtocol5` onto one new handle. 2. Call `LocateHandleBuffer()` to locate the handle via each protocol. The new handles should be located.
5.1.3.1.59	0x6b85ed1e, 0x287d, 0x46d2, 0xa0, 0x36, 0x7c, 0x53, 0xfa, 0x24, 0xab, 0x75	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with multiple protocols at `EFI_TPL_APPLICATION`	1. Call `InstallProtocolInterface()` 5 times to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3`, `TestProtocol4`, and `TestProtocol5` onto one new handle. 2. Call `HandleProtocol()` to locate the protocol via the handle. All protocols should be located.
5.1.3.1.60	0x71f094cd, 0x53fd, 0x4ff7, 0x95, 0xd7, 0x1b, 0x8e, 0x97, 0x26, 0x92, 0xb0	`BS.InstallProtocolInterface – InstallProtocolInterface()` returns `EFI_SUCCESS` with multiple protocols at `EFI_TPL_CALLBACK`	1. Call `InstallProtocolInterface()` 5 times to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3`, `TestProtocol4`, and `TestProtocol5` onto one new handle. 2. Call `HandleProtocol()` to locate the protocol via the handle. All protocols should be located.

Number	GUID	Assertion	Test Description
5.1.3.1.61	0x064740c2, 0xccce, 0x45f5, 0xbb, 0x37, 0xd4, 0xd0, 0xe1, 0x66, 0x8d, 0x8c	**BS.InstallProtocolInterface – InstallProtocolInterface()** returns **EFI_SUCCESS** with multiple protocols at **EFI_TPL_NOTIFY**	1. Call **InstallProtocolInterface()** 5 times to install **TestProtocol1**, **TestProtocol2**, **TestProtocol3**, **TestProtocol4**, and **TestProtocol5** onto one new handle. 2. Call **HandleProtocol()** to locate the protocol via the handle. All protocols should be located.
5.1.3.1.62	0x2f94a7ec, 0x4d30, 0x4572, 0xbc, 0x3b, 0x87, 0xc9, 0x26, 0x99, 0x53, 0x8d	**BS.InstallProtocolInterface – InstallProtocolInterface()** returns **EFI_SUCCESS** with **NULL** interface at **EFI_TPL_APPLICATION**	1. Call **InstallProtocolInterface()** to install **TestProtocolNoInterface1** to a new handle. **InstallProtocolInterface()** return code should be **EFI_SUCCESS**.
5.1.3.1.63	0x382cee61, 0xb25c, 0x43a1, 0xb2, 0xde, 0x07, 0x27, 0x37, 0xc6, 0x79, 0xf5	**BS.InstallProtocolInterface – InstallProtocolInterface()** returns **EFI_SUCCESS** with **NULL** interface at **EFI_TPL_CALLBACK**	1. Call **InstallProtocolInterface()** to install **TestProtocolNoInterface1** to a new handle. **InstallProtocolInterface()** return code should be **EFI_SUCCESS**.
5.1.3.1.64	0xc58b2515, 0xe066, 0x4a2f, 0x97, 0x5c, 0x7f, 0x80, 0x00, 0x73, 0x3e, 0xf3	**BS.InstallProtocolInterface – InstallProtocolInterface()** returns **EFI_SUCCESS** with **NULL** interface at **EFI_TPL_NOTIFY**	1. Call **InstallProtocolInterface()** to install **TestProtocolNoInterface1** to a new handle. **InstallProtocolInterface()** return code should be **EFI_SUCCESS**.
5.1.3.1.65	0x1b223dc2, 0x5d17, 0x40e1, 0x93, 0x99, 0x3c, 0x45, 0xf0, 0xe4, 0xf8, 0x88	**BS.InstallProtocolInterface – InstallProtocolInterface()** returns **EFI_SUCCESS** with **NULL** interface at **EFI_TPL_APPLICATION**	1. Call **InstallProtocolInterface()** to install **TestProtocolNoInterface1** to a new handle. The new handle should be created.

Number	GUID	Assertion	Test Description
5.1.3.1.66	0x6b039e16, 0x5420, 0x4520, 0x85, 0x25, 0xb9, 0xbd, 0x5a, 0x3c, 0x22, 0x66	**BS.InstallProtocolInterface – InstallProtocolInterface()** returns **EFI_SUCCESS** with **NULL** interface at **EFI_TPL_CALLBACK**	1. Call **InstallProtocolInterface()** to install **TestProtocolNoInterface1** to a new handle. The new handle should be created.
5.1.3.1.67	0x763a4629, 0x18ec, 0x41b3, 0x9f, 0xa6, 0x4a, 0xc6, 0x4e, 0x44, 0x8b, 0x49	**BS.InstallProtocolInterface – InstallProtocolInterface()** returns **EFI_SUCCESS** with **NULL** interface at **EFI_TPL_NOTIFY**	1. Call **InstallProtocolInterface()** to install **TestProtocolNoInterface1** to a new handle. The new handle should be created.
5.1.3.1.68	0xa366c643, 0xeac3, 0x4994, 0xbe, 0xe5, 0x6c, 0x6f, 0xf5, 0xb8, 0x3f, 0x5e	**BS.InstallProtocolInterface – InstallProtocolInterface()** returns **EFI_SUCCESS** with **NULL** interface at **EFI_TPL_APPLICATION**	1. Call **InstallProtocolInterface()** to install **TestProtocolNoInterface1** to a new handle. 2. Call **LocateHandleBuffer()** to locate the handle via the protocol.The new handles should be located.
5.1.3.1.69	0xaf59a8ed, 0x144b, 0x48b5, 0x88, 0x0f, 0xa2, 0x20, 0x0a, 0xf0, 0x4a, 0xcd	**BS.InstallProtocolInterface – InstallProtocolInterface()** returns **EFI_SUCCESS** with **NULL** interface at **EFI_TPL_CALLBACK**	1. Call **InstallProtocolInterface()** to install **TestProtocolNoInterface1** to a new handle. 2. Call **LocateHandleBuffer()** to locate the handle via the protocol.The new handles should be located.
5.1.3.1.70	0xfec89489, 0x0c0d, 0x493b, 0xa5, 0x4d, 0x94, 0xf7, 0x15, 0x04, 0xe9, 0x32	**BS.InstallProtocolInterface – InstallProtocolInterface()** returns **EFI_SUCCESS** with **NULL** interface at **EFI_TPL_NOTIFY**	1. Call **InstallProtocolInterface()** to install **TestProtocolNoInterface1** to a new handle. 2. Call **LocateHandleBuffer()** to locate the handle via the protocol.The new handles should be located.

Number	GUID	Assertion	Test Description
5.1.3.1.71	0xa94c8ad5, 0xc578, 0x45f6, 0x9d, 0x5c, 0xcb, 0x15, 0x62, 0x65, 0xe6, 0x72	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with NULL interface at EFI_TPL_APPLICATION	1. Call InstallProtocolInterface() to install TestProtocolNoInterface1 to a new handle. 2. Call HandleProtocol() to locate the protocol via the handle. The TestProtocolNoInterface1 should be located.
5.1.3.1.72	0xfccbcf28, 0xc207, 0x440a, 0xbb, 0xa0, 0x0e, 0x43, 0xc4, 0xc1, 0xb4, 0xa0	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with NULL interface at EFI_TPL_CALLBACK	1. Call InstallProtocolInterface() to install TestProtocolNoInterface1 to a new handle. 2. Call HandleProtocol() to locate the protocol via the handle. The TestProtocolNoInterface1 should be located.
5.1.3.1.73	0x67a70da1, 0x8211, 0x4d76, 0xa0, 0x2c, 0xf8, 0x64, 0xb1, 0x99, 0x92, 0x94	BS.InstallProtocolInterface – InstallProtocolInterface() returns EFI_SUCCESS with NULL interface at EFI_TPL_NOTIFY	1. Call InstallProtocolInterface() to install TestProtocolNoInterface1 to a new handle. 2. Call HandleProtocol() to locate the protocol via the handle. The TestProtocolNoInterface1 should be located.

3.3.2 UninstallProtocolInterface()

Number	GUID	Assertion	Test Description
5.1.3.2.1	0x9646236e, 0x0603, 0x488e, 0x91, 0x16, 0x83, 0x4f, 0x76, 0xfa, 0x06, 0x5c	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_INVALID_PARAMETER` with *`Protocol`* is `NULL`	1. Call `UninstallProtocolInterface()` with the protocol GUID value of `NULL`. The return code must be `EFI_INVALID_PARAMETER`.
5.1.3.2.2	0x3647da0d, 0x50a1, 0x4800, 0xbe, 0x24, 0xc1, 0xb5, 0x84, 0x20, 0xcf, 0xf4	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_INVALID_PARAMETER` with invalid handle	1. Call `UninstallProtocolInterface()` with an invalid handle (*`Handle`* = `NULL` or *`Handle`* is invalid). Each return code must be `EFI_INVALID_PARAMETER`.
5.1.3.2.3	0x696cd520, 0x897e, 0x4e91, 0xa7, 0xd8, 0x3e, 0xfd, 0xa1, 0x83, 0xc1, 0x12	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_NOT_FOUND` with a non-existent protocol	1. Call `UninstallProtocolInterface()` to attempt to uninstall a non-existent protocol from a handle. The return code must be `EFI_NOT_FOUND`.
5.1.3.2.4	0xe41a6aac, 0xa293, 0x499a, 0xbe, 0xb9, 0x40, 0xa2, 0x95, 0x36, 0x72, 0xac	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_NOT_FOUND` with invalid interface	1. Call `UninstallProtocolInterface()` to attempt to uninstall a protocol from a handle with an invalid interface. The return code must be `EFI_NOT_FOUND`.
5.1.3.2.5	0x3c7352fc, 0xca03, 0x493b, 0x8e, 0x87, 0x89, 0x0d, 0xcd, 0x4d, 0xfa, 0x1a	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls non-opened protocol at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.2.6	0xb29effa0, 0xdd3d, 0x4585, 0x80, 0xff, 0xe3, 0x1d, 0xad, 0x9f, 0xa6, 0x4c	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls non-opened protocol at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto a new handle. 2. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The return code should be EFI_SUCCESS.
5.1.3.2.7	0x7625c205, 0x42d3, 0x408b, 0x97, 0x76, 0x87, 0x58, 0xae, 0xdf, 0xa8, 0xce	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls non-opened protocol at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The return code should be EFI_SUCCESS.
5.1.3.2.8	0xe4b8f72f, 0xd72b, 0x47ce, 0x8f, 0x07, 0x73, 0x5f, 0xad, 0x79, 0xfa, 0xec	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls non-opened protocol at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The handle should still exist.
5.1.3.2.9	0xb92ffcbc, 0x45c0, 0x454e, 0xa5, 0x64, 0xea, 0x4a, 0xd0, 0x35, 0xe2, 0x11	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls non-opened protocol at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto a new handle. 2. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The handle should still exist.
5.1.3.2.10	0x7c01d7d3, 0x1ec6, 0x4550, 0x92, 0xbf, 0x58, 0xba, 0xe6, 0x08, 0xd6, 0x41	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls non-opened protocol at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto a new handle. 2. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The handle should still exist.

Number	GUID	Assertion	Test Description
5.1.3.2.11	0x563401ca, 0x9fb4, 0x4ded, 0x88, 0x84, 0xbd, 0x0d, 0xee, 0xb7, 0x77, 0xea	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls non-opened protocol at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** & **TestProtocol2** onto a new handle. 2. Call **UninstallProtocolInterface()** to remove **TestProtocol1** from the handle. 3. Call **LocateHandleBuffer()** to locate the handle via **TestProtocol1**. The return code should be **EFI_NOT_FOUND**.
5.1.3.2.12	0xa5ffafa1, 0x672e, 0x4c49, 0x9a, 0xb6, 0x93, 0xc3, 0x3f, 0xe4, 0x6f, 0x2e	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls non-opened protocol at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto a new handle. 2. Call **UninstallProtocolInterface()** to remove **TestProtocol1** from the handle. 3. Call **LocateHandleBuffer()** to locate the handle via **TestProtocol1**. The return code should be **EFI_NOT_FOUND**.
5.1.3.2.13	0x5e71353f, 0x4c05, 0x4205, 0xbe, 0xfa, 0x14, 0xa8, 0x5b, 0xc1, 0xf0, 0xf9	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls non-opened protocol at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto a new handle. 2. Call **UninstallProtocolInterface()** to remove **TestProtocol1** from the handle. 3. Call **LocateHandleBuffer()** to locate the handle via **TestProtocol1**. The return code should be **EFI_NOT_FOUND**.

Number	GUID	Assertion	Test Description
5.1.3.2.14	0xac16ea87, 0x9311, 0x4cb0, 0xaa, 0xf5, 0x96, 0x0e, 0x24, 0xd4, 0xa8, 0xf4	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls non-opened protocol at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 3. Call `HandleProtocol()` to locate `TestProtocol1` via the handle. The protocol should no longer exist.
5.1.3.2.15	0xc805ddbb, 0xbefe, 0x45aa, 0x94, 0x52, 0xb2, 0x48, 0xd8, 0xb9, 0xe4, 0x6e	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls non-opened protocol at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 3. Call `HandleProtocol()` to locate `TestProtocol1` via the handle. The protocol should no longer exist.
5.1.3.2.16	0x1a828703, 0x32a5, 0x481a, 0x8c, 0xdd, 0x22, 0xb0, 0x20, 0x51, 0xe1, 0x50	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls non-opened protocol at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 3. Call `HandleProtocol()` to locate `TestProtocol1` via the handle. The protocol should no longer exist.

Number	GUID	Assertion	Test Description
5.1.3.2.17	0x53756d94, 0xc5c0, 0x47ad, 0x8a, 0x89, 0xa9, 0x86, 0x07, 0xd2, 0x31, 0x8c	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls non-opened protocol at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 3. Call `HandleProtocol()` to locate `TestProtocol2` via the handle. The protocol should still exist.
5.1.3.2.18	0xbe257dd2, 0xe51d, 0x40be, 0x99, 0x8b, 0xec, 0xbd, 0x09, 0x27, 0x22, 0x96	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls non-opened protocol at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 3. Call `HandleProtocol()` to locate `TestProtocol2` via the handle. The protocol should still exist.
5.1.3.2.19	0x8c2b696c, 0x87b0, 0x4a82, 0x8b, 0x87, 0x07, 0xfb, 0x0e, 0x89, 0x57, 0x43	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls non-opened protocol at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 3. Call `HandleProtocol()` to locate `TestProtocol2` via the handle. The protocol should still exist.

Number	GUID	Assertion	Test Description
5.1.3.2.20	0x1f991bf6, 0x05a2, 0x4858, 0xa4, 0x71, 0x79, 0x2e, 0xf5, 0x0b, 0xab, 0xd9	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls all non-opened protocols at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto a new handle. 2. Call UninstallProtocolInterface() to remove TestProtocol1 and TestProtocol2 from the handle. The return code should be EFI_SUCCESS.
5.1.3.2.21	0x836e62c9, 0x2d3b, 0x4c55, 0xb8, 0xd9, 0x94, 0x3a, 0xee, 0x99, 0xbe, 0x3b	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls all non-opened protocols at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto a new handle. 2. Call UninstallProtocolInterface() to remove TestProtocol1 and TestProtocol2 from the handle. The return code should be EFI_SUCCESS.
5.1.3.2.22	0xe95e5e34, 0x1ee6, 0x4e71, 0xa0, 0x39, 0x6e, 0x61, 0x71, 0x75, 0xb1, 0x3d	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls all non-opened protocols at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto a new handle. 2. Call UninstallProtocolInterface() to remove TestProtocol1 and TestProtocol2 from the handle. The return code should be EFI_SUCCESS.
5.1.3.2.23	0x3acc0c56, 0x0b26, 0x4612, 0x8e, 0xd4, 0x23, 0x01, 0x80, 0xde, 0xa9, 0x86	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls all non-opened protocols at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto a new handle. 2. Call UninstallProtocolInterface() to remove TestProtocol1 and TestProtocol2 from the handle. The handle should no longer exist.

Number	GUID	Assertion	Test Description
5.1.3.2.24	0x7eb03eb1, 0x9159, 0x4b52, 0x83, 0x6c, 0x60, 0xd1, 0xc6, 0x52, 0x10, 0xe3	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls all non-opened protocols at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto a new handle. 2. Call **UninstallProtocolInterface()** to remove **TestProtocol1** and **TestProtocol2** from the handle. The handle should no longer exist.
5.1.3.2.25	0x7b201d9e, 0x296a, 0x4a39, 0xa0, 0xfe, 0xed, 0x34, 0xb4, 0x69, 0x3e, 0xdf	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls all non-opened protocols at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto a new handle. 2. Call **UninstallProtocolInterface()** to remove **TestProtocol1** and **TestProtocol2** from the handle. The handle should no longer exist.
5.1.3.2.26	0x7dcb87f6, 0x5522, 0x4a4f, 0x8d, 0xe5, 0xfa, 0xc8, 0x0b, 0x5d, 0x03, 0x09	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls all non-opened protocols at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** & **TestProtocol2** onto a new handle. 2. Call **UninstallProtocolInterface()** to remove **TestProtocol1** and **TestProtocol2** from the handle. 3. Call **LocateHandleBuffer()** to locate the handle via **TestProtocol1**. The return code should be **EFI_NOT_FOUND**.

Number	GUID	Assertion	Test Description
5.1.3.2.27	0x49ab9ed1, 0xf041, 0x42d4, 0xbf, 0x48, 0x46, 0x1b, 0x04, 0x78, 0x4c, 0xa8	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls all non-opened protocols at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` and `TestProtocol2` from the handle. 3. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol1`. The return code should be `EFI_NOT_FOUND`.
5.1.3.2.28	0x136369f3, 0x766a, 0x4a90, 0xa5, 0xcb, 0x8d, 0xb3, 0x0e, 0x83, 0x71, 0x82	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls all non-opened protocols at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` and `TestProtocol2` from the handle. 3. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol1`. The return code should be `EFI_NOT_FOUND`.
5.1.3.2.29	0x28db37d6, 0xdf2d, 0x4fbe, 0x8a, 0x14, 0xbb, 0x06, 0x90, 0xc3, 0x99, 0xfd	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls all non-opened protocols at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` and `TestProtocol2` from the handle. 3. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol2`. The return code should be `EFI_NOT_FOUND`.

Number	GUID	Assertion	Test Description
5.1.3.2.30	0xcc1b25a6, 0x0268, 0x443f, 0xa0, 0x6f, 0xd8, 0x4c, 0x79, 0x28, 0xdd, 0x4c	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls all non-opened protocols at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` and `TestProtocol2` from the handle. 3. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol2`. The return code should be `EFI_NOT_FOUND`.
5.1.3.2.31	0x1259358c, 0xf63b, 0x4f87, 0xa7, 0x3f, 0x5b, 0x46, 0x34, 0xa5, 0x7f, 0x53	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls all non-opened protocols at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocol1` and `TestProtocol2` from the handle. 3. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol2`. The return code should be `EFI_NOT_FOUND`.
5.1.3.2.32	0x901ab829, 0xeec3, 0x4560, 0xb4, 0xa0, 0x68, 0x85, 0x77, 0x4a, 0x82, 0xa1	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.2.33	0x99f7dd6a, 0xa50d, 0x4849, 0xb0, 0x44, 0xcb, 0xe9, 0xa6, 0x94, 0xb6, 0xde	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.2.34	0xf0de7d9f, 0x858b, 0x4cb3, 0x81, 0xa0, 0xfe, 0xa6, 0xa3, 0x8f, 0xad, 0xd7	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.2.35	0xee7df286, 0x3936, 0x4122, 0x88, 0x88, 0x45, 0x9a, 0x9c, 0x84, 0x81, 0x73	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The handle should no longer exist.
5.1.3.2.36	0x23f14ed9, 0xffe9, 0x440c, 0xb3, 0xf5, 0x62, 0x44, 0xd1, 0x6d, 0xcc, 0x91	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The handle should no longer exist.

Number	GUID	Assertion	Test Description
5.1.3.2.37	0xdbf315df, 0x30cf, 0x4814, 0x84, 0xa6, 0x07, 0x16, 0x59, 0x4a, 0x18, 0xca	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls opened **BY_HANDLE_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_HANDLE_PROTOCOL**. 3. Call **UninstallProtocolInterface()** to remove **TestProtocol1** from the handle. The handle should no longer exist.
5.1.3.2.38	0x5ccc9c7c, 0xbbad, 0x4faa, 0xa1, 0x98, 0x45, 0x1d, 0xfb, 0x4c, 0xd1, 0xbb	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls opened **BY_HANDLE_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_HANDLE_PROTOCOL**. 3. Call **UninstallProtocolInterface()** to remove **TestProtocol1** from the handle. 4. Call **LocateHandleBuffer()** to locate the handle via **TestProtocol1**. The return code should be **EFI_NOT_FOUND**.
5.1.3.2.39	0x95ead6e8, 0x5e59, 0x47ca, 0x8d, 0xb4, 0x10, 0x4d, 0x2a, 0x36, 0x19, 0xf3	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls opened **BY_HANDLE_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_HANDLE_PROTOCOL**. 3. Call **UninstallProtocolInterface()** to remove **TestProtocol1** from the handle. 4. Call **LocateHandleBuffer()** to locate the handle via **TestProtocol1**. The return code should be **EFI_NOT_FOUND**.

Number	GUID	Assertion	Test Description
5.1.3.2.40	0x77e117af, 0x92ee, 0x48db, 0x9c, 0x32, 0xf2, 0xf6, 0xb4, 0x63, 0x2a, 0xcc	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol1`. The return code should be `EFI_NOT_FOUND`.
5.1.3.2.41	0xcb7b8fcd, 0xd0dd, 0x4d78, 0xa9, 0x6c, 0xc7, 0x52, 0xf1, 0x93, 0x21, 0xfd	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `GET_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.2.42	0x7d01a157, 0x98ea, 0x4120, 0xb0, 0xec, 0xcf, 0x9c, 0xa7, 0x59, 0x2b, 0xf5	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `GET_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.2.43	0x05a40340, 0xcc89, 0x4162, 0xa2, 0x94, 0xcd, 0xd9, 0x97, 0x86, 0x1d, 0xe3	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened GET_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The return code should be EFI_SUCCESS.
5.1.3.2.44	0x3f7d45dd, 0x400e, 0x4b39, 0x94, 0xba, 0xa4, 0x61, 0xa7, 0xb0, 0xbb, 0x1b	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened GET_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The handle should no longer exist.
5.1.3.2.45	0xee9f6130, 0xc1e3, 0x4207, 0x8b, 0x95, 0x7e, 0xa2, 0x5e, 0xf1, 0xa1, 0xa1	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened GET_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The handle should no longer exist.
5.1.3.2.46	0x76b0500e, 0x7f2d, 0x4eac, 0xa6, 0xbc, 0xc0, 0xb9, 0x29, 0x5b, 0xb0, 0x54	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened GET_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The handle should no longer exist.

Number	GUID	Assertion	Test Description
5.1.3.2.47	0xda2360cc, 0x9a59, 0x485f, 0xb2, 0xc6, 0xeb, 0x00, 0x93, 0xfc, 0x51, 0x30	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened GET_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The return code should be EFI_NOT_FOUND.
5.1.3.2.48	0x84c0acce, 0xca54, 0x44da, 0x85, 0xd6, 0x40, 0x0a, 0x8c, 0x62, 0xbf, 0x37	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened GET_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The return code should be EFI_NOT_FOUND.
5.1.3.2.49	0xaa72ce83, 0x0ba4, 0x4f47, 0x9f, 0xb3, 0x5d, 0xb2, 0x35, 0x93, 0x88, 0x5e	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened GET_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The return code should be EFI_NOT_FOUND.

Number	GUID	Assertion	Test Description
5.1.3.2.50	0x7c9eede7, 0x9881, 0x42f8, 0x94, 0xa5, 0x53, 0xf7, 0xf2, 0x7f, 0x95, 0xb3	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls opened **TEST_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 TEST_PROTOCOL**. 3. Call **UninstallProtocolInterface()** to remove **TestProtocol1** from the handle. The return code should be **EFI_SUCCESS**.
5.1.3.2.51	0x54c4db30, 0x7115, 0x418b, 0xa4, 0x9e, 0x4c, 0x4d, 0x32, 0xde, 0xa6, 0xf9	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls opened **TEST_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 TEST_PROTOCOL**. 3. Call **UninstallProtocolInterface()** to remove **TestProtocol1** from the handle. The return code should be **EFI_SUCCESS**.
5.1.3.2.52	0x61d1b5cf, 0x4efe, 0x4b26, 0xaa, 0x3b, 0x35, 0x04, 0x07, 0xa5, 0xb6, 0xd3	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls opened **TEST_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 TEST_PROTOCOL**. 3. Call **UninstallProtocolInterface()** to remove **TestProtocol1** from the handle. The return code should be **EFI_SUCCESS**.
5.1.3.2.53	0xc6b5cfbc, 0x3814, 0x47ff, 0x9a, 0xec, 0x81, 0x91, 0x0b, 0xb0, 0x34, 0x48	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls opened **TEST_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 TEST_PROTOCOL**. 3. Call **UninstallProtocolInterface()** to remove **TestProtocol1** from the handle. The handle should no longer exist.

Number	GUID	Assertion	Test Description
5.1.3.2.54	0xd18c3a3a, 0x8022, 0x42e6, 0x9c, 0x6b, 0x6d, 0x65, 0x9b, 0x4b, 0xa9, 0xb7	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened TEST_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The handle should no longer exist.
5.1.3.2.55	0x7090235f, 0x6049, 0x44c1, 0xaf, 0x6c, 0xdb, 0x7c, 0xee, 0x9b, 0xf5, 0x95	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened TEST_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The handle should no longer exist.
5.1.3.2.56	0x8d82ba65, 0x9de9, 0x4081, 0xaf, 0xc2, 0x8f, 0xcb, 0x87, 0x14, 0x20, 0x18	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened TEST_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The return code should be EFI_NOT_FOUND.

Number	GUID	Assertion	Test Description
5.1.3.2.57	0xf327f4a3, 0xa3b1, 0x453f, 0x8a, 0x32, 0xe3, 0x21, 0x54, 0xfb, 0xbc, 0x5a	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened TEST_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The return code should be EFI_NOT_FOUND.
5.1.3.2.58	0x2d41eabb, 0xd34e, 0x45c6, 0x87, 0xae, 0xbe, 0xdc, 0xb3, 0x21, 0x67, 0x29	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened TEST_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The return code should be EFI_NOT_FOUND.
5.1.3.2.59	0x6b7d19b4, 0x34cc, 0x4595, 0xb3, 0x1e, 0x03, 0xb2, 0x5c, 0x7a, 0xe1, 0x29	BS.UninstallProtocolInterface – UninstallProtocolInterface() returns EFI_ACCESS_DENIED to uninstall opened BY_CHILD_CONTROLLER at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.2.60	0x7a710244, 0xe5d4, 0x46a9, 0x89, 0x19, 0x0e, 0x57, 0x88, 0xd3, 0x3b, 0x0b	BS.UninstallProtocolInterface – UninstallProtocolInterface() returns EFI_ACCESS_DENIED to uninstall opened BY_CHILD_CONTROLLER at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The return code should be EFI_ACCESS_DENIED.
5.1.3.2.61	0x866401d9, 0x9f44, 0x4af9, 0x8a, 0x45, 0x64, 0x85, 0xe7, 0x7e, 0xb2, 0x6b	BS.UninstallProtocolInterface – UninstallProtocolInterface() returns EFI_ACCESS_DENIED to uninstall opened BY_CHILD_CONTROLLER at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The return code should be EFI_ACCESS_DENIED.
5.1.3.2.62	0xc5b4e393, 0x052a, 0x4abe, 0xa6, 0x44, 0x63, 0x6e, 0x83, 0xab, 0x98, 0x86	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened BY_CHILD_CONTROLLER at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The handle should still exist.
5.1.3.2.63	0x4cfacc16, 0x447d, 0x4e8f, 0xae, 0xb9, 0x24, 0x39, 0xfb, 0xbe, 0xd3, 0xe0	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened BY_CHILD_CONTROLLER at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The handle should still exist.

Number	GUID	Assertion	Test Description
5.1.3.2.64	0xf9867e6a, 0xec14, 0x43f5, 0x81, 0xab, 0x46, 0xd0, 0x4b, 0x02, 0xd0, 0xdc	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened BY_CHILD_CONTROLLER at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. The handle should still exist.
5.1.3.2.65	0xdb2edcbc, 0x6c27, 0x4d27, 0xae, 0xf0, 0x90, 0x86, 0x73, 0xd3, 0x38, 0x90	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened BY_CHILD_CONTROLLER at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The protocol should still exist.
5.1.3.2.66	0x1af6079a, 0x20b8, 0x470f, 0xba, 0x7b, 0x75, 0x17, 0xf0, 0xd2, 0x77, 0x12	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened BY_CHILD_CONTROLLER at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The protocol should still exist.

Number	GUID	Assertion	Test Description
5.1.3.2.67	0xb5178b36, 0xa886, 0x427a, 0xa6, 0x6d, 0x8a, 0x9e, 0xa4, 0xf1, 0x37, 0x43	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol1`. The protocol should still exist.
5.1.3.2.68	0xe21dae05, 0xad6a, 0x4a49, 0xbc, 0xf0, 0xfb, 0xaa, 0x3a, 0xa3, 0xb4, 0x1c	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.2.69	0x4aca3c71, 0x0a1a, 0x421d, 0xb8, 0x86, 0xcd, 0x8f, 0x20, 0x08, 0x94, 0x58	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle again. The return code should be `EFI_SUCCESS`.
5.1.3.2.70	0xe3622cc4, 0x828e, 0x4dbd, 0xbd, 0xf6, 0x4a, 0x60, 0xb5, 0x79, 0x73, 0x6e	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.2.71	0x7fae8711, 0xf023, 0x4193, 0x9c, 0x6e, 0xab, 0x92, 0x7a, 0x2a, 0x9f, 0x74	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_ACCESS_DENIED` to uninstall opened `EXCLUSIVE` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.2.72	0x5b031e9c, 0xcc65, 0x4638, 0xb7, 0x4d, 0xd0, 0x3e, 0x4a, 0xea, 0xd3, 0x22	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_ACCESS_DENIED` to uninstall opened `EXCLUSIVE` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.2.73	0x7d0240a7, 0xe3dd, 0x4066, 0x8e, 0x56, 0x15, 0x03, 0xc0, 0x17, 0x9d, 0x22	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_ACCESS_DENIED` to uninstall opened `EXCLUSIVE` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.2.74	0x419755bd, 0xdcf7, 0x46fd, 0xb8, 0x82, 0x73, 0x89, 0x3e, 0xb0, 0x13, 0x79	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `EXCLUSIVE` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The handle should still exist.

Number	GUID	Assertion	Test Description
5.1.3.2.75	0x049261e7, 0x0fcb, 0x4861, 0x9d, 0x54, 0x0b, 0x08, 0x41, 0x8b, 0x4e, 0x2b	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `EXCLUSIVE` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The handle should still exist.
5.1.3.2.76	0x8d6d3a66, 0x1778, 0x4b2e, 0xb0, 0x20, 0x6d, 0xa0, 0x5d, 0xa8, 0x14, 0x9d	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `EXCLUSIVE` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The handle should still exist.
5.1.3.2.77	0x47b3ab81, 0xbdcc, 0x435b, 0xbd, 0xbc, 0x99, 0xf5, 0x79, 0x4a, 0x04, 0xbd	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `EXCLUSIVE` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol1`. The protocol should still exist.

Number	GUID	Assertion	Test Description
5.1.3.2.78	0xe6ffc0cf, 0xf8e4, 0x44db, 0x8c, 0xec, 0x8f, 0x68, 0x9b, 0xf4, 0xf6, 0xfe	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened EXCLUSIVE at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The protocol should still exist.
5.1.3.2.79	0x29b13f82, 0x3ab3, 0x4f47, 0xbe, 0xa5, 0x0a, 0x87, 0xa5, 0x95, 0x2e, 0xc1	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened EXCLUSIVE at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The protocol should still exist.
5.1.3.2.80	0x438a4fbf, 0xd811, 0x4082, 0xad, 0x01, 0xe1, 0x7c, 0x24, 0x03, 0x11, 0x1f	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened EXCLUSIVE at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call CloseProtocol() to close TestProtocol1. 5. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle again. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.2.81	0xcfb6aa7a, 0xb91a, 0x45c1, 0x81, 0x8f, 0xc5, 0x53, 0x0b, 0x01, 0xc0, 0xe5	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `EXCLUSIVE` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle again. The return code should be `EFI_SUCCESS`.
5.1.3.2.82	0x09efb83c, 0x0d16, 0x4a0b, 0xa7, 0x0b, 0xbc, 0x31, 0x64, 0xc8, 0x69, 0xb1	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `EXCLUSIVE` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.2.83	0x9afa33ae, 0x22ea, 0x45f8, 0xba, 0x79, 0x39, 0x14, 0xff, 0x96, 0x2b, 0xf0	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_ACCESS_DENIED` to uninstall opened `BY_DRIVER` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.2.84	0x571996c7, 0x12cc, 0x47b5, 0xbc, 0xab, 0x86, 0xe9, 0x39, 0x92, 0x84, 0xbe	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_ACCESS_DENIED` to uninstall opened `BY_DRIVER` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.2.85	0x8af64391, 0x81c3, 0x436d, 0xa3, 0xbc, 0xbe, 0x5e, 0x87, 0xe4, 0x6a, 0xbb	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_ACCESS_DENIED` to uninstall opened `BY_DRIVER` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.2.86	0x0fdd4f9a, 0xc2ee, 0x4ae4, 0x86, 0x64, 0x33, 0x9b, 0x5b, 0xf5, 0xe7, 0xbe	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_DRIVER` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The handle should still exist.

Number	GUID	Assertion	Test Description
5.1.3.2.87	0x14a00be5, 0x7cd5, 0x4a85, 0x87, 0xd9, 0x26, 0xb5, 0xf9, 0x52, 0xdf, 0x57	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_DRIVER` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The handle should still exist.
5.1.3.2.88	0x910a91ef, 0x5905, 0x48fd, 0xa3, 0x2f, 0xfa, 0x7e, 0xa2, 0x89, 0xab, 0xa8	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_DRIVER` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The handle should still exist.
5.1.3.2.89	0x9fb2b08f, 0xe896, 0x41f0, 0xb7, 0x91, 0xfe, 0xc8, 0x5f, 0xbd, 0xeb, 0xa1	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_DRIVER` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol1`. The protocol should still exist.

Number	GUID	Assertion	Test Description
5.1.3.2.90	0x762ef3c2, 0x6b3d, 0x43de, 0xa7, 0x1f, 0x59, 0x2c, 0xaa, 0x86, 0x83, 0xae	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_DRIVER` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol1`. The protocol should still exist.
5.1.3.2.91	0xfd5294e8, 0x55af, 0x4351, 0xa2, 0xab, 0x9f, 0x17, 0x6f, 0xa8, 0x61, 0x92	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_DRIVER` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol1`. The protocol should still exist.
5.1.3.2.92	0xe5c06a77, 0x3cec, 0x441f, 0xaf, 0xf2, 0x8a, 0x8c, 0x48, 0x86, 0x0a, 0x79	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_DRIVER` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.2.93	0x0f5dc8b8, 0x4a25, 0x4aaf, 0x9e, 0x60, 0xda, 0xd8, 0x77, 0x4d, 0x0b, 0x7f	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_DRIVER` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle again. The return code should be `EFI_SUCCESS`.
5.1.3.2.94	0xf33a826f, 0x02fd, 0x4a25, 0xbf, 0x1d, 0x4f, 0xa8, 0x8e, 0x66, 0x18, 0x31	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened `BY_DRIVER` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.2.95	0xbe28e107, 0xb5f6, 0x40d4, 0xb0, 0xcf, 0x58, 0xae, 0x87, 0x4d, 0x7f, 0x52	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_ACCESS_DENIED` to uninstall opened `BY_DRIVER|EXCLUSIVE` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER|EXCLUSIVE`. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.2.96	0x5abe9734, 0x3670, 0x4f0f, 0x8e, 0xaa, 0x52, 0x3f, 0x0c, 0xbd, 0xf3, 0xd3	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_ACCESS_DENIED` to uninstall opened `BY_DRIVER|EXCLUSIVE` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER|EXCLUSIVE. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.2.97	0xbac49627, 0xa912, 0x4d44, 0x84, 0xeb, 0x12, 0x0f, 0xe2, 0xcd, 0x91, 0x78	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` returns `EFI_ACCESS_DENIED` to uninstall opened BY_DRIVER|EXCLUSIVE at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER|EXCLUSIVE. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.2.98	0x8684158a, 0xf0b6, 0x4d70, 0x8f, 0xf8, 0xa1, 0x62, 0x2e, 0x8e, 0x6a, 0x66	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened BY_DRIVER|EXCLUSIVE at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER|EXCLUSIVE. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The handle should still exist.

Number	GUID	Assertion	Test Description
5.1.3.2.99	0x30eb72bb, 0x6451, 0x424c, 0xb7, 0x87, 0xad, 0x06, 0x49, 0x68, 0x97, 0x74	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened BY_DRIVER|EXCLUSIVE at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER|EXCLUSIVE. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The handle should still exist.
5.1.3.2.100	0x5167f4ff, 0x1647, 0x402c, 0xa8, 0x4f, 0x83, 0x02, 0x3e, 0x2e, 0x3e, 0x6a	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened BY_DRIVER|EXCLUSIVE at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER|EXCLUSIVE. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. The handle should still exist.
5.1.3.2.101	0x68190bde, 0x8248, 0x4c88, 0x89, 0x63, 0xaa, 0xb6, 0x32, 0xc3, 0x0f, 0xe6	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened BY_DRIVER|EXCLUSIVE at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER|EXCLUSIVE. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `LocateHandleBuffer()` to locate the handle via `TestProtocol1`. The protocol should still exist.

Number	GUID	Assertion	Test Description
5.1.3.2.102	0xc7a928d3, 0x6fba, 0x40bb, 0xa1, 0xc3, 0x18, 0x2e, 0x83, 0x48, 0x0a, 0x99	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened BY_DRIVER|EXCLUSIVE at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER|EXCLUSIVE. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The protocol should still exist.
5.1.3.2.103	0xbc91617f, 0xb732, 0x4464, 0xad, 0xf2, 0xf4, 0x8d, 0x2f, 0x78, 0x4d, 0x75	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened BY_DRIVER|EXCLUSIVE at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER|EXCLUSIVE. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call LocateHandleBuffer() to locate the handle via TestProtocol1. The protocol should still exist.
5.1.3.2.104	0xee7a01b0, 0x0dee, 0x49a7, 0xa8, 0xd3, 0x53, 0x9c, 0xfe, 0x27, 0xe4, 0x92	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls opened BY_DRIVER|EXCLUSIVE at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER|EXCLUSIVE. 3. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle. 4. Call CloseProtocol() to close TestProtocol1. 5. Call UninstallProtocolInterface() to remove TestProtocol1 from the handle again. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.2.105	0x26c0638e, 0x546c, 0x4729, 0xac, 0x25, 0x37, 0x56, 0xc1, 0x41, 0xb1, 0x79	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened BY_DRIVER|EXCLUSIVE at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER|EXCLUSIVE. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle again. The return code should be `EFI_SUCCESS`.
5.1.3.2.106	0x70fad80b, 0x9713, 0x46fd, 0xac, 0xdf, 0x25, 0x6c, 0x6f, 0xd9, 0xe4, 0x08	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls opened BY_DRIVER|EXCLUSIVE at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER|EXCLUSIVE. 3. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UninstallProtocolInterface()` to remove `TestProtocol1` from the handle again. The return code should be `EFI_SUCCESS`.
5.1.3.2.107	0x4621ba9e, 0xbc10, 0x4ff5, 0x99, 0xdc, 0x12, 0x90, 0x89, 0xa1, 0x63, 0x7d	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls `NULL` interface protocol at `EFI_TPL_APPLICATION`	1. Install `TestProtocolNoInterface1` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocolNoInterface1` from the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.2.108	0xb08ae228, 0x749e, 0x4d71, 0xb5, 0xc7, 0x7f, 0xfd, 0x8a, 0x97, 0x09, 0x6a	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls NULL interface protocol at EFI_TPL_CALLBACK	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call UninstallProtocolInterface() to remove TestProtocolNoInterface1 from the handle. The return code should be EFI_SUCCESS.
5.1.3.2.109	0x0b87b005, 0x552d, 0x4b7c, 0xb4, 0x9e, 0x05, 0x8d, 0x09, 0x26, 0xdc, 0xff	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls NULL interface protocol at EFI_TPL_NOTIFY	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call UninstallProtocolInterface() to remove TestProtocolNoInterface1 from the handle. The return code should be EFI_SUCCESS.
5.1.3.2.110	0x5ab7b1eb, 0xdb8c, 0x4b6b, 0x91, 0x78, 0x44, 0xef, 0x7b, 0x3c, 0xe0, 0x02	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls NULL interface protocol at EFI_TPL_APPLICATION	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call UninstallProtocolInterface() to remove TestProtocolNoInterface1 from the handle. The handle should no longer exist.
5.1.3.2.111	0x32ee9898, 0x6828, 0x4812, 0x9a, 0x41, 0x6e, 0x09, 0xb4, 0xd0, 0xe5, 0x54	BS.UninstallProtocolInterface – UninstallProtocolInterface() uninstalls NULL interface protocol at EFI_TPL_CALLBACK	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call UninstallProtocolInterface() to remove TestProtocolNoInterface1 from the handle. The handle should no longer exist.

Number	GUID	Assertion	Test Description
5.1.3.2.112	0x483766c8, 0xd28c, 0x4f5f, 0xb2, 0x6f, 0xa6, 0xb0, 0x36, 0xca, 0x0c, 0x36	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls `NULL` interface protocol at `EFI_TPL_NOTIFY`	1. Install `TestProtocolNoInterface1` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocolNoInterface1` from the handle. The handle should no longer exist.
5.1.3.2.113	0x07812110, 0xa22d, 0x4993, 0xa6, 0xd1, 0x25, 0x3e, 0x5f, 0x56, 0xa5, 0x56	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls `NULL` interface protocol at `EFI_TPL_APPLICATION`	1. Install `TestProtocolNoInterface1` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocolNoInterface1` from the handle. 3. Call `LocateHandleBuffer()` to locate the handle via `TestProtocolNoInterface1`. The return code should be `EFI_NOT_FOUND`.
5.1.3.2.114	0x97aaeeb5, 0x49e2, 0x4503, 0x9d, 0x2e, 0x37, 0x60, 0xce, 0x4f, 0x5d, 0x22	`BS.UninstallProtocolInterface – UninstallProtocolInterface()` uninstalls `NULL` interface protocol at `EFI_TPL_CALLBACK`	1. Install `TestProtocolNoInterface1` onto a new handle. 2. Call `UninstallProtocolInterface()` to remove `TestProtocolNoInterface1` from the handle. 3. Call `LocateHandleBuffer()` to locate the handle via `TestProtocolNoInterface1`. The return code should be `EFI_NOT_FOUND`.

Number	GUID	Assertion	Test Description
5.1.3.2.115	0xf08269a6, 0xe921, 0x408a, 0x97, 0xa7, 0xea, 0x6a, 0x60, 0x50, 0x97, 0x28	**BS.UninstallProtocolInterface – UninstallProtocolInterface()** uninstalls **NULL** interface protocol at **EFI_TPL_NOTIFY**	1. Install **TestProtocolNoInterface1** onto new handle. 2. Call **UninstallProtocolInterface()** to remove **TestProtocolNoInterface1** from the handle. 3. Call **LocateHandleBuffer()** to locate the handle via **TestProtocolNoInterface1**. The return code should be **EFI_NOT_FOUND**.

3.3.3 ReinstallProtocolInterface()

Number	GUID	Assertion	Test Description
5.1.3.3.1	0x2b830887, 0x5547, 0x4cfd, 0xb9, 0xf7, 0xb9, 0x1b, 0xf1, 0x48, 0xf5, 0x4c	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_INVALID_PARAMETER` with *`Protocol`* is `NULL`	1. Call `ReinstallProtocolInterface()` with the protocol GUID value of `NULL`. The return code must be `EFI_INVALID_PARAMETER`.
5.1.3.3.2	0xc7aedca3, 0xc600, 0x4fac, 0x84, 0xfa, 0x0c, 0x01, 0x0f, 0xf9, 0x9e, 0x67	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_NOT_FOUND` with invalid old protocol interface	1. Call `ReinstallProtocolInterface()` with the old protocol interface that does not point to the protocol interface installed upon current handle. The return code must be `EFI_NOT_FOUND`.
5.1.3.3.3	0xf7c8a812, 0x97c8, 0x4283, 0xa7, 0x79, 0x9c, 0x3a, 0x0d, 0xf9, 0x9b, 0x44	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_NOT_FOUND` with a non-existent protocol	1. Call `ReinstallProtocolInterface()` to attempt to install a new protocol that is not currently on the existing handle. The return code must be `EFI_NOT_FOUND`.
5.1.3.3.4	0x38e08d98, 0x7868, 0x4182, 0xb5, 0x61, 0xb5, 0x5d, 0x18, 0x70, 0xaa, 0x97	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_INVALID_PARAMETER` with invalid handle	1. Call `ReinstallProtocolInterface()` with an invalid handle (*`Handle`* is `NULL` or *`Handle`* is not valid). Each return code must be `EFI_INVALID_PARAMETER`.
5.1.3.3.5	0xe201db4d, 0x86bc, 0x470c, 0xa6, 0x6d, 0x78, 0xf7, 0x38, 0x72, 0xb0, 0x90	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_SUCCESS` with same interface at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface = old interface. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.3.6	0x40f531de, 0xe658, 0x4db5, 0xb4, 0xc6, 0x1a, 0xe6, 0x23, 0xbf, 0xb6, 0xc0	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_SUCCESS` with same interface at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface = old interface. The return code should be `EFI_SUCCESS`.
5.1.3.3.7	0x8e5fc1b6, 0xdad5, 0x45bd, 0x8d, 0x21, 0x0a, 0xd9, 0xef, 0x14, 0x17, 0x01	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_SUCCESS` with same interface at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface = old interface. The return code should be `EFI_SUCCESS`.
5.1.3.3.8	0x1f14d26c, 0x42a5, 0x49ff, 0x9e, 0xe2, 0x9f, 0x09, 0x58, 0xd2, 0x01, 0x10	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls same interface at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface = old interface. The new interface pointer should equal the address of the old interface.
5.1.3.3.9	0x113905d2, 0x997b, 0x487b, 0xb2, 0x61, 0x1f, 0xcc, 0x50, 0x82, 0xc0, 0x3b	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls same interface at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface = old interface. The new interface pointer should equal the address of the old interface.
5.1.3.3.10	0x7763db01, 0x78e5, 0x478a, 0xbf, 0xbb, 0xe7, 0xe2, 0xf1, 0xa4, 0xe3, 0xf6	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls same interface at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface = old interface. The new interface pointer should equal the address of the old interface.

Number	GUID	Assertion	Test Description
5.1.3.3.11	0x27cf47b1, 0xfff0, 0x41ce, 0xa0, 0x34, 0x9c, 0xde, 0x2c, 0xdf, 0x60, 0xa1	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls same interface at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface = old interface. The protocol interface should be really updated.
5.1.3.3.12	0x5d49efba, 0x9476, 0x4912, 0xa5, 0xf4, 0x36, 0xb6, 0x5d, 0x5f, 0xca, 0x2e	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls same interface at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface = old interface. The protocol interface should be really updated.
5.1.3.3.13	0xa18b9681, 0x284b, 0x416f, 0xaa, 0x60, 0x85, 0xb4, 0x45, 0x7b, 0x5e, 0x29	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls same interface at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface = old interface. The protocol interface should be really updated.
5.1.3.3.14	0x8e0e04cb, 0xe2c6, 0x40b4, 0x98, 0x11, 0x3e, 0x3f, 0x31, 0x18, 0x78, 0x0d	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_SUCCESS` with different interfaces at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface != old interface. The return code should be `EFI_SUCCESS`.
5.1.3.3.15	0x3c358ff2, 0x01fe, 0x45d2, 0x82, 0xf7, 0xe3, 0x01, 0x81, 0x9e, 0xa9, 0xa2	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_SUCCESS` with different interfaces at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface != old interface. The return code should be `EFI_SUCCESS`.
5.1.3.3.16	0x39f8a385, 0xfb98, 0x409b, 0xb9, 0x64, 0x27, 0xce, 0x2d, 0x8a, 0x97, 0x64	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_SUCCESS` with different interfaces at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface != old interface. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.3.17	0x283aa2e7, 0xc3e1, 0x4c51, 0x91, 0x30, 0x25, 0x8e, 0x3f, 0x23, 0xc2, 0x76	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls different interfaces at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface != old interface. The new interface pointer should equal the address of the new interface.
5.1.3.3.18	0xa7015b15, 0xcf81, 0x4e00, 0x8f, 0x37, 0xeb, 0xaa, 0xde, 0xac, 0xaa, 0x85	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls different interfaces at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface != old interface. The new interface pointer should equal the address of the new interface.
5.1.3.3.19	0xebdf5d21, 0x83f8, 0x4ba5, 0xa2, 0x9b, 0x6c, 0x6b, 0x0b, 0x46, 0xf6, 0xc3	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls different interfaces at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface != old interface. The new interface pointer should equal the address of the new interface.
5.1.3.3.20	0xdb9916f1, 0x58b4, 0x494f, 0x8e, 0x5a, 0x80, 0x8a, 0x6e, 0x8c, 0x7d, 0x01	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls different interfaces at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface != old interface. The protocol interface should be really updated.
5.1.3.3.21	0xdd723861, 0x1787, 0x48ab, 0xb5, 0xb5, 0xc7, 0xed, 0x9d, 0xa0, 0xb7, 0xa8	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls different interfaces at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `ReinstallProtocolInterface()` with the new interface != old interface. The protocol interface should be really updated.

Number	GUID	Assertion	Test Description
5.1.3.3.22	0xef59b8ea, 0x5b3f, 0x471b, 0xa2, 0x5a, 0x22, 0xb7, 0x27, 0x34, 0x22, 0xda	**BS.ReinstallProtocolInterface – ReinstallProtocolInterface()** reinstalls different interfaces at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **ReinstallProtocolInterface()** with the new interface != old interface. The protocol interface should be really updated.
5.1.3.3.23	0xb9309d48, 0xe467, 0x4836, 0x84, 0x97, 0x97, 0xdd, 0x58, 0x32, 0xc3, 0xff	**BS.ReinstallProtocolInterface – ReinstallProtocolInterface()** returns **EFI_SUCCESS** with opened **BY_HANDLE_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_HANDLE_PROTOCOL**. 3. Call **ReinstallProtocolInterface()** to reinstall the protocol. The return code should be **EFI_SUCCESS**.
5.1.3.3.24	0x1c319111, 0x6aaf, 0x4a88, 0xa5, 0x62, 0xe3, 0xc9, 0xa9, 0xc8, 0x35, 0xf0	**BS.ReinstallProtocolInterface – ReinstallProtocolInterface()** returns **EFI_SUCCESS** with opened **BY_HANDLE_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_HANDLE_PROTOCOL**. 3. Call **ReinstallProtocolInterface()** to reinstall the protocol. The return code should be **EFI_SUCCESS**.
5.1.3.3.25	0xed702361, 0x93d1, 0x4482, 0xb8, 0xf8, 0xb0, 0xcd, 0xc7, 0xc5, 0x5f, 0xe8	**BS.ReinstallProtocolInterface – ReinstallProtocolInterface()** returns **EFI_SUCCESS** with opened **BY_HANDLE_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_HANDLE_PROTOCOL**. 3. Call **ReinstallProtocolInterface()** to reinstall the protocol. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.3.26	0x0e8e9149, 0x41de, 0x4a21, 0xa5, 0x6d, 0xbb, 0xa1, 0x24, 0xfe, 0x26, 0xba	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened BY_HANDLE_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The new interface pointer should equal the address of the old interface.
5.1.3.3.27	0xae28eef8, 0xa415, 0x47bf, 0x87, 0x88, 0xe9, 0x3d, 0xad, 0xc4, 0x34, 0x20	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened BY_HANDLE_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The new interface pointer should equal the address of the old interface.
5.1.3.3.28	0x78893f3f, 0xb402, 0x45a5, 0x91, 0xd8, 0xc6, 0x5f, 0x67, 0xe7, 0xdc, 0xb4	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened BY_HANDLE_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The new interface pointer should equal the address of the old interface.
5.1.3.3.29	0x9ddcb93c, 0xec9a, 0x4185, 0x84, 0xbe, 0xe6, 0xa3, 0xa5, 0x17, 0x09, 0x97	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened BY_HANDLE_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The protocol interface was really updated.

Number	GUID	Assertion	Test Description
5.1.3.3.30	0x06638a28, 0x9534, 0x4e35, 0x9c, 0x20, 0x97, 0xd0, 0xd3, 0x8b, 0x5f, 0x09	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened BY_HANDLE_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The protocol interface was really updated.
5.1.3.3.31	0xeca41895, 0x43c3, 0x4f3b, 0xa7, 0x31, 0x85, 0x63, 0xdd, 0x3a, 0xeb, 0xcd	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened BY_HANDLE_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The protocol interface was really updated.
5.1.3.3.32	0x2c70bdd0, 0xb541, 0x4f03, 0xa5, 0x86, 0xb3, 0x1c, 0x7e, 0x47, 0xe2, 0xa0	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with opened GET_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_SUCCESS.
5.1.3.3.33	0xb02d6997, 0xba31, 0x4ea3, 0xaf, 0x25, 0x45, 0x1a, 0x4b, 0x05, 0x92, 0x4c	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with opened GET_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.3.34	0x7559ac82, 0xecc5, 0x460f, 0xa2, 0xf5, 0x75, 0x3a, 0x1f, 0xce, 0x0c, 0x97	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_SUCCESS` with opened `GET_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_SUCCESS`.
5.1.3.3.35	0xcf6c7824, 0x510d, 0x4547, 0xae, 0x31, 0x76, 0xe5, 0xdb, 0x18, 0x2f, 0x5a	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `GET_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The new interface pointer should equal the address of the old interface.
5.1.3.3.36	0x2812b788, 0xc622, 0x4aa2, 0x90, 0x5d, 0xa6, 0xb5, 0x29, 0xde, 0x31, 0x43	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `GET_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The new interface pointer should equal the address of the old interface.
5.1.3.3.37	0xeceb799c, 0xd852, 0x4f4f, 0xa3, 0x9f, 0x7e, 0x47, 0x30, 0x4b, 0xf6, 0x24	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `GET_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The new interface pointer should equal the address of the old interface.

Number	GUID	Assertion	Test Description
5.1.3.3.38	0x7f61a831, 0x357d, 0x4664, 0x8e, 0x26, 0xb3, 0xc5, 0x9d, 0xfb, 0x56, 0x3c	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `GET_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface was really updated.
5.1.3.3.39	0x87a27695, 0xd5c9, 0x4712, 0x9f, 0x7b, 0xd6, 0x00, 0x45, 0xb6, 0x77, 0xaa	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `GET_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface was really updated.
5.1.3.3.40	0x6056c396, 0x56a8, 0x4dbe, 0xbc, 0xd1, 0x00, 0x05, 0x3a, 0xa1, 0xd5, 0x04	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `GET_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface was really updated.
5.1.3.3.41	0x5e835916, 0x0850, 0x4380, 0xa9, 0x2c, 0x88, 0x24, 0x7c, 0x13, 0x67, 0x3a	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_SUCCESS` with opened `TEST_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.3.42	0xbc384cce, 0x25e7, 0x4ab4, 0x9b, 0x92, 0x8d, 0xd6, 0xca, 0xe2, 0x6a, 0x29	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with opened TEST_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_SUCCESS.
5.1.3.3.43	0xe8bfcebf, 0x4a8e, 0x4b76, 0xb6, 0xe9, 0xf4, 0xc2, 0x28, 0x72, 0x1a, 0x5b	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with opened TEST_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_SUCCESS.
5.1.3.3.44	0x0e0fc183, 0xaf09, 0x418d, 0x93, 0xf6, 0x17, 0x72, 0x80, 0xf9, 0x0d, 0x67	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened TEST_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The new interface pointer should equal the address of the old interface.
5.1.3.3.45	0x477f42d0, 0x5755, 0x4907, 0xa4, 0xe9, 0x49, 0x2e, 0x12, 0x47, 0x11, 0xeb	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened TEST_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The new interface pointer should equal the address of the old interface.

Number	GUID	Assertion	Test Description
5.1.3.3.46	0xa05dfd9c, 0x4c54, 0x43b1, 0xbf, 0x78, 0x32, 0x27, 0x4a, 0x67, 0x28, 0x5a	**BS.ReinstallProtocolInterface – ReinstallProtocolInterface()** reinstalls opened **TEST_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **TEST_PROTOCOL**. 3. Call **ReinstallProtocolInterface()** to reinstall the protocol. The new interface pointer should equal the address of the old interface.
5.1.3.3.47	0x9537f350, 0xa519, 0x4272, 0xbf, 0xe6, 0x97, 0x0e, 0xe1, 0xf2, 0x95, 0x87	**BS.ReinstallProtocolInterface – ReinstallProtocolInterface()** reinstalls opened **TEST_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **TEST_PROTOCOL**. 3. Call **ReinstallProtocolInterface()** to reinstall the protocol. The protocol interface was really updated.
5.1.3.3.48	0x1d00d8e3, 0xe6a3, 0x46ee, 0xa3, 0x4e, 0x5f, 0xe2, 0xf7, 0x23, 0xf3, 0xf8	**BS.ReinstallProtocolInterface – ReinstallProtocolInterface()** reinstalls opened **TEST_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **TEST_PROTOCOL**. 3. Call **ReinstallProtocolInterface()** to reinstall the protocol. The protocol interface was really updated.
5.1.3.3.49	0x9ab51ea3, 0xbe65, 0x44c7, 0xbe, 0x31, 0x2b, 0xc8, 0xea, 0x6d, 0x23, 0xa9	**BS.ReinstallProtocolInterface – ReinstallProtocolInterface()** reinstalls opened **TEST_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **TEST_PROTOCOL**. 3. Call **ReinstallProtocolInterface()** to reinstall the protocol. The protocol interface was really updated.

Number	GUID	Assertion	Test Description
5.1.3.3.50	0xffaacc85, 0x9e40, 0x433b, 0xbc, 0x21, 0xe2, 0xae, 0xad, 0x5f, 0xa9, 0x15	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_ACCESS_DENIED` with opened `BY_CHILD_CONTROLLER` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.3.51	0xa8354a22, 0x115e, 0x4a3d, 0xb7, 0x39, 0xa3, 0x78, 0x64, 0xf8, 0x0b, 0xa2	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_ACCESS_DENIED` with opened `BY_CHILD_CONTROLLER` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.3.52	0x0af4e34f, 0x8af0, 0x485f, 0x91, 0x9d, 0x2d, 0xe9, 0x2e, 0x30, 0xee, 0x3d	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_ACCESS_DENIED` with opened `BY_CHILD_CONTROLLER` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.3.53	0xf757a668, 0x07e6, 0x4744, 0xa3, 0x2a, 0x79, 0x0b, 0xe9, 0x16, 0xa2, 0xad	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface should not be updated.

Number	GUID	Assertion	Test Description
5.1.3.3.54	0x5c504893, 0x0ab2, 0x4282, 0xba, 0x26, 0x12, 0xe6, 0xbd, 0x26, 0xa1, 0xb3	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface should not be updated.
5.1.3.3.55	0xc06e1bcd, 0x10a7, 0x4d16, 0xaa, 0x74, 0x2a, 0xaf, 0x34, 0xef, 0x9d, 0xca	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface should not be updated.
5.1.3.3.56	0x83410d83, 0x5a33, 0x4f8b, 0x89, 0xee, 0x93, 0x84, 0x3a, 0xf0, 0xfc, 0xd2	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.3.57	0x5c89d64f, 0x479e, 0x403a, 0xb8, 0xcd, 0xc2, 0x3a, 0x38, 0xad, 0x39, 0xe1	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened BY_CHILD_CONTROLLER at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call ReinstallProtocolInterface() to reinstall the protocol. 4. Call CloseProtocol() to close TestProtocol1. 5. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_SUCCESS.
5.1.3.3.58	0x02216a3f, 0xa63f, 0x4844, 0x9d, 0x57, 0x87, 0x59, 0xcc, 0x0e, 0xbc, 0x9e	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened BY_CHILD_CONTROLLER at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call ReinstallProtocolInterface() to reinstall the protocol. 4. Call CloseProtocol() to close TestProtocol1. 5. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_SUCCESS.
5.1.3.3.59	0x4e466e37, 0xd264, 0x455c, 0xb2, 0x37, 0x4b, 0x8a, 0x52, 0x98, 0x6e, 0xe6	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_ACCESS_DENIED with opened BY_DRIVER at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.3.60	0xd8ae4f16, 0x1a15, 0x4e23, 0xa1, 0xb3, 0xb2, 0xbc, 0x14, 0x00, 0x17, 0x11	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_ACCESS_DENIED` with opened BY_DRIVER at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.3.61	0xf1743d0d, 0x7d64, 0x433a, 0x90, 0xd9, 0x75, 0x06, 0xbc, 0x2d, 0xf9, 0xe6	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_ACCESS_DENIED` with opened BY_DRIVER at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.3.62	0x9152e17f, 0x7d25, 0x4b84, 0xaa, 0x1c, 0xd0, 0x9e, 0x4d, 0x99, 0x7d, 0x7c	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened BY_DRIVER at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface should not be updated.
5.1.3.3.63	0x557ed71a, 0x83db, 0x476f, 0xb4, 0x02, 0x5e, 0xec, 0x8d, 0x89, 0xf0, 0xd8	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened BY_DRIVER at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface should not be updated.

Number	GUID	Assertion	Test Description
5.1.3.3.64	0x6b425b04, 0xf68c, 0x44e7, 0xbe, 0x5d, 0x8b, 0xea, 0x39, 0x78, 0xc7, 0x45	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened BY_DRIVER at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface should not be updated.
5.1.3.3.65	0x0b55c435, 0xed26, 0x459c, 0xa5, 0x36, 0x70, 0xf4, 0x51, 0x18, 0xe8, 0x93	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened BY_DRIVER at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_SUCCESS`.
5.1.3.3.66	0x1fd7feef, 0xd9a4, 0x46dc, 0x94, 0x97, 0x4a, 0xff, 0x06, 0x0b, 0xca, 0x84	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened BY_DRIVER at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.3.67	0x60c75742, 0x8c58, 0x40e2, 0x88, 0xb4, 0x0d, 0x7d, 0x4c, 0x81, 0x25, 0xe6	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened BY_DRIVER at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` BY_DRIVER. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_SUCCESS`.
5.1.3.3.68	0x335d503c, 0x1624, 0x4d44, 0x84, 0x22, 0x94, 0x74, 0xb3, 0xcd, 0xb7, 0xb2	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_ACCESS_DENIED` with opened `EXCLUSIVE` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.3.69	0xb5c308fb, 0x8ea7, 0x428e, 0xa7, 0x62, 0x1e, 0x70, 0x9d, 0x90, 0x10, 0x74	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_ACCESS_DENIED` with opened `EXCLUSIVE` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_ACCESS_DENIED`.

Number	GUID	Assertion	Test Description
5.1.3.3.70	0xd05e98dd, 0x157e, 0x49db, 0xbf, 0xd9, 0x43, 0x25, 0x5b, 0x91, 0x5c, 0x53	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_ACCESS_DENIED` with opened `EXCLUSIVE` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.3.71	0x18e2625f, 0x1066, 0x4467, 0x9f, 0x8c, 0xa1, 0x84, 0xa7, 0x46, 0xaa, 0x43	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `EXCLUSIVE` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface should not be updated.
5.1.3.3.72	0x6797c7e3, 0xbddd, 0x4519, 0x85, 0x1e, 0x6c, 0x81, 0x71, 0xba, 0xbe, 0x52	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `EXCLUSIVE` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface should not be updated.
5.1.3.3.73	0x37bfec5b, 0x8899, 0x48b2, 0x9e, 0x3d, 0x6c, 0x48, 0x74, 0x80, 0xfd, 0x00	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `EXCLUSIVE` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface should not be updated.

Number	GUID	Assertion	Test Description
5.1.3.3.74	0x4f15dee5, 0x6319, 0x431b, 0xb4, 0x2c, 0x7c, 0x88, 0x36, 0x35, 0x4b, 0x1c	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call ReinstallProtocolInterface() to reinstall the protocol. 4. Call CloseProtocol() to close TestProtocol1. 5. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_SUCCESS.
5.1.3.3.75	0x9478a613, 0x8521, 0x4832, 0xa3, 0x74, 0xfc, 0x5d, 0xe9, 0xaa, 0x0b, 0xa1	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call ReinstallProtocolInterface() to reinstall the protocol. 4. Call CloseProtocol() to close TestProtocol1. 5. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_SUCCESS.
5.1.3.3.76	0x109a1695, 0xaf0a, 0x43a7, 0xad, 0xb5, 0x7d, 0x50, 0x9b, 0x85, 0xff, 0xd3	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call ReinstallProtocolInterface() to reinstall the protocol. 4. Call CloseProtocol() to close TestProtocol1. 5. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.3.77	0xcf4bb456, 0x29fe, 0x4e46, 0x9b, 0x38, 0x09, 0x73, 0x93, 0x9a, 0xa9, 0x2a	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_ACCESS_DENIED` with opened `BY_DRIVER | EXCLUSIVE` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER | EXCLUSIVE`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.3.78	0x71890aa7, 0xa7e5, 0x454c, 0xb6, 0xc3, 0x69, 0xb1, 0x1d, 0x7d, 0xac, 0x55	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_ACCESS_DENIED` with opened `BY_DRIVER | EXCLUSIVE` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER | EXCLUSIVE`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.3.79	0x5ae4c26a, 0xcbed, 0x4aa2, 0x9f, 0x52, 0x47, 0x78, 0x60, 0xd3, 0x13, 0xcc	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` returns `EFI_ACCESS_DENIED` with opened `BY_DRIVER | EXCLUSIVE` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER | EXCLUSIVE`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.3.80	0xcfc17ae1, 0x8cc8, 0x4e46, 0xaa, 0x91, 0xf6, 0xaa, 0x6a, 0xe0, 0x10, 0x76	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls opened `BY_DRIVER | EXCLUSIVE` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER | EXCLUSIVE`. 3. Call `ReinstallProtocolInterface()` to reinstall the protocol. The protocol interface should not be updated.

Number	GUID	Assertion	Test Description
5.1.3.3.81	0x7cd52d24, 0xd8b9, 0x458a, 0xa7, 0x0b, 0x35, 0x3c, 0x34, 0xbe, 0xa0, 0x3f	**BS.ReinstallProtocolInterface – ReinstallProtocolInterface()** reinstalls opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER | EXCLUSIVE**. 3. Call **ReinstallProtocolInterface()** to reinstall the protocol. The protocol interface should not be updated.
5.1.3.3.82	0x1e43e41e, 0x0119, 0x4ab5, 0x81, 0x3f, 0x99, 0xe3, 0xcc, 0x20, 0x79, 0xd7	**BS.ReinstallProtocolInterface – ReinstallProtocolInterface()** reinstalls opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER | EXCLUSIVE**. 3. Call **ReinstallProtocolInterface()** to reinstall the protocol. The protocol interface should not be updated.
5.1.3.3.83	0xee9a742a, 0xc536, 0x47c1, 0x8c, 0x36, 0x79, 0x2a, 0x97, 0x36, 0x77, 0x61	**BS.ReinstallProtocolInterface – ReinstallProtocolInterface()** reinstalls opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER | EXCLUSIVE**. 3. Call **ReinstallProtocolInterface()** to reinstall the protocol. 4. Call **CloseProtocol()** to close **TestProtocol1**. 5. Call **ReinstallProtocolInterface()** to reinstall the protocol. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.3.84	0x29b926e6, 0x8279, 0x44ca, 0x97, 0x26, 0xf1, 0xd6, 0x54, 0xbf, 0xe1, 0x83	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened BY_DRIVER | EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE. 3. Call ReinstallProtocolInterface() to reinstall the protocol. 4. Call CloseProtocol() to close TestProtocol1. 5. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_SUCCESS.
5.1.3.3.85	0x5c1a7657, 0x40ad, 0x473c, 0xaf, 0xf5, 0xd1, 0x4a, 0xcd, 0xdf, 0xf3, 0xad	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls opened BY_DRIVER | EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE. 3. Call ReinstallProtocolInterface() to reinstall the protocol. 4. Call CloseProtocol() to close TestProtocol1. 5. Call ReinstallProtocolInterface() to reinstall the protocol. The return code should be EFI_SUCCESS.
5.1.3.3.86	0xb83b3c39, 0x6e9d, 0x4289, 0xa2, 0x42, 0x14, 0x2d, 0xda, 0x62, 0x0b, 0xe1	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with NULL to NULL at EFI_TPL_APPLICATION	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with NULL interface. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.3.87	0x77dc0aed, 0x6f4a, 0x45a4, 0xaa, 0x99, 0x29, 0xaf, 0x10, 0xc8, 0x4d, 0xf5	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with NULL to NULL at EFI_TPL_CALLBACK	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with NULL interface. The return code should be EFI_SUCCESS.
5.1.3.3.88	0xf97d5424, 0xa904, 0x40f2, 0x8a, 0xc8, 0x23, 0xa8, 0xac, 0xca, 0xc2, 0xad	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with NULL to NULL at EFI_TPL_NOTIFY	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with NULL interface. The return code should be EFI_SUCCESS.
5.1.3.3.89	0xf3cb0a58, 0x4682, 0x425d, 0x91, 0xfd, 0x7a, 0x10, 0xe4, 0xa0, 0xf3, 0x50	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with NULL to non-NULL at EFI_TPL_APPLICATION	1. Install TestProtocol1 with a non-NULL interface onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with NULL interface. The return code should be EFI_SUCCESS.
5.1.3.3.90	0x7ed1d007, 0x7f32, 0x493a, 0xb0, 0xc9, 0xba, 0xce, 0xdc, 0x2d, 0xdd, 0xed	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with NULL to non-NULL at EFI_TPL_CALLBACK	1. Install TestProtocol1 with a non-NULL interface onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with NULL interface. The return code should be EFI_SUCCESS.
5.1.3.3.91	0x48c64365, 0x01dd, 0x41c6, 0x93, 0x6e, 0x28, 0xea, 0x1d, 0xde, 0x0c, 0x1f	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with NULL to non-NULL at EFI_TPL_NOTIFY	1. Install TestProtocol1 with a non-NULL interface onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with NULL interface. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.3.92	0xa22e15c8, 0xe151, 0x4b84, 0xa0, 0x6b, 0x7f, 0x99, 0x28, 0x7f, 0xff, 0x64	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls with NULL interface to non-NULL at EFI_TPL_APPLICATION	1. Install TestProtocol1 with a non-NULL interface onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with NULL interface. The TestProtocol1's interface should be NULL.
5.1.3.3.93	0xc9da7aef, 0x77e0, 0x44d4, 0xbd, 0xa8, 0x6e, 0xd6, 0xad, 0x3a, 0xf3, 0xfd	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls with NULL interface to non-NULL at EFI_TPL_CALLBACK	1. Install TestProtocol1 with a non-NULL interface onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with NULL interface. The TestProtocol1's interface should be NULL.
5.1.3.3.94	0xa6f419a6, 0xcf35, 0x40ea, 0x80, 0x9c, 0x19, 0xe7, 0xcf, 0x8e, 0xcb, 0x95	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls with NULL interface to non-NULL at EFI_TPL_NOTIFY	1. Install TestProtocol1 with a non-NULL interface onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with NULL interface. The TestProtocol1's interface should be NULL.
5.1.3.3.95	0x6926fa2f, 0xf78c, 0x454a, 0x91, 0x85, 0x56, 0x7b, 0x93, 0x8d, 0x17, 0x29	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with non-NULL to NULL at EFI_TPL_APPLICATION	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with non-NULL interface. The return code should be EFI_SUCCESS.
5.1.3.3.96	0x0d00253b, 0x00d7, 0x429a, 0xba, 0x56, 0x7f, 0x91, 0x84, 0x77, 0xd8, 0xba	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with non-NULL to NULL at EFI_TPL_CALLBACK	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with non-NULL interface. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.3.97	0x799c3528, 0x4d2e, 0x4329, 0xa6, 0x9b, 0xce, 0x5c, 0x42, 0xf8, 0x3e, 0x00	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() returns EFI_SUCCESS with non-NULL to NULL at EFI_TPL_NOTIFY	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with non-NULL interface. The return code should be EFI_SUCCESS.
5.1.3.3.98	0x339ae67e, 0xdc65, 0x4411, 0xb6, 0x11, 0x5d, 0xfc, 0xd5, 0xcb, 0x70, 0x06	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls with non-NULL interface to NULL at EFI_TPL_APPLICATION	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with non-NULL interface. The new interface pointer should equal the address of the new interface.
5.1.3.3.99	0x75c6076f, 0xf57b, 0x4892, 0xaf, 0xa7, 0x1c, 0xa5, 0x51, 0x04, 0x36, 0x2a	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls with non-NULL interface to NULL at EFI_TPL_CALLBACK	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with non-NULL interface. The new interface pointer should equal the address of the new interface.
5.1.3.3.100	0x03ad7b51, 0x36c3, 0x4bf9, 0x91, 0x18, 0x2c, 0x50, 0xe7, 0x1d, 0x36, 0x1d	BS.ReinstallProtocolInterface – ReinstallProtocolInterface() reinstalls with non-NULL interface to NULL at EFI_TPL_NOTIFY	1. Install TestProtocolNoInterface1 onto a new handle. 2. Call ReinstallProtocolInterface() to reinstall the protocol with non-NULL interface. The new interface pointer should equal the address of the new interface.

Number	GUID	Assertion	Test Description
5.1.3.3.101	0x0f91c7bb, 0x0e0b, 0x426a, 0x8b, 0x6b, 0xe5, 0x7f, 0x12, 0xb9, 0xa8, 0x5c	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls with non-`NULL` interface to `NULL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocolNoInterface1` onto a new handle. 2. Call `ReinstallProtocolInterface()` to reinstall the protocol with non-`NULL` interface. The protocol interface was really updated.
5.1.3.3.102	0x254d9491, 0x1249, 0x4abd, 0xa6, 0x72, 0x5d, 0xfa, 0x68, 0xd9, 0x58, 0x6f	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls with non-`NULL` interface to `NULL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocolNoInterface1` onto a new handle. 2. Call `ReinstallProtocolInterface()` to reinstall the protocol with non-`NULL` interface. The protocol interface was really updated.
5.1.3.3.103	0x662e7cb3, 0x297b, 0x4d97, 0x81, 0x6d, 0xc7, 0x61, 0x74, 0xad, 0x72, 0xee	`BS.ReinstallProtocolInterface – ReinstallProtocolInterface()` reinstalls with non-`NULL` interface to `NULL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocolNoInterface1` onto a new handle. 2. Call `ReinstallProtocolInterface()` to reinstall the protocol with non-`NULL` interface. The protocol interface was really updated.

3.3.4 RegisterProtocolNotify()

Number	GUID	Assertion	Test Description
5.1.3.4.1	0x4bce9d1a, 0xffae, 0x4809, 0x82, 0xae, 0xf6, 0x6e, 0x10, 0xeb, 0x59, 0x74	BS.RegisterProtocolNotify – RegisterProtocolNotify() returns EFI_SUCCESS with valid event at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() with each event listed. Each return code should be EFI_SUCCESS.
5.1.3.4.2	0x11b76c1d, 0xdba6, 0x4535, 0x94, 0xe0, 0xf3, 0x9d, 0xcf, 0x86, 0x24, 0xd7	BS.RegisterProtocolNotify – RegisterProtocolNotify() returns EFI_SUCCESS with valid event at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() with each event listed. Each return code should be EFI_SUCCESS.
5.1.3.4.3	0x1390658d, 0x9c5e, 0x4af6, 0x9d, 0x9e, 0xe9, 0x19, 0xf3, 0x80, 0xa9, 0x71	BS.RegisterProtocolNotify – RegisterProtocolNotify() returns EFI_SUCCESS with valid event at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() with each event listed. Each return code should be EFI_SUCCESS.
5.1.3.4.4	0x47249e03, 0x836b, 0x4c44, 0xad, 0xe5, 0x4a, 0x0f, 0x79, 0xdd, 0x60, 0x99	BS.RegisterProtocolNotify – RegisterProtocolNotify() gets the registration key with valid event at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() with each event listed. After each calling, a registration key should be returned.
5.1.3.4.5	0xbd50e782, 0xaa2b, 0x4f5f, 0x85, 0x69, 0x12, 0x3d, 0x4f, 0x81, 0x7b, 0x78	BS.RegisterProtocolNotify – RegisterProtocolNotify() gets the registration key with valid event at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() with each event listed. After each calling, a registration key should be returned.
5.1.3.4.6	0x434968fe, 0x0a2f, 0x4806, 0x94, 0x7a, 0xc6, 0x69, 0x4f, 0x8f, 0x5a, 0x57	BS.RegisterProtocolNotify – RegisterProtocolNotify() gets the registration key with valid event at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() with each event listed. After each calling, a registration key should be returned.
5.1.3.4.7	0x18a14727, 0x39f9, 0x4dce, 0xa2, 0xf2, 0xaf, 0x82, 0x56, 0x29, 0x67, 0x6d	BS.RegisterProtocolNotify – RegisterProtocolNotify() returns EFI_SUCCESS with protocol at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. Each return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.4.8	0x94bc9e2d, 0x048b, 0x4c76, 0xaf, 0xe3, 0xfe, 0x93, 0x96, 0xe1, 0xef, 0x3d	BS.RegisterProtocolNotify – RegisterProtocolNotify() returns EFI_SUCCESS with protocol at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. Each return code should be EFI_SUCCESS.
5.1.3.4.9	0xdd09bb3a, 0x7e6b, 0x441d, 0xb3, 0xce, 0xa6, 0x98, 0x78, 0x16, 0xce, 0x9b	BS.RegisterProtocolNotify – RegisterProtocolNotify() returns EFI_SUCCESS with protocol at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. Each return code should be EFI_SUCCESS.
5.1.3.4.10	0x11cca836, 0x9ff0, 0x481b, 0x84, 0x03, 0x8e, 0xe2, 0x72, 0x52, 0x57, 0xb2	BS.RegisterProtocolNotify – RegisterProtocolNotify() registers the notify function with protocol at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. The return colde should be EFI_SUCCESS.
5.1.3.4.11	0xdcb04d09, 0xfd98, 0x495e, 0xaa, 0x14, 0x4c, 0x16, 0xae, 0xe5, 0x81, 0xcc	BS.RegisterProtocolNotify – RegisterProtocolNotify() registers the notify function with protocol at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. The return colde should be EFI_SUCCESS.
5.1.3.4.12	0xe8708024, 0x8a28, 0x4fac, 0xa5, 0x86, 0x80, 0xaf, 0xa1, 0x26, 0x55, 0x33	BS.RegisterProtocolNotify – RegisterProtocolNotify() registers the notify function with protocol at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. The return colde should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.4.13	0xd0587022, 0x05e4, 0x4127, 0x98, 0x2f, 0x83, 0xe6, 0x84, 0x9e, 0xb1, 0x50	**BS.RegisterProtocolNotify – RegisterProtocolNotify()** registers the notify function with protocol at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** with each event registered for **TestProtocol1**. 2. Call **InstallProtocolInterface()** to install **TestProtocol1**. All events notify functions should be invoked, and each was invoked once.
5.1.3.4.14	0x43a33e3d, 0x48d1, 0x4ea2, 0x82, 0x3c, 0xf9, 0xb5, 0x5a, 0xbe, 0x3f, 0xdc	**BS.RegisterProtocolNotify – RegisterProtocolNotify()** registers the notify function with protocol at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** with each event registered for **TestProtocol1**. 2. Call **InstallProtocolInterface()** to install **TestProtocol1**. All events notify functions should be invoked, and each was invoked once.
5.1.3.4.15	0xb55fd245, 0xfd96, 0x4dc7, 0x9f, 0xa6, 0x97, 0xf1, 0x84, 0x7e, 0x8c, 0x4e	**BS.RegisterProtocolNotify – RegisterProtocolNotify()** registers the notify function with protocol at **EFI_TPL_NOTIFY**	1. Call **RegisterProtocolNotify()** with each event registered for **TestProtocol1**. 2. Call **InstallProtocolInterface()** to install **TestProtocol1**. All events notify functions should be invoked, and each was invoked once.

Number	GUID	Assertion	Test Description
5.1.3.4.16	0x4864b70d, 0x5573, 0x4ac7, 0x86, 0xd7, 0xb2, 0x0d, 0xcb, 0x9e, 0x06, 0x4c	`BS.RegisterProtocolNotify – RegisterProtocolNotify()` registers the notify function with protocol at `EFI_TPL_APPLICATION`	1. Call `RegisterProtocolNotify()` with each event registered for `TestProtocol1`. 2. Call `InstallProtocolInterface()` to install `TestProtocol1`. 3. Call `ReinstallProtocolInterface()` to reinstall `TestProtocol1`. The return code should be `EFI_SUCCESS`.
5.1.3.4.17	0x52c7b2b1, 0x828c, 0x4e1c, 0x95, 0xa7, 0xb9, 0x96, 0xc8, 0xcf, 0x08, 0x02	`BS.RegisterProtocolNotify – RegisterProtocolNotify()` registers the notify function with protocol at `EFI_TPL_CALLBACK`	1. Call `RegisterProtocolNotify()` with each event registered for `TestProtocol1`. 2. Call `InstallProtocolInterface()` to install `TestProtocol1`. 3. Call `ReinstallProtocolInterface()` to reinstall `TestProtocol1`. The return code should be `EFI_SUCCESS`.
5.1.3.4.18	0xe9c27a4d, 0x17ec, 0x4edd, 0x9c, 0xe0, 0x75, 0x0b, 0x7d, 0x41, 0xf6, 0x70	`BS.RegisterProtocolNotify – RegisterProtocolNotify()` registers the notify function with protocol at `EFI_TPL_NOTIFY`	1. Call `RegisterProtocolNotify()` with each event registered for `TestProtocol1`. 2. Call `InstallProtocolInterface()` to install `TestProtocol1`. 3. Call `ReinstallProtocolInterface()` to reinstall `TestProtocol1`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.4.19	0x86f38f07, 0x185a, 0x498a, 0x9b, 0x66, 0xf9, 0xe0, 0x5c, 0xc4, 0x18, 0xd7	`BS.RegisterProtocolNotify – RegisterProtocolNotify()` registers the notify function with protocol at `EFI_TPL_APPLICATION`	1. Call `RegisterProtocolNotify()` with each event registered for `TestProtocol1`. 2. Call `InstallProtocolInterface()` to install `TestProtocol1`. 3. Call `ReinstallProtocolInterface()` to reinstall `TestProtocol1`. All events notify functions should be invoked again, and the total invocation time for each function is twice.
5.1.3.4.20	0x9b7d258e, 0xd87f, 0x4a91, 0xb5, 0x73, 0xeb, 0x06, 0x92, 0x7f, 0xbd, 0x3b	`BS.RegisterProtocolNotify – RegisterProtocolNotify()` registers the notify function with protocol at `EFI_TPL_CALLBACK`	1. Call `RegisterProtocolNotify()` with each event registered for `TestProtocol1`. 2. Call `InstallProtocolInterface()` to install `TestProtocol1`. 3. Call `ReinstallProtocolInterface()` to reinstall `TestProtocol1`. All events notify functions should be invoked again, and the total invocation time for each function is twice.

Number	GUID	Assertion	Test Description
5.1.3.4.21	0x1906999e, 0x7c7e, 0x4a3e, 0x96, 0x44, 0x0a, 0x25, 0xd5, 0xd9, 0x50, 0x53	BS.RegisterProtocolNotify – RegisterProtocolNotify() registers the notify function with protocol at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. 3. Call ReinstallProtocolInterface() to reinstall TestProtocol1. All events notify functions should be invoked again, and the total invocation time for each function is twice.
5.1.3.4.22	0x90068144, 0xc425, 0x47d3, 0x89, 0x72, 0xb5, 0xab, 0xf1, 0x2c, 0x82, 0x7a	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. The return code should be EFI_SUCCESS
5.1.3.4.23	0x9ef7d002, 0x2ea2, 0x486d, 0xbf, 0xad, 0x25, 0x43, 0x5c, 0x43, 0xf7, 0x2a	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. The return code should be EFI_SUCCESS
5.1.3.4.24	0xa81be45d, 0x7534, 0x43a3, 0xb9, 0xf1, 0x60, 0x4f, 0x01, 0x87, 0xfb, 0x62	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. The return code should be EFI_SUCCESS

Number	GUID	Assertion	Test Description
5.1.3.4.25	0xb2d4b97e, 0xee48, 0x40f7, 0xb3, 0x49, 0xac, 0x1b, 0x0f, 0x8c, 0xc3, 0x92	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. The return code should be EFI_SUCCESS.
5.1.3.4.26	0x5263bb06, 0x8ae4, 0x46c4, 0xb0, 0xee, 0x4b, 0xd8, 0x88, 0x41, 0xe7, 0x85	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. The return code should be EFI_SUCCESS.
5.1.3.4.27	0xa39497a5, 0x7a70, 0x43e1, 0x80, 0x86, 0x8b, 0x8d, 0x89, 0xe7, 0xf3, 0xed	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. The return code should be EFI_SUCCESS.
5.1.3.4.28	0xfc11a5e8, 0x3b22, 0x4e75, 0xbb, 0xb0, 0xc3, 0x3b, 0x1c, 0x57, 0xfd, 0xa5	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. All events notify functions should be invoked, and the return code of LocateHandleBuffer() should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.4.29	0x5b9b80ae, 0x9d2f, 0x4506, 0x86, 0xc7, 0x0b, 0xa9, 0x30, 0x85, 0x27, 0xcf	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. All events notify functions should be invoked, and the return code of LocateHandleBuffer() should be EFI_SUCCESS.
5.1.3.4.30	0x5ec22e94, 0xcce7, 0x4448, 0x86, 0xad, 0xe3, 0xe0, 0x11, 0xf9, 0x2d, 0xdc	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. All events notify functions should be invoked, and the return code of LocateHandleBuffer() should be EFI_SUCCESS.
5.1.3.4.31	0xdca77cf4, 0x72d4, 0x4762, 0x8f, 0x7d, 0x27, 0xe5, 0xdd, 0x2a, 0x73, 0x31	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. 3. Call ReinstallProtocolInterface() to reinstall TestProtocol1. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.4.32	0xc0ca4f13, 0xf662, 0x4f2b, 0xb6, 0x68, 0xbe, 0x7c, 0x5a, 0xfc, 0x51, 0x1a	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. 3. Call ReinstallProtocolInterface() to reinstall TestProtocol1. The return code should be EFI_SUCCESS.
5.1.3.4.33	0x30abe85d, 0x2093, 0x4405, 0xb3, 0x48, 0x9f, 0x7f, 0xa1, 0xda, 0x71, 0xe2	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. 3. Call ReinstallProtocolInterface() to reinstall TestProtocol1. The return code should be EFI_SUCCESS.
5.1.3.4.34	0xfb8dcf11, 0xf107, 0x4bee, 0xa3, 0x2e, 0xb4, 0xb5, 0xe9, 0x86, 0x22, 0x2b	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. 3. Call ReinstallProtocolInterface() to reinstall TestProtocol1. All events notify functions should be invoked, and the return code of LocateHandleBuffer() is EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.4.35	0x6a48a665, 0xf22a, 0x4014, 0xaf, 0x11, 0x78, 0x72, 0x97, 0x5a, 0x13, 0x20	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. 3. Call ReinstallProtocolInterface() to reinstall TestProtocol1. All events notify functions should be invoked, and the return code of LocateHandleBuffer() is EFI_SUCCESS.
5.1.3.4.36	0x292a3e09, 0x6e51, 0x4025, 0xb5, 0xb4, 0xf9, 0x46, 0x9a, 0x4b, 0x39, 0x4e	BS.RegisterProtocolNotify – LocateHandleBuffer() with registration key at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() with each event registered for TestProtocol1. 2. Call InstallProtocolInterface() to install TestProtocol1. 3. Call ReinstallProtocolInterface() to reinstall TestProtocol1. All events notify functions should be invoked, and the return code of LocateHandleBuffer() is EFI_SUCCESS.
5.1.3.4.37	0x8922622c, 0x2b5a, 0x4438, 0x92, 0x31, 0xda, 0x35, 0x85, 0xac, 0x83, 0x0c	BS.RegisterProtocolNotify - ConsistencyTestCheckpoint3	Call RegisterProtocolNotify() with a Protocol Guid being NULL. The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.1.3.4.38	0x51761a02, 0xdd1f, 0x4d8a, 0x95, 0xa6, 0x38, 0xb6, 0x0e, 0x1d, 0xdb, 0xf5	`BS.RegisterProtocolNotify` - ConsistencyTestCheckpoint3	Call `RegisterProtocolNotify()` with a `Event` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.
5.1.3.4.39	0xdf8f26aa, 0xdf96, 0x4700, 0xbc, 0xbb, 0x6a, 0x3c, 0x98, 0x8c, 0xfd, 0x97	`BS.RegisterProtocolNotify` - ConsistencyTestCheckpoint3	Call `RegisterProtocolNotify()` with the `Registration` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.
5.1.3.4.40	0xc74cea76, 0xac9a, 0x4a43, 0x80, 0xa6, 0xb5, 0xe3, 0xe3, 0x85, 0x45, 0xe7	**BS.RegisterProtocolNotify -** Events that have been registered for protocol interface notification can be unregistered by calling **CloseEvent()**.	.1. Call **CreateEvent()** to create Event1 with **EVT_NOTIFY_SIGNAL** and **CALLBACK TPL**, create Event2 with **EVT_NOTIFY_SIGNAL** and **NOTIFY TPL**. They are registered with **RegisterProtocolNotify**() with the specified protocol. 2. Call **CloseEvent()** to close Event1 and Event2. 3. Call **InstallProtocolInterface()** to install the specified protocol. 4. The two Events should not be signaled.
5.1.3.4.41	0xd642220c, 0x6d31, 0x4676, 0x96, 0xf0, 0xb0, 0x55, 0x1c, 0xdc, 0xa2, 0xf2	**BS.RegisterProtocolNotify -** Events that have been registered for protocol interface notification can be unregistered by calling **CloseEvent()**.	5. Call **ReInstallProtocolInterface()** to install the specified protocol. 6. The two Events should not be signaled.

3.3.5 LocateHandle()

Number	GUID	Assertion	Test Description
5.1.3.5.1	0x52d5cdec, 0xf9cf, 0x4a48, 0x86, 0x4b, 0x87, 0x9e, 0x92, 0xe5, 0x1a, 0x3b	**BS.LocateHandle – LocateHandle()** returns **EFI_INVALID_PARAMETER** with invalid search type	1. Call **LocateHandle()** with search type other than **AllHandles**, **ByRegisterNotify** and **ByProtocol**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.5.2	0x6cad11b3, 0x9ea5, 0x4d60, 0xb0, 0x6c, 0xaf, 0xf3, 0xfd, 0xef, 0x90, 0x8d	**BS.LocateHandle – LocateHandle()** returns **EFI_INVALID_PARAMETER** with *SearchKey* is **NULL** when searching **ByRegisterNotify**	1. Call **LocateHandle()** with search type **ByRegisterNotify**, but the *SearchKey* is **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.5.3	0x3b59cad8, 0x4c97, 0x49b2, 0xbb, 0xfa, 0x9f, 0x15, 0x6a, 0x3e, 0x7f, 0x44	**BS.LocateHandle – LocateHandle()** returns **EFI_NOT_FOUND** with a never installed protocol	1. Call **LocateHandle()** to locate the handles for a never installed protocol. The return code should be **EFI_NOT_FOUND**.
5.1.3.5.4	0x40a82fe1, 0x7c20, 0x4307, 0xa4, 0x3b, 0xfa, 0x6e, 0x21, 0x16, 0x2c, 0xdb	**BS.LocateHandle – LocateHandle()** returns **EFI_BUFFER_TOO_SMALLEFI_BUFFER_TOO_SMALL** with *Buffer* size is 0	1. Call **LocateHandle()** to locate all handles with 0 length handle buffer. The return code should be **EFI_BUFFER_TOO_SMALL EFI_BUFFER_TOO_SMALL**.
5.1.3.5.5	0xa66db8d1, 0x6ea7, 0x40c2, 0x99, 0x8c, 0xd3, 0xc6, 0xc8, 0xff, 0x33, 0xe6	**BS.LocateHandle – LocateHandles()** sets the required buffer size with *Buffer* size is 0	1. Call **LocateHandle()** to locate all handles with 0 length handle buffer. The buffer size is updated to the size of the buffer needed to obtain the handle array.
5.1.3.5.6	0x11449d53, 0xa735, 0x45b2, 0xa7, 0x81, 0xb6, 0x0f, 0x22, 0x73, 0x46, 0x0f	**BS.LocateHandle – LocateHandle()** returns **EFI_BUFFER_TOO_SMALLEFI_BUFFER_TOO_SMALL** with *Buffer* size less than the required.	1. Call **LocateHandle()** to locate all handles with the required buffer size – 1 length handle buffer. The return code should be **EFI_BUFFER_TOO_SMALL EFI_BUFFER_TOO_SMALL**.
5.1.3.5.7	0xf7d46144, 0x290c, 0x48da, 0xad, 0x11, 0xca, 0x67, 0x8e, 0xa5, 0xab, 0x1b	**BS.LocateHandle – LocateHandle()** sets the required buffer size with *Buffer* size less than the required.	1. Call **LocateHandle()** to locate all handles with the required buffer size – 1 length handle buffer. The buffer size is updated to the size of the buffer needed to obtain the handle array.

Number	GUID	Assertion	Test Description
5.1.3.5.8	0x69eec7bb, 0x55d6, 0x475f, 0xbc, 0x57, 0x2e, 0xaf, 0xe4, 0x8c, 0x52, 0x0f	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **AllHandles** at **EFI_TPL_APPLICATION**	1. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system. The return code should be **EFI_SUCCESS**.
5.1.3.5.9	0xb8cd32a7, 0x7a94, 0x4c75, 0xbc, 0x8a, 0x2b, 0x72, 0xec, 0xb5, 0xe8, 0x62	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **AllHandles** at **EFI_TPL_CALLBACK**	1. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system. The return code should be **EFI_SUCCESS**.
5.1.3.5.10	0xfdea67c6, 0x6cb8, 0x4d0f, 0xa5, 0x5c, 0xfe, 0xd3, 0x73, 0xac, 0x18, 0xd1	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **AllHandles** at **EFI_TPL_NOTIFY**	1. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system. The return code should be **EFI_SUCCESS**.
5.1.3.5.11	0x25ee90ed, 0x3cf6, 0x4c1c, 0xa3, 0xad, 0x82, 0x33, 0xaf, 0x05, 0x0b, 0x77	**BS.LocateHandle – LocateHandle()** locates all handles at **EFI_TPL_APPLICATION**	1. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handle. The return code should be **EFI_SUCCESS**.
5.1.3.5.12	0x0129241e, 0x0b63, 0x47ba, 0x9d, 0xd5, 0xdc, 0xb5, 0x8a, 0x4e, 0x62, 0x60	**BS.LocateHandle – LocateHandle()** locates all handles at **EFI_TPL_CALLBACK**	1. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handle. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.5.13	0xdc3cff6a, 0x86d2, 0x4dc4, 0x85, 0x25, 0x06, 0x81, 0x81, 0xb3, 0xe6, 0x87	`BS.LocateHandle – LocateHandle()` locates all handles at `EFI_TPL_NOTIFY`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. The return code should be `EFI_SUCCESS`.
5.1.3.5.14	0x3c3e2f8f, 0xe33f, 0x4ef1, 0x99, 0xa7, 0xb2, 0x37, 0xf2, 0xea, 0x2c, 0xab	`BS.LocateHandle – LocateHandle()` locates all handles at `EFI_TPL_APPLICATION`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. The return code should be `EFI_SUCCESS`.
5.1.3.5.15	0x48dc0c46, 0x053a, 0x4314, 0xa9, 0xa3, 0x34, 0x4c, 0xe2, 0xc8, 0x57, 0xec	`BS.LocateHandle – LocateHandle()` locates all handles at `EFI_TPL_CALLBACK`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.5.16	0xd5de5eaa, 0x71ab, 0x4caf, 0xb7, 0xe0, 0x4a, 0x87, 0x10, 0x65, 0xbb, 0x55	`BS.LocateHandle –` `LocateHandle()` locates all handles at `EFI_TPL_NOTIFY`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. The return code should be `EFI_SUCCESS`.
5.1.3.5.17	0xd7fa21f2, 0xbe25, 0x4696, 0x87, 0x55, 0xef, 0xa8, 0x50, 0x30, 0xc8, 0x78	`BS.LocateHandle –` `LocateHandle()` locates all handles at `EFI_TPL_APPLICATION`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. The number of handles of the system increases by 1.
5.1.3.5.18	0xa82151e4, 0x5b2a, 0x475b, 0xa5, 0xe0, 0x6a, 0x75, 0x9c, 0xed, 0x22, 0x93	`BS.LocateHandle –` `LocateHandle()` locates all handles at `EFI_TPL_CALLBACK`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. The number of handles of the system increases by 1.

Number	GUID	Assertion	Test Description
5.1.3.5.19	0xf3787309, 0xb7c9, 0x418b, 0xb3, 0xa5, 0x28, 0x42, 0x61, 0xc5, 0x17, 0xf6	**BS.LocateHandle – LocateHandle()** locates all handles at **EFI_TPL_NOTIFY**	1. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handle. 3. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system again. The number of handles of the system increases by 1.
5.1.3.5.20	0x096eaa87, 0x17c3, 0x43c1, 0x82, 0x00, 0x8d, 0xfd, 0x93, 0x45, 0xee, 0xe5	**BS.LocateHandle – LocateHandle()** locates all handles at **EFI_TPL_APPLICATION**	1. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handle. 3. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system again. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.5.21	0xf67331e1, 0x7881, 0x47b5, 0xa5, 0xc6, 0xd9, 0x0d, 0xa0, 0x52, 0x45, 0xd3	`BS.LocateHandle – LocateHandle()` locates all handles at `EFI_TPL_CALLBACK`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. The return code should be `EFI_SUCCESS`.
5.1.3.5.22	0xfc881982, 0x3387, 0x4aae, 0x98, 0xd8, 0x31, 0x78, 0xf6, 0xee, 0x66, 0x5d	`BS.LocateHandle – LocateHandle()` locates all handles at `EFI_TPL_NOTIFY`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.5.23	0xa03b492d, 0x40a3, 0x4726, 0xb5, 0xb9, 0x82, 0x84, 0x2b, 0xae, 0x77, 0x56	`BS.LocateHandle – LocateHandle()` locates all handles at `EFI_TPL_APPLICATION`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. The return code should be `EFI_SUCCESS`.
5.1.3.5.24	0xa47869b0, 0x45f2, 0x47c3, 0xb0, 0xa3, 0xac, 0x53, 0xee, 0xe4, 0x94, 0x1f	`BS.LocateHandle – LocateHandle()` locates all handles at `EFI_TPL_CALLBACK`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.5.25	0x34127434, 0x40c5, 0x4f9e, 0xb1, 0x45, 0x5b, 0x7f, 0x3f, 0x88, 0x6a, 0x8f	`BS.LocateHandle – LocateHandle()` locates all handles at `EFI_TPL_NOTIFY`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. The return code should be `EFI_SUCCESS`.
5.1.3.5.26	0x598cd1aa, 0xe3d2, 0x4cae, 0x9e, 0x44, 0xa1, 0x9d, 0xbc, 0x72, 0xed, 0x89	`BS.LocateHandle – LocateHandle()` locates all handles at `EFI_TPL_APPLICATION`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. The number of handles of the system decreases by 1.

Number	GUID	Assertion	Test Description
5.1.3.5.27	0x487d12ed, 0xdc96, 0x41a1, 0x8c, 0xc1, 0xc6, 0xe3, 0x74, 0x54, 0x6a, 0xd7	`BS.LocateHandle – LocateHandle()` locates all handles at `EFI_TPL_CALLBACK`	1. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `LocateHandle()` via search type `AllHandles` to retrieve all handles in the system again. The number of handles of the system decreases by 1.

Number	GUID	Assertion	Test Description
5.1.3.5.28	0xd76dedf9, 0xe98e, 0x473b, 0x87, 0xa1, 0x76, 0x62, 0x56, 0x84, 0x46, 0x85	**BS.LocateHandle – LocateHandle()** locates all handles at **EFI_TPL_NOTIFY**	1. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handle. 3. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system again. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system again. The number of handles of the system decreases by 1.
5.1.3.5.29	0x278161f9, 0xbfdc, 0x4627, 0xb1, 0x1e, 0x7c, 0x64, 0x55, 0x92, 0x73, 0xfd	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **ByRegisterNotify** at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto 10 new handles. The return code should be **EFI_SUCCESS**.
5.1.3.5.30	0x4f61b8d3, 0xb78d, 0x42f7, 0x8c, 0x47, 0xab, 0x66, 0x0f, 0x93, 0x87, 0x6b	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **ByRegisterNotify** at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto 10 new handles. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.5.31	0x05c8a6c6, 0x0629, 0x46a5, 0x86, 0x72, 0xfc, 0xd5, 0x8a, 0x24, 0xa1, 0xdf	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **ByRegisterNotify** at **EFI_TPL_NOTIFY**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto 10 new handles. The return code should be **EFI_SUCCESS**.
5.1.3.5.32	0xe971ed0a, 0xe0ea, 0x48db, 0xae, 0x13, 0x53, 0x2e, 0xda, 0xd6, 0xbc, 0xc7	**BS.LocateHandle – LocateHandle()** locates the new register handle at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto 10 new handles. 3. Call **LocateHandle()** 10 times via search type **ByRegisterNotify** with the search key generated by previous **RegisterProtocolNotify**. The return code should be **EFI_SUCCESS**.
5.1.3.5.33	0x9022c21e, 0x153d, 0x443d, 0xa5, 0x6a, 0x72, 0x3c, 0x02, 0xae, 0x5b, 0x7d	**BS.LocateHandle – LocateHandle()** locates the new register handle at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto 10 new handles. 3. Call **LocateHandle()** 10 times via search type **ByRegisterNotify** with the search key generated by previous **RegisterProtocolNotify**. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.5.34	0xa24c8d25, 0x8b4a, 0x4e65, 0x9a, 0x91, 0x3f, 0x8b, 0x72, 0x60, 0x42, 0x90	`BS.LocateHandle – LocateHandle()` locates the new register handle at `EFI_TPL_NOTIFY`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto 10 new handles. 3. Call `LocateHandle()` 10 times via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. The return code should be `EFI_SUCCESS`.
5.1.3.5.35	0x023ac3c9, 0x3305, 0x45d4, 0xa0, 0x20, 0x74, 0x71, 0x33, 0xf3, 0x66, 0xc1	`BS.LocateHandle – LocateHandle()` locates the new register handle at `EFI_TPL_APPLICATION`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto 10 new handles. 3. Call `LocateHandle()` 10 times via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. The return *`BufferSize`* should be the size of `(EFI_HANDLE)`.

Number	GUID	Assertion	Test Description
5.1.3.5.36	0x18ab1f0c, 0x6972, 0x436d, 0x9d, 0x7b, 0xea, 0x35, 0x13, 0xaa, 0x09, 0x19	`BS.LocateHandle – LocateHandle()` locates the new register handle at `EFI_TPL_CALLBACK`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto 10 new handles. 3. Call `LocateHandle()` 10 times via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. The return *`BufferSize`* should be the size of `(EFI_HANDLE)`.
5.1.3.5.37	0xf4bd2b49, 0xa409, 0x42d8, 0xa1, 0xe6, 0xe9, 0xdd, 0x0e, 0x1d, 0xca, 0x0e	`BS.LocateHandle – LocateHandle()` locates the new register handle at `EFI_TPL_NOTIFY`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto 10 new handles. 3. Call `LocateHandle()` 10 times via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. The return *`BufferSize`* should be the size of `(EFI_HANDLE)`.

Number	GUID	Assertion	Test Description
5.1.3.5.38	0xd913ed57, 0xd7d9, 0x4108, 0x92, 0x66, 0x71, 0x10, 0x28, 0x1f, 0xd5, 0x9a	BS.LocateHandle – LocateHandle() locates the new register handle at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() to register for TestProtocol1's installation. 2. Call InstallProtocolInterface() to install TestProtocol1 onto 10 new handles. 3. Call LocateHandle() 10 times via search type ByRegisterNotify with the search key generated by previous RegisterProtocolNotify. The return handle should be the new created handle.
5.1.3.5.39	0xbf1d6210, 0x96e2, 0x4417, 0xb7, 0xe9, 0x9f, 0xba, 0x62, 0x20, 0x32, 0xe0	BS.LocateHandle – LocateHandle() locates the new register handle at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() to register for TestProtocol1's installation. 2. Call InstallProtocolInterface() to install TestProtocol1 onto 10 new handles. 3. Call LocateHandle() 10 times via search type ByRegisterNotify with the search key generated by previous RegisterProtocolNotify. The return handle should be the new created handle.

Number	GUID	Assertion	Test Description
5.1.3.5.40	0x05f0c339, 0xce7e, 0x4e51, 0xb7, 0xbe, 0xd6, 0x2d, 0xbe, 0x34, 0x04, 0x27	`BS.LocateHandle – LocateHandle()` locates the new register handle at `EFI_TPL_NOTIFY`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto 10 new handles. 3. Call `LocateHandle()` 10 times via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. The return handle should be the new created handle.
5.1.3.5.41	0x7a7b904c, 0x600a, 0x41d0, 0xb0, 0x19, 0x06, 0x5d, 0xee, 0x14, 0x3d, 0xf8	`BS.LocateHandle – LocateHandle()` locates the new register handle at `EFI_TPL_APPLICATION`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto 10 new handles. 3. Call `LocateHandle()` 10 times via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. 4. Call `LocateHandle()` again. The return code should be `EFI_NOT_FOUND`.

Number	GUID	Assertion	Test Description
5.1.3.5.42	0x94b01d4c, 0x149f, 0x4750, 0xa3, 0x61, 0x37, 0x6c, 0xcd, 0xf6, 0x2f, 0xcf	**BS.LocateHandle –** `LocateHandle()` locates the new register handle at `EFI_TPL_CALLBACK`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto 10 new handles. 3. Call `LocateHandle()` 10 times via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. 4. Call `LocateHandle()` again. The return code should be `EFI_NOT_FOUND`.
5.1.3.5.43	0x9eb4e947, 0xdac3, 0x4b24, 0xa1, 0xd3, 0x6f, 0x5a, 0xb0, 0x02, 0x09, 0x10	**BS.LocateHandle –** `LocateHandle()` locates the new register handle at `EFI_TPL_NOTIFY`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto 10 new handles. 3. Call `LocateHandle()` 10 times via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. 4. Call `LocateHandle()` again. The return code should be `EFI_NOT_FOUND`.
5.1.3.5.44	0xe42ce5bb, 0x0c74, 0x4fde, 0x99, 0x71, 0xcc, 0xfe, 0x1d, 0x21, 0x0d, 0xb3	**BS.LocateHandle –** `LocateHandle()` returns `EFI_SUCCESS` with a *Type* value of `ByProtocol` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto 10 new handles. `InstallProtocolInterface()` return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.5.45	0x8b4d0f9e, 0x80a0, 0x451a, 0x88, 0x04, 0x86, 0x22, 0x04, 0x51, 0xfa, 0xce	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **ByProtocol** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto 10 new handles. **InstallProtocolInterface()** return code should be **EFI_SUCCESS**.
5.1.3.5.46	0x9eb47d37, 0xc0c1, 0x48a3, 0x85, 0x2c, 0x26, 0xad, 0x0e, 0x33, 0xe6, 0x55	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **ByProtocol** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto 10 new handles. **InstallProtocolInterface()** return code should be **EFI_SUCCESS**.
5.1.3.5.47	0x1f8ec2e8, 0x5597, 0x4c45, 0xb5, 0x50, 0xf9, 0x31, 0xc8, 0x2f, 0x0b, 0x50	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **ByProtocol** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return code should be **EFI_SUCCESS**.
5.1.3.5.48	0xcd2fd544, 0x58ea, 0x4bef, 0x9c, 0xd0, 0xa4, 0x6b, 0xfc, 0x43, 0xc9, 0xf2	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **ByProtocol** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return code should be **EFI_SUCCESS**.
5.1.3.5.49	0xe72afb35, 0xd416, 0x4dcc, 0x9a, 0x87, 0x9b, 0x29, 0x64, 0xb9, 0x04, 0x6e	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **ByProtocol** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return code should be **EFI_SUCCESS**.
5.1.3.5.50	0x5437505f, 0x064f, 0x4b19, 0x97, 0x06, 0xba, 0xbe, 0x19, 0x3e, 0xa8, 0xcc	**BS.LocateHandle – LocateHandle()** locates handles by protocol at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return handle number should be 10.

Number	GUID	Assertion	Test Description
5.1.3.5.51	0x50aa234f, 0x9140, 0x4016, 0x83, 0x2f, 0x53, 0xb6, 0xd4, 0x60, 0x40, 0x91	**BS.LocateHandle – LocateHandle()** locates handles by protocol at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return handle number should be 10.
5.1.3.5.52	0x35dfcf9e, 0xfae6, 0x4715, 0x81, 0x85, 0xff, 0xa3, 0x4a, 0xda, 0x2f, 0x14	**BS.LocateHandle – LocateHandle()** locates handles by protocol at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return handle number should be 10.
5.1.3.5.53	0x342ed823, 0x9e57, 0x46bd, 0x9f, 0x9f, 0x8b, 0x08, 0x64, 0x75, 0x50, 0x05	**BS.LocateHandle – LocateHandle()** locates handles by protocol at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return handles should equal to those created.
5.1.3.5.54	0xa151beda, 0x5e43, 0x46c7, 0x9d, 0xa6, 0xf0, 0xcb, 0x59, 0x2a, 0x0f, 0x03	**BS.LocateHandle – LocateHandle()** locates handles by protocol at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return handles should equal to those created.
5.1.3.5.55	0xedf89e16, 0x81cf, 0x4202, 0x88, 0x95, 0xab, 0x94, 0xaa, 0x4e, 0xe6, 0x47	**BS.LocateHandle – LocateHandle()** locates handles by protocol at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return handles should equal to those created.

Number	GUID	Assertion	Test Description
5.1.3.5.56	0xd96a0071, 0x3e0c, 0x4ad5, 0xbd, 0x2a, 0x8c, 0x2a, 0x19, 0x01, 0xa6, 0x31	**BS.LocateHandle – LocateHandle()** locates handles by protocol at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. **TestProtocol1** should be located via each return handle.
5.1.3.5.57	0x902adedd, 0x58cc, 0x4f3d, 0x95, 0x9b, 0x7e, 0x4d, 0xcb, 0x60, 0xea, 0x2d	**BS.LocateHandle – LocateHandle()** locates handles by protocol at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. **TestProtocol1** should be located via each return handle.
5.1.3.5.58	0x98d1053f, 0xb223, 0x48d2, 0x82, 0x5a, 0x73, 0xc8, 0x85, 0x35, 0x2a, 0xfb	**BS.LocateHandle – LocateHandle()** locates handles by protocol at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandle()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. **TestProtocol1** should be located via each return handle.
5.1.3.5.59	0x552ccd79, 0x14bd, 0x45d0, 0x8a, 0x0f, 0x86, 0xb0, 0x30, 0x85, 0xb2, 0x63	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **ByRegisterNotify** at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.5.60	0xf9b1720f, 0x6916, 0x41a1, 0x86, 0xd0, 0x2f, 0x79, 0x28, 0xad, 0x2b, 0x80	`BS.LocateHandle – LocateHandle()` returns `EFI_SUCCESS` with a *Type* value of `ByRegisterNotify` at `EFI_TPL_CALLBACK`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. The return code should be `EFI_SUCCESS`.
5.1.3.5.61	0x7043b8ef, 0x7bd5, 0x4ecc, 0x95, 0x1e, 0xde, 0x3f, 0x6f, 0xcf, 0xbd, 0x17	`BS.LocateHandle – LocateHandle()` returns `EFI_SUCCESS` with a *Type* value of `ByRegisterNotify` at `EFI_TPL_NOTIFY`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. The return code should be `EFI_SUCCESS`.
5.1.3.5.62	0x28a0256f, 0x95a3, 0x4050, 0x87, 0x9d, 0x99, 0xd2, 0x29, 0xbb, 0xcc, 0x95	`BS.LocateHandle – LocateHandle()` returns `EFI_SUCCESS` with a *Type* value of `ByRegisterNotify` at `EFI_TPL_APPLICATION`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandle()` via search type `ByRegisterNotify`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.5.63	0x46f1b43a, 0x1943, 0x401c, 0x95, 0xce, 0xe0, 0x0a, 0x8e, 0x84, 0xd9, 0x73	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **ByRegisterNotify** at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. The return code should be **EFI_SUCCESS**.
5.1.3.5.64	0x05219d9d, 0x0e3b, 0x4336, 0xba, 0x98, 0x04, 0xc9, 0xb5, 0xbe, 0x12, 0xa7	**BS.LocateHandle – LocateHandle()** returns **EFI_SUCCESS** with a *Type* value of **ByRegisterNotify** at **EFI_TPL_NOTIFY**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. The return code should be **EFI_SUCCESS**.
5.1.3.5.65	0xe1f78301, 0x0106, 0x4088, 0xa9, 0x4c, 0x4c, 0x25, 0x14, 0x98, 0xa4, 0x5a	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. The return *BufferSize* should be the size of (**EFI_HANDLE**).

Number	GUID	Assertion	Test Description
5.1.3.5.66	0x11e8389a, 0x3d37, 0x48d0, 0xa1, 0x50, 0xf9, 0x05, 0x03, 0x49, 0x90, 0xca	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. The return *BufferSize* should be the size of (**EFI_HANDLE**).
5.1.3.5.67	0xa5ed261d, 0x73aa, 0x4ef0, 0x8f, 0x3c, 0xbe, 0x2e, 0xae, 0xe2, 0xcc, 0xe9	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_NOTIFY**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. The return *BufferSize* should be the size of (**EFI_HANDLE**).
5.1.3.5.68	0x849585d5, 0x1f53, 0x450c, 0x81, 0x70, 0xb1, 0x70, 0xbd, 0x29, 0x5b, 0x1c	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. The return handles should be matched.

Number	GUID	Assertion	Test Description
5.1.3.5.69	0x2932e563, 0xe4dd, 0x4ea8, 0xb0, 0xfa, 0xb1, 0x6a, 0x34, 0x4d, 0x9b, 0x74	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. The return handles should be matched.
5.1.3.5.70	0xc415861b, 0xb3f3, 0x44dd, 0xbd, 0x40, 0xad, 0xda, 0x37, 0x54, 0x01, 0xb6	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_NOTIFY**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. The return handles should be matched.

Number	GUID	Assertion	Test Description
5.1.3.5.71	0x2a646138, 0x4526, 0x484a, 0x81, 0xb6, 0x2e, 0x27, 0xd1, 0xe2, 0xb2, 0xf0	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles again. The return code should be **EFI_SUCCESS**.
5.1.3.5.72	0xad4cd436, 0x3b5c, 0x491e, 0x96, 0x79, 0xb4, 0x88, 0xea, 0x1f, 0xf8, 0x90	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles again. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.5.73	0x8d4d2c27, 0x0cfc, 0x483a, 0xa6, 0xda, 0xce, 0x8b, 0xc5, 0xdc, 0x8f, 0xaf	BS.LocateHandle – LocateHandle() locates new register handle at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() to register for TestProtocol1's installation. 2. Call InstallProtocolInterface() to install TestProtocol1 onto a new handles. 3. Call LocateHandle() via search type ByRegisterNotify. 4. Call UninstallProtocolInterface() to uninstall TestProtocol1. 5. Call InstallProtocolInterface() to install TestProtocol1 onto a new handles again. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.5.74	0x09b908f2, 0x81da, 0x4dbd, 0x9a, 0x1f, 0x5b, 0xa8, 0xca, 0x47, 0x36, 0x32	`BS.LocateHandle – LocateHandle()` locates new register handle at `EFI_TPL_APPLICATION`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandle()` via search type `ByRegisterNotify`. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles again. 6. Call `LocateHandle()` via search type `ByRegisterNotify`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.5.75	0x837de4c2, 0xdd2c, 0x4739, 0xad, 0xdf, 0xa9, 0xef, 0xb4, 0xc8, 0xf0, 0x6a	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type "ByRegisterNotify. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles again. 6. Call **LocateHandle()** via search type **ByRegisterNotify**. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.5.76	0xfe439c44, 0x1f30, 0x465e, 0x9a, 0x91, 0x3a, 0x06, 0x7d, 0x06, 0xd2, 0x98	`BS.LocateHandle – LocateHandle()` locates new register handle at `EFI_TPL_NOTIFY`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`’s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandle()` via search type `ByRegisterNotify`. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles again. 6. Call `LocateHandle()` via search type `ByRegisterNotify`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.5.77	0xe73f9a4d, 0x3d43, 0x48e8, 0xab, 0xe4, 0x08, 0xc0, 0x64, 0xef, 0xeb, 0x28	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles again. 6. Call **LocateHandle()** via search type **ByRegisterNotify**. The return *BufferSize* should be the size of **(EFI_HANDLE)**.

Number	GUID	Assertion	Test Description
5.1.3.5.78	0xd4336a63, 0xa8a5, 0x48ff, 0xa4, 0x52, 0x7b, 0x9b, 0x44, 0x24, 0x4e, 0x3a	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles again. 6. Call **LocateHandle()** via search type **ByRegisterNotify**. The return *BufferSize* should be the size of **(EFI_HANDLE)**.

Number	GUID	Assertion	Test Description
5.1.3.5.79	0xa1d137fa, 0x3270, 0x4d3e, 0x92, 0x0b, 0xd9, 0x3f, 0x31, 0x4f, 0x39, 0x43	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_NOTIFY**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles again. 6. Call **LocateHandle()** via search type **ByRegisterNotify**. The return *BufferSize* should be the size of **(EFI_HANDLE)**.

Number	GUID	Assertion	Test Description
5.1.3.5.80	0x4f8f1009, 0xe23f, 0x41e3, 0x82, 0xb7, 0xf0, 0xbb, 0x96, 0x5a, 0xda, 0xca	`BS.LocateHandle – LocateHandle()` locates new register handle at `EFI_TPL_APPLICATION`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandle()` via search type `ByRegisterNotify`. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles again. 6. Call `LocateHandle()` via search type `ByRegisterNotify`. The return handles should be matched.

Number	GUID	Assertion	Test Description
5.1.3.5.81	0x621afecb, 0xd170, 0x4a19, 0x92, 0x3f, 0xa4, 0xf1, 0xd3, 0x8b, 0x0f, 0x81	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles again. 6. Call **LocateHandle()** via search type **ByRegisterNotify**. The return handles should be matched.

Number	GUID	Assertion	Test Description
5.1.3.5.82	0x77efed09, 0xb369, 0x40bd, 0x99, 0xa4, 0x27, 0x61, 0xba, 0xb6, 0xbf, 0x1b	`BS.LocateHandle – LocateHandle()` locates new register handle at `EFI_TPL_NOTIFY`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandle()` via search type `ByRegisterNotify`. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles again. 6. Call `LocateHandle()` via search type `ByRegisterNotify`. The return handles should be matched.

Number	GUID	Assertion	Test Description
5.1.3.5.83	0xf927d0b9, 0x0d7d, 0x4e89, 0x8f, 0xd7, 0x04, 0x2a, 0x4c, 0xeb, 0xd9, 0xbe	**BS.LocateHandle – LocateHandle()** locates new register handle at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandle()** via search type **ByRegisterNotify**. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles again. 6. Call **LocateHandle()** via search type **ByRegisterNotify**. 7. Call **LocateHandle()** again. The return code should be **EFI_NOT_FOUND**.

Number	GUID	Assertion	Test Description
5.1.3.5.84	0x9162854c, 0x7516, 0x4a9e, 0xb7, 0x57, 0x04, 0xf6, 0x88, 0x3e, 0x7c, 0x8b	`BS.LocateHandle – LocateHandle()` locates new register handle at `EFI_TPL_CALLBACK`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandle()` via search type `ByRegisterNotify`. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles again. 6. Call `LocateHandle()` via search type `ByRegisterNotify`. 7. Call `LocateHandle()` again. The return code should be `EFI_NOT_FOUND`.

Number	GUID	Assertion	Test Description
5.1.3.5.85	0x5c391bcb, 0xcdaf, 0x45c5, 0xab, 0x2d, 0xbb, 0x72, 0x98, 0x01, 0x4c, 0xb6	`BS.LocateHandle – LocateHandle()` locates new register handle at `EFI_TPL_NOTIFY`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandle()` via search type `ByRegisterNotify`. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles again. 6. Call `LocateHandle()` via search type `ByRegisterNotify`. 7. Call `LocateHandle()` again. The return code should be `EFI_NOT_FOUND`.

3.3.6 HandleProtocol()

Number	GUID	Assertion	Test Description
5.1.3.6.1	0xbb124c57, 0x654a, 0x44e2, 0x91, 0x25, 0x9b, 0x65, 0x46, 0xba, 0xc1, 0x10	**BS.HandleProtocol – HandleProtocol()** returns **EFI_INVALID_PARAMETER** with invalid handle	1. Call **HandleProtocol()** with invalid handle (*Handle* = **NULL** or *Handle* is invalid). Each return code should be **EFI_INVALID_PARAMETER**.
5.1.3.6.2	0xeb5fc568, 0x67f1, 0x412a, 0xa2, 0xce, 0xe4, 0xad, 0x11, 0xef, 0xbd, 0x27	**BS.HandleProtocol – HandleProtocol()** returns **EFI_INVALID_PARAMETER** with **NULL** protocol	1. Call **HandleProtocol()** with **NULL** protocol GUID. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.6.3	0x3257ddd0, 0xe28c, 0x4f2e, 0xac, 0xf3, 0x52, 0x9a, 0x87, 0x38, 0x64, 0x27	**BS.HandleProtocol – HandleProtocol()** returns **EFI_INVALID_PARAMETER** with **NULL** interface	1. Call **HandleProtocol()** with **NULL** interface. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.6.4	0x25ece62d, 0x5c0e, 0x4f33, 0x9e, 0x55, 0xe3, 0xbb, 0x12, 0x2d, 0x8d, 0x8f	**BS.HandleProtocol – HandleProtocol()** returns **EFI_UNSUPPORTED** with never installed protocol	1. Call **HandleProtocol()** to attempt to retrieve a protocol instance that was never installed on the handle. The return code should be **EFI_UNSUPPORTED**.
5.1.3.6.5	0x8696c014, 0x6bd7, 0x4a98, 0xa1, 0xdd, 0xeb, 0x07, 0xc0, 0x1a, 0xbd, 0x15	**BS.HandleProtocol – HandleProtocol()** locates protocol from handle at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **HandleProtocol()** to attempt to retrieve **TestProtocol1** from the handle. The return code should be **EFI_SUCCESS**.
5.1.3.6.6	0x752790d2, 0xf46a, 0x4956, 0x9b, 0x78, 0xc0, 0x54, 0x6f, 0x26, 0x44, 0xb5	**BS.HandleProtocol – HandleProtocol()** locates protocol from handle at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **HandleProtocol()** to attempt to retrieve **TestProtocol1** from the handle. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.6.7	0x30e46bfd, 0xe3b9, 0x4196, 0x8e, 0xa7, 0xcc, 0xd8, 0xc0, 0x75, 0x93, 0x3f	`BS.HandleProtocol – HandleProtocol()` locates protocol from handle at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `HandleProtocol()` to attempt to retrieve `TestProtocol1` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.6.8	0xa4b84540, 0xa81c, 0x44f0, 0xb3, 0xbe, 0xae, 0x9c, 0xda, 0xd0, 0x80, 0xbf	`BS.HandleProtocol – HandleProtocol()` locates protocol from handle at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `HandleProtocol()` to attempt to retrieve `TestProtocol1` from the handle. The `TestProtocol1`'s function should be accessed and executed correctly.
5.1.3.6.9	0x8e0b5eea, 0x8f0b, 0x46e3, 0xa6, 0xa3, 0x20, 0xfa, 0x7c, 0xfa, 0xde, 0x3c	`BS.HandleProtocol – HandleProtocol()` locates protocol from handle at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `HandleProtocol()` to attempt to retrieve `TestProtocol1` from the handle. The `TestProtocol1`'s function should be accessed and executed correctly.
5.1.3.6.10	0xf58819f0, 0xc0c8, 0x4583, 0xb0, 0x07, 0x67, 0x08, 0x07, 0xc5, 0x71, 0x88	`BS.HandleProtocol – HandleProtocol()` locates protocol from handle at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `HandleProtocol()` to attempt to retrieve `TestProtocol1` from the handle. The `TestProtocol1`'s function should be accessed and executed correctly.

Number	GUID	Assertion	Test Description
5.1.3.6.11	0x00c5156d, 0x6b47, 0x441a, 0xb2, 0x97, 0x9b, 0xb0, 0x83, 0x07, 0x42, 0x76	**BS.HandleProtocol – HandleProtocol()** locates protocol from handle at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **HandleProtocol()** to attempt to retrieve **TestProtocol1** from the handle. 3. Reinstall **TestProtocol1** onto the handle. 4. Call **HandleProtocol()** again. The return code should be **EFI_SUCCESS**.
5.1.3.6.12	0x0b4e7e97, 0xcb38, 0x48a2, 0xb9, 0x2a, 0x16, 0x1a, 0x93, 0x5f, 0x5b, 0x05	**BS.HandleProtocol – HandleProtocol()** locates protocol from handle at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **HandleProtocol()** to attempt to retrieve **TestProtocol1** from the handle. 3. Reinstall **TestProtocol1** onto the handle. 4. Call **HandleProtocol()** again. The return code should be **EFI_SUCCESS**.
5.1.3.6.13	0x0bc2127b, 0xcaf7, 0x4073, 0xa3, 0x9b, 0x42, 0x7b, 0x16, 0x56, 0x82, 0x02	**BS.HandleProtocol – HandleProtocol()** locates protocol from handle at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **HandleProtocol()** to attempt to retrieve **TestProtocol1** from the handle. 3. Reinstall **TestProtocol1** onto the handle. 4. Call **HandleProtocol()** again. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.6.14	0xd1a554d5, 0x07d0, 0x437b, 0x82, 0xa2, 0xbb, 0xa3, 0x67, 0xc8, 0x58, 0xec	**BS.HandleProtocol – HandleProtocol()** locates protocol from handle at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **HandleProtocol()** to attempt to retrieve **TestProtocol1** from the handle. 3. Reinstall **TestProtocol1** onto the handle. 4. Call **HandleProtocol()** again. The new **TestProtocol1**'s function should be accessed and executed correctly.
5.1.3.6.15	0x8cae93e7, 0x438e, 0x4c9f, 0x99, 0xc7, 0x7c, 0x20, 0x87, 0x25, 0xd8, 0xca	**BS.HandleProtocol – HandleProtocol()** locates protocol from handle at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **HandleProtocol()** to attempt to retrieve **TestProtocol1** from the handle. 3. Reinstall **TestProtocol1** onto the handle. 4. Call **HandleProtocol()** again. The new **TestProtocol1**'s function should be accessed and executed correctly.
5.1.3.6.16	0x7884805e, 0x6660, 0x4e8e, 0xab, 0x32, 0xa6, 0xf5, 0x70, 0xc1, 0x8c, 0xcd	**BS.HandleProtocol – HandleProtocol()** locates protocol from handle at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **HandleProtocol()** to attempt to retrieve **TestProtocol1** from the handle. 3. Reinstall **TestProtocol1** onto the handle. 4. Call **HandleProtocol()** again. The new **TestProtocol1**'s function should be accessed and executed correctly.

3.3.7 LocateDevicePath()

Number	GUID	Assertion	Test Description
5.1.3.7.1	0x1657bf8a, 0x005e, 0x46c5, 0xa1, 0xb4, 0x93, 0x84, 0x81, 0xa4, 0x3b, 0x6a	**BS.LocateDevicePath – LocateDevicePath()** returns **EFI_INVALID_PARAMETER** with **NULL** protocol	1. Call **LocateDevicePath()** with protocol GUID pointer be **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.7.2	0xef52e7d7, 0x6346, 0x48e0, 0xa6, 0x4c, 0x78, 0x71, 0x87, 0x52, 0x18, 0x8d	**BS.LocateDevicePath – LocateDevicePath()** returns **EFI_NOT_FOUND** with never installed protocol	1. Call LocateDevicePath() to search for a handle with a never installed protocol. The return code should be **EFI_NOT_FOUND**.
5.1.3.7.3	0xd71106c1, 0xfbdb, 0x4ada, 0xbf, 0x69, 0xf1, 0xde, 0x57, 0x2d, 0x29, 0x6a	**BS.LocateDevicePath – LocateDevicePath()** returns **EFI_NOT_FOUND** with never installed protocol and a NULL input device.	1. Call **LocateDevicePath()** to search for a handle with a never installed protocol and a NULL input device. The return code should be **EFI_NOT_FOUND**.
5.1.3.7.4	0xbc272c41, 0x030c, 0x443d, 0xaa, 0xbf, 0x90, 0xd4, 0x50, 0x9e, 0xf7, 0xb3	**BS.LocateDevicePath – LocateDevicePath()** returns **EFI_INVALID_PARAMETER** with NULL device path input.	1. Call **LocateDevicePath()** to search for a handle with NULL device path input. The return code should be **EFI_INVALID_PARAMETER.**
5.1.3.7.5	0x2a8392aa, 0x7362, 0x4edd, 0xab, 0x52, 0x07, 0xe1, 0x7e, 0x84, 0x93, 0xf3	**BS.LocateDevicePath – LocateDevicePath()** returns **EFI_INVALID_PARAMETER** with NULL device and protocol is already installed on given device path.	1. Call **LocateDevicePath()** to search for a handle with NULL device and protocol is already installed on given device path. The return code should be **EFI_INVALID_PARAMETER.**

Number	GUID	Assertion	Test Description
5.1.3.7.6	0x7451c26a, 0x2e5b, 0x438d, 0x92, 0x96, 0x37, 0xe0, 0x52, 0x7e, 0xa5, 0x09	**BS.LocateDevicePath – LocateDevicePath()** returns **EFI_SUCCESS** with exist protocol at **EFI_TPL_APPLICATION**	1. Create 5 device pathses, and each device path is the parent of the follow one. 2. Install each device path and a test protocol onto a new handle. 3. Call **LocateDevicePath()** to locate each test protocol. The return code should be **EFI_SUCCESS**.
5.1.3.7.7	0xebdc8762, 0x84f7, 0x4e04, 0x8b, 0x95, 0x46, 0x33, 0x72, 0xc5, 0xc6, 0x16	**BS.LocateDevicePath – LocateDevicePath()** returns **EFI_SUCCESS** with exist protocol at **EFI_TPL_CALLBACK**	1. Create 5 device pathses, and each device path is the parent of the follow one. 2. Install each device path and a test protocol onto a new handle. 3. Call **LocateDevicePath()** to locate each test protocol. The return code should be **EFI_SUCCESS**.
5.1.3.7.8	0x6b886422, 0x1358, 0x4e40, 0x83, 0x4d, 0xe6, 0x04, 0x66, 0x3f, 0x4a, 0x6c	**BS.LocateDevicePath – LocateDevicePath()** returns **EFI_SUCCESS** with exist protocol at **EFI_TPL_NOTIFY**	1. Create 5 device pathses, and each device path is the parent of the follow one. 2. Install each device path and a test protocol onto a new handle. 3. Call **LocateDevicePath()** to locate each test protocol. The return code should be **EFI_SUCCESS**.
5.1.3.7.9	0x67c59d93, 0x28cd, 0x4b71, 0xa9, 0xf0, 0xbc, 0x21, 0xb4, 0x4e, 0xa1, 0xb3	**BS.LocateDevicePath – LocateDevicePath()** gets the remaining device path at **EFI_TPL_APPLICATION**	1. Create 5 device pathses, and each device path is the parent of the follow one. 2. Install each device path and a test protocol onto a new handle. 3. Call **LocateDevicePath()** to locate each test protocol. The return device path should be the remaining device path.

Number	GUID	Assertion	Test Description
5.1.3.7.10	0x8427cd13, 0x3f7c, 0x41d2, 0x88, 0x5e, 0xf6, 0x0f, 0x53, 0x01, 0xf1, 0xaf	**BS.LocateDevicePath – LocateDevicePath()** gets the remaining device path at **EFI_TPL_CALLBACK**	1. Create 5 device pathses, and each device path is the parent of the follow one. 2. Install each device path and a test protocol onto a new handle. 3. Call **LocateDevicePath()** to locate each test protocol. The return device path should be the remaining device path.
5.1.3.7.11	0xffe496ea, 0x9207, 0x4ff1, 0x83, 0x19, 0x1c, 0xde, 0x02, 0x5d, 0xda, 0x0c	**BS.LocateDevicePath – LocateDevicePath()** gets the remaining device path at **EFI_TPL_NOTIFY**	1. Create 5 device pathses, and each device path is the parent of the follow one. 2. Install each device path and a test protocol onto a new handle. 3. Call **LocateDevicePath()** to locate each test protocol. The return device path should be the remaining device path.
5.1.3.7.12	0xf7f49158, 0x91f5, 0x4357, 0xaa, 0x88, 0x6e, 0x76, 0x29, 0x10, 0x65, 0x23	**BS.LocateDevicePath – LocateDevicePath()** locates the protocol by device path at **EFI_TPL_APPLICATION**	1. Create 5 device pathses, and each device path is the parent of the follow one. 2. Install each device path and a test protocol onto a new handle. 3. Call **LocateDevicePath()** to locate each test protocol. The test protocol's function should be accessed and executed correctly.
5.1.3.7.13	0x3349f1a1, 0xb6df, 0x4fac, 0x81, 0xda, 0xe6, 0xa5, 0xb7, 0xb3, 0xf3, 0xa5	**BS.LocateDevicePath – LocateDevicePath()** locates the protocol by device path at **EFI_TPL_CALLBACK**	1. Create 5 device pathses, and each device path is the parent of the follow one. 2. Install each device path and a test protocol onto a new handle. 3. Call **LocateDevicePath()** to locate each test protocol. The test protocol's function should be accessed and executed correctly.

Number	GUID	Assertion	Test Description
5.1.3.7.14	0xa3dee53d, 0x11e3, 0x46ec, 0xbb, 0xc6, 0xd6, 0x5e, 0xb5, 0x05, 0xf1, 0xd4	**BS.LocateDevicePath – LocateDevicePath()** locates the protocol by device path at **EFI_TPL_NOTIFY**	1. Create 5 device paths, and each device path is the parent of the follow one. 2. Install each device path and a test protocol onto a new handle. 3. Call **LocateDevicePath()** to locate each test protocol. The test protocol's function should be accessed and executed correctly.

3.3.8 OpenProtocol()

Number	GUID	Assertion	Test Description
5.1.3.8.1	0xe04aea6f, 0xc5dd, 0x4d53, 0xbc, 0x7a, 0x94, 0xa3, 0xd8, 0x54, 0x2c, 0x4d	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with invalid handle	1. Call **OpenProtocol()** with invalid handle. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.8.2	0xd2fba07a, 0xff1f, 0x452e, 0x86, 0x51, 0x5e, 0x88, 0x44, 0x9d, 0xea, 0xc4	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with **NULL** protocol	1. Call **OpenProtocol()** with protocol GUID value of **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.8.3	0xb4e6dee7, 0x3038, 0x4ff8, 0x87, 0x69, 0xf4, 0x82, 0xe0, 0xc5, 0xd2, 0x0d	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with **NULL** interface when *Attributes* is not **TEST_PROTOCOL**	1. Call **OpenProtocol()** with **NULL** interface and *Attributes* does not equal **TEST_PROTOCOL**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.8.4	0x0e01e46a, 0x20eb, 0x45dd, 0x84, 0xc3, 0xf9, 0x3e, 0x99, 0x1e, 0xf4, 0x33	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with invalid attributes	1. Call **OpenProtocol()** with attributes other than **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, **BY_CHILD_CONTROLLER**, **BY_DRIVER**, **BY_DRIVER | EXCLUSIVE**, **EXCLUSIVE**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.8.5	0xdca26772, 0x48b7, 0x4921, 0xa9, 0xb7, 0x7b, 0xf5, 0xd9, 0x29, 0x5d, 0x27	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with attributes is **BY_CHILD_CONTROLLER** and invalid *AgentHandle*	1. Call **OpenProtocol()** with attributes is **BY_CHILD_CONTROLLER** and *AgentHandle* is an invalid **EFI_HANDLE**. The return code should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.1.3.8.6	0xc84dd52d, 0xb9eb, 0x42aa, 0x8c, 0x01, 0xea, 0x85, 0xa3, 0x08, 0xc0, 0x72	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with attributes is **BY_DRIVER** and invalid *AgentHandle*	1. Call **OpenProtocol()** with attributes is **BY_DRIVER** and *AgentHandle* is an invalid **EFI_HANDLE**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.8.7	0xe7a8eadd, 0x3874, 0x4f8e, 0xa1, 0x6b, 0x1e, 0xeb, 0x4d, 0x7c, 0xc8, 0xfa	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with attributes is **BY_DRIVER | EXCLUSIVE** and invalid *AgentHandle*	1. Call **OpenProtocol()** with attributes is **BY_DRIVER | EXCLUSIVE** and *AgentHandle* is an invalid **EFI_HANDLE**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.8.8	0x5abda0f9, 0x17a2, 0x40ce, 0x85, 0x62, 0x1a, 0xe7, 0x0a, 0xa1, 0x37, 0xd0	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with attributes is **EXCLUSIVE** and invalid *AgentHandle*	1. Call **OpenProtocol()** with attributes is **EXCLUSIVE** and *AgentHandle* is an invalid **EFI_HANDLE**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.8.9	0x822792bd, 0x0a83, 0x426f, 0x9d, 0x6a, 0xd3, 0x52, 0x8b, 0xf4, 0x67, 0x60	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with attributes is **BY_CHILD_CONTROLLER** and invalid *ControllerHandle*	1. Call **OpenProtocol()** with attributes is **BY_CHILD_CONTROLLER** and *ControllerHandle* is an invalid **EFI_HANDLE**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.8.10	0x17e1ac28, 0xfcd2, 0x4459, 0xb2, 0xee, 0x3c, 0xca, 0xc5, 0x74, 0xf6, 0x21	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with attributes is **BY_DRIVER** and invalid *ControllerHandle*	1. Call **OpenProtocol()** with attributes is **BY_DRIVER** and *ControllerHandle* is an invalid **EFI_HANDLE**. The return code should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.1.3.8.11	0x7a027e60, 0xd967, 0x4162, 0xb6, 0x99, 0xb9, 0x80, 0xe0, 0xfe, 0xf9, 0xcf	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with attributes is **BY_DRIVER | EXCLUSIVE** and invalid *ControllerHandle*	1. Call **OpenProtocol()** with attributes is **BY_DRIVER | EXCLUSIVE** and *ControllerHandle* is an invalid **EFI_HANDLE**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.8.12	0x357d40b9, 0xa9b0, 0x4462, 0xa4, 0xc7, 0x40, 0xca, 0x18, 0xcb, 0x17, 0x34	**BS.OpenProtocol – OpenProtocol()** returns **EFI_INVALID_PARAMETER** with attributes is **BY_CHILD_CONTROLLER** and handle is identical to the *ControllerHandle* .	1. Call **OpenProtocol()** with attributes is **BY_CHILD_CONTROLLER** and *Handle* is identical to *ControllerHandle*. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.8.13	0x4f733e46, 0xdacb, 0x4f6f, 0x80, 0x2b, 0x05, 0x45, 0x00, 0x3a, 0x6a, 0x64	**BS.OpenProtocol – OpenProtocol()** returns **EFI_UNSUPPORTED** with never installed protocol	1. Call **OpenProtocol()** to attempt to open a never installed protocol on the handle. The return code should be **EFI_UNSUPPORTED**.
5.1.3.8.14	0xf8b8c1a0, 0xda67, 0x48b6, 0x9c, 0xee, 0xd7, 0xbc, 0x81, 0xc5, 0x3b, 0x74	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **BY_HANDLE_PROTOCOL** at **EFI_TPL_APPLICATION.**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** when *Attributes* is always **BY_HANDLE_PROTOCOL**. The return code should be **EFI_SUCCESS**.
5.1.3.8.15	0xe24ad52e, 0x6596, 0x4bad, 0x80, 0xdb, 0x05, 0x3b, 0x5b, 0x26, 0x5d, 0xa7	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **BY_HANDLE_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** when *Attributes* is **BY_HANDLE_PROTOCOL**. The return code should be **EFI_SUCCESS**.
5.1.3.8.16	0x28471b73, 0x3543, 0x4021, 0xa8, 0xe6, 0x66, 0x09, 0x04, 0x0a, 0xce, 0xd9	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **BY_HANDLE_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** when *Attributes* is **BY_HANDLE_PROTOCOL**. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.8.17	0x4cd217f8, 0x439e, 0x4c94, 0xa0, 0xad, 0x2a, 0x84, 0x1a, 0xb8, 0x14, 0xdc	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with GET_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is GET_PROTOCOL. The return code should be EFI_SUCCESS.
5.1.3.8.18	0x04f77931, 0x6264, 0x4c07, 0xb2, 0xb7, 0x75, 0x8b, 0x88, 0xb0, 0xd1, 0xd9	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with GET_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is GET_PROTOCOL. The return code should be EFI_SUCCESS.
5.1.3.8.19	0x26405688, 0x8ade, 0x4501, 0xb1, 0xc9, 0x35, 0x9b, 0x27, 0xc4, 0x2d, 0x48	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with GET_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is GET_PROTOCOL. The return code should be EFI_SUCCESS.
5.1.3.8.20	0xc68c7ab9, 0x4f2b, 0x402b, 0xb4, 0x35, 0x4c, 0xa1, 0x58, 0x7d, 0x77, 0xd4	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with TEST_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is TEST_PROTOCOL. The return code should be EFI_SUCCESS.
5.1.3.8.21	0x729bf68d, 0x281a, 0x41fe, 0x80, 0xc9, 0xfc, 0x2a, 0x80, 0x50, 0x5b, 0xc0	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with TEST_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is TEST_PROTOCOL. The return code should be EFI_SUCCESS.
5.1.3.8.22	0x659ddd65, 0x0c44, 0x4bbb, 0xad, 0xc8, 0x77, 0x71, 0x48, 0x3f, 0x47, 0xe8	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with TEST_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is TEST_PROTOCOL. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.8.23	0xdc16b745, 0x528b, 0x4552, 0x80, 0x20, 0xed, 0xaf, 0x54, 0x43, 0xf5, 0x6d	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **BY_CHILD_CONTROLLER** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** when *Attributes* is **BY_CHILD_CONTROLLER**. The return code should be **EFI_SUCCESS**.
5.1.3.8.24	0x4c93f05c, 0x3d94, 0x4f92, 0xae, 0x9b, 0x28, 0x0b, 0xe0, 0x09, 0x32, 0xb2	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **BY_CHILD_CONTROLLER** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** when *Attributes* is **BY_CHILD_CONTROLLER**. The return code should be **EFI_SUCCESS**.
5.1.3.8.25	0xd9871fff, 0xc2aa, 0x445a, 0x9a, 0xd7, 0x92, 0xa8, 0xea, 0x57, 0x14, 0x92	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **BY_CHILD_CONTROLLER** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** when *Attributes* is **BY_CHILD_CONTROLLER**. The return code should be **EFI_SUCCESS**.
5.1.3.8.26	0xb8228793, 0x2c72, 0x4583, 0x8f, 0xa4, 0x7b, 0x09, 0xd1, 0x38, 0x0a, 0xe5	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** when *Attributes* is **EXCLUSIVE**. The return code should be **EFI_SUCCESS**.
5.1.3.8.27	0xc2d6fe86, 0xbc2f, 0x4086, 0xb9, 0x05, 0xe6, 0x14, 0xa8, 0xf6, 0x9b, 0xe7	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** when *Attributes* is **EXCLUSIVE**. The return code should be **EFI_SUCCESS**.
5.1.3.8.28	0x7e1aa146, 0x38bb, 0x421f, 0xb7, 0x4b, 0x2e, 0x1a, 0x8a, 0xbe, 0xdf, 0xc4	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** when *Attributes* is **EXCLUSIVE**. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.8.29	0x80e045bd, 0x884d, 0x4bc5, 0x97, 0x57, 0x83, 0x79, 0xc6, 0xd3, 0xf4, 0x51	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_DRIVER at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is BY_DRIVER. The return code should be EFI_SUCCESS.
5.1.3.8.30	0x5395226d, 0x3efb, 0x48be, 0xa8, 0x1d, 0x42, 0x6f, 0x5b, 0xac, 0x5a, 0x81	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_DRIVER at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is BY_DRIVER. The return code should be EFI_SUCCESS.
5.1.3.8.31	0x39b175d6, 0x6609, 0x4ae5, 0x85, 0x9f, 0x89, 0x73, 0x03, 0x87, 0xf2, 0xa1	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_DRIVER at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is BY_DRIVER. The return code should be EFI_SUCCESS.
5.1.3.8.32	0xa344c400, 0x679a, 0x42e3, 0x8b, 0xdc, 0xcc, 0xf6, 0xda, 0x10, 0xdd, 0xad	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_DRIVER | EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is BY_DRIVER | EXCLUSIVE. The return code should be EFI_SUCCESS.
5.1.3.8.33	0x501ff789, 0x3380, 0x415f, 0xab, 0x29, 0xf1, 0x1c, 0xa3, 0x61, 0x70, 0x63	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_DRIVER | EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is BY_DRIVER | EXCLUSIVE. The return code should be EFI_SUCCESS.
5.1.3.8.34	0xce6b58c7, 0xd505, 0x489d, 0xb9, 0xfe, 0x11, 0xdd, 0x25, 0x4c, 0xef, 0x69	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_DRIVER | EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() when *Attributes* is BY_DRIVER | EXCLUSIVE. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.8.35	0x8ba08878, 0xc464, 0x4749, 0xaf, 0x64, 0xbb, 0xe1, 0x20, 0xa6, 0x28, 0x24	**BS.OpenProtocol – OpenProtocol()** returns **EFI_ALREADY_STARTED** to open a opened **BY_DRIVER** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** ~ 3 onto **TestHandle1**. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER**, **TestProtocol2 EXCLUSIVE**, **TestProtocol3 BY_DRIVER | EXCLUSIVE**. 3. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER** again. The return code should be **EFI_ALREADY_STARTED**.
5.1.3.8.36	0xa5abd4d4, 0xeba4, 0x448d, 0x9c, 0xca, 0x99, 0xd7, 0xca, 0x39, 0x9a, 0x1d	**BS.OpenProtocol – OpenProtocol()** returns **EFI_ALREADY_STARTED** to open a opened **BY_DRIVER** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** ~ 3 onto **TestHandle1**. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER**, **TestProtocol2 EXCLUSIVE**, **TestProtocol3 BY_DRIVER | EXCLUSIVE**. 3. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER** again. The return code should be **EFI_ALREADY_STARTED**.
5.1.3.8.37	0x81c7eb16, 0x6075, 0x4e85, 0xa0, 0xa5, 0x49, 0x7b, 0xc0, 0x0c, 0x24, 0x32	**BS.OpenProtocol – OpenProtocol()** returns **EFI_ALREADY_STARTED** to open a opened **BY_DRIVER** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** ~ 3 onto **TestHandle1**. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER**, **TestProtocol2 EXCLUSIVE**, **TestProtocol3 BY_DRIVER | EXCLUSIVE**. 3. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER** again. The return code should be **EFI_ALREADY_STARTED**.

Number	GUID	Assertion	Test Description
5.1.3.8.38	0x51987dd1, 0xc45a, 0x4389, 0x9a, 0x0d, 0xdc, 0xf2, 0xc6, 0x5c, 0xba, 0x98	**BS.OpenProtocol – OpenProtocol()** returns **EFI_ACCESS_DENIED** to open a opened **EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** ~ 3 onto **TestHandle1**. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER**, **TestProtocol2 EXCLUSIVE**, **TestProtocol3 BY_DRIVER | EXCLUSIVE**. 3. Call **OpenProtocol()** to open **TestProtocol2 BY_DRIVER** again. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.8.39	0xfceb340e, 0xd583, 0x4b26, 0x8d, 0x1c, 0x2b, 0x0f, 0x22, 0x67, 0xbb, 0xeb	**BS.OpenProtocol – OpenProtocol()** returns **EFI_ACCESS_DENIED** to open a opened **EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** ~ 3 onto **TestHandle1**. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER**, **TestProtocol2 EXCLUSIVE**, **TestProtocol3 BY_DRIVER | EXCLUSIVE**. 3. Call **OpenProtocol()** to open **TestProtocol2 BY_DRIVER** again. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.8.40	0x1e5a90f9, 0x5fec, 0x4a83, 0xb8, 0xf2, 0x41, 0xa5, 0x8f, 0x1c, 0xba, 0x5e	**BS.OpenProtocol – OpenProtocol()** returns **EFI_ACCESS_DENIED** to open a opened **EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** ~ 3 onto **TestHandle1**. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER**, **TestProtocol2 EXCLUSIVE**, **TestProtocol3 BY_DRIVER | EXCLUSIVE**. 3. Call **OpenProtocol()** to open **TestProtocol2 BY_DRIVER** again. The return code should be **EFI_ACCESS_DENIED**.

Number	GUID	Assertion	Test Description
5.1.3.8.41	0x310ad89c, 0x192d, 0x4714, 0xa2, 0x41, 0x00, 0x79, 0xfb, 0x8b, 0x3d, 0xf3	**BS.OpenProtocol – OpenProtocol()** returns **EFI_ACCESS_DENIED** to open a opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** ~ 3 onto **TestHandle1**. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER**, **TestProtocol2 EXCLUSIVE**, **TestProtocol3 BY_DRIVER | EXCLUSIVE**. 3. Call **OpenProtocol()** to open **TestProtocol3 BY_DRIVER** again. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.8.42	0x66c275cb, 0x39dd, 0x409e, 0xaa, 0xc9, 0x5c, 0x1e, 0xdd, 0xec, 0x3f, 0x34	**BS.OpenProtocol – OpenProtocol()** returns **EFI_ACCESS_DENIED** to open a opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** ~ 3 onto **TestHandle1**. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER**, **TestProtocol2 EXCLUSIVE**, **TestProtocol3 BY_DRIVER | EXCLUSIVE**. 3. Call **OpenProtocol()** to open **TestProtocol3 BY_DRIVER** again. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.8.43	0xe2e53aa7, 0xe3f5, 0x4afc, 0xbb, 0x70, 0x8a, 0x8a, 0xe0, 0x39, 0xc1, 0xc2	**BS.OpenProtocol – OpenProtocol()** returns **EFI_ACCESS_DENIED** to open a opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** ~ 3 onto **TestHandle1**. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER**, **TestProtocol2 EXCLUSIVE**, **TestProtocol3 BY_DRIVER | EXCLUSIVE**. 3. Call **OpenProtocol()** to open **TestProtocol3 BY_DRIVER** again. The return code should be **EFI_ACCESS_DENIED**.

Number	GUID	Assertion	Test Description
5.1.3.8.44	0x6a0534df, 0xf826, 0x46de, 0x9a, 0x0b, 0x2a, 0x58, 0xcc, 0x95, 0x17, 0xc3	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened BY_DRIVER at EFI_TPL_APPLICATION	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.45	0xb2545ee7, 0x63a3, 0x440f, 0x91, 0x28, 0x19, 0xbe, 0xc1, 0xaf, 0x73, 0x52	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened BY_DRIVER at EFI_TPL_CALLBACK	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.46	0xff316241, 0x8d83, 0x4e13, 0x9e, 0x6d, 0x9e, 0x7b, 0xb8, 0x59, 0x79, 0xbf	BS.OpenProtocol – OpenProtocol() returns EFI_ALREADY_STARTED to open a opened BY_DRIVER at EFI_TPL_NOTIFY	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE again. The return code should be EFI_ALREADY_STARTED.

Number	GUID	Assertion	Test Description
5.1.3.8.47	0x8be67955, 0x31b8, 0x4c1f, 0x99, 0xfe, 0x59, 0x9a, 0x9c, 0xe6, 0xf8, 0xb7	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol2 EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.48	0x84c30135, 0x86fa, 0x43c2, 0xba, 0x33, 0xbe, 0x3a, 0x0d, 0x4f, 0x3b, 0x5a	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol2 EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.49	0x75e90310, 0x22bd, 0x41d3, 0xb4, 0xee, 0x10, 0x28, 0x4d, 0x8f, 0x58, 0xca	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol2 EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.50	0x0e495234, 0x478c, 0x4668, 0x9a, 0x48, 0xd1, 0x8d, 0x76, 0xd3, 0x9f, 0x39	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened BY_DRIVER | EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol3 EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.51	0x5c288d57, 0x93e4, 0x4111, 0x88, 0x5f, 0x88, 0x1f, 0xe9, 0x5d, 0x89, 0x3a	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened BY_DRIVER | EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol3 EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.52	0xd9f3625f, 0x9f31, 0x420d, 0xa8, 0xe8, 0x60, 0xd4, 0x29, 0x09, 0xd2, 0x2f	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened BY_DRIVER | EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol3 EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.53	0x105b44cb, 0x04ad, 0x456d, 0x92, 0x16, 0x77, 0xb6, 0x3e, 0x01, 0x10, 0x50	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened BY_DRIVER at EFI_TPL_APPLICATION	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.54	0xf23c3c33, 0xcbb4, 0x48fc, 0x8c, 0x35, 0xc1, 0x67, 0xb6, 0x93, 0x08, 0x3d	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened BY_DRIVER at EFI_TPL_CALLBACK	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.55	0x7d5271f9, 0xddb0, 0x47d3, 0xa5, 0xb7, 0x27, 0xe1, 0x12, 0xf6, 0xc1, 0xdd	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened BY_DRIVER at EFI_TPL_NOTIFY	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.56	0xc0f8ce0b, 0x77f2, 0x4c39, 0x82, 0x02, 0xaa, 0x58, 0xe2, 0x68, 0xab, 0x9e	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol2 BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.57	0xc2043b13, 0x3827, 0x42b7, 0xb6, 0xa4, 0x67, 0x5b, 0xbb, 0x2e, 0x9e, 0x04	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol2 BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.58	0x4d0b2d09, 0xa55a, 0x41b6, 0x8e, 0x0d, 0x38, 0x3b, 0x2a, 0x69, 0x1c, 0xa4	BS.OpenProtocol – OpenProtocol() returns EFI_ACCESS_DENIED to open a opened EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol2 BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.59	0x5768e02b, 0x605c, 0x4d1c, 0xb9, 0xf3, 0x7e, 0xaf, 0x73, 0xd1, 0x2f, 0x38	BS.OpenProtocol – OpenProtocol() returns EFI_ALREADY_STARTED to open a opened BY_DRIVER | EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol3 BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ALREADY_STARTED.
5.1.3.8.60	0xea96e021, 0xd431, 0x44b8, 0x95, 0xd1, 0xb7, 0xd0, 0x66, 0x12, 0xaa, 0x8d	BS.OpenProtocol – OpenProtocol() returns EFI_ALREADY_STARTED to open a opened BY_DRIVER | EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol3 BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ALREADY_STARTED.
5.1.3.8.61	0xe4f5fba0, 0xaef9, 0x4ff7, 0xa8, 0xbd, 0x6b, 0x0d, 0xb5, 0x7c, 0x52, 0xfb	BS.OpenProtocol – OpenProtocol() returns EFI_ALREADY_STARTED to open a opened BY_DRIVER | EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 ~ 3 onto TestHandle1. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER, TestProtocol2 EXCLUSIVE, TestProtocol3 BY_DRIVER | EXCLUSIVE. 3. Call OpenProtocol() to open TestProtocol3 BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ALREADY_STARTED.

Number	GUID	Assertion	Test Description
5.1.3.8.62	0x2aa15ebf, 0x0886, 0x45ec, 0x90, 0x8f, 0xa6, 0x85, 0x35, 0x47, 0xc2, 0x7e	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_HANDLE_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_HANDLE_PROTOCOL. The return code should be EFI_SUCCESS.
5.1.3.8.63	0xbf0c2a4b, 0x3666, 0x4521, 0x96, 0xda, 0xb1, 0x10, 0x8b, 0x5d, 0x13, 0x34	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_HANDLE_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_HANDLE_PROTOCOL. The return code should be EFI_SUCCESS.
5.1.3.8.64	0xaad371a4, 0x9cdd, 0x4821, 0xb5, 0xb0, 0x1e, 0xf5, 0x62, 0x76, 0x71, 0xbe	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_HANDLE_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_HANDLE_PROTOCOL. The return code should be EFI_SUCCESS.
5.1.3.8.65	0xbf1d8fa1, 0x16d4, 0x4812, 0x99, 0x10, 0x12, 0x7a, 0x3c, 0xf4, 0x57, 0x1a	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_HANDLE_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_HANDLE_PROTOCOL in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.66	0x7846f5d2, 0xd936, 0x486b, 0x9a, 0x94, 0x87, 0xce, 0x23, 0xc3, 0x30, 0x19	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_HANDLE_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_HANDLE_PROTOCOL in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.67	0xde91d40a, 0xe684, 0x4eb1, 0x9b, 0xf4, 0x40, 0x8c, 0x04, 0x53, 0x8f, 0xcc	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_HANDLE_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_HANDLE_PROTOCOL in an external driver that does not follow EFI driver model. The driver should be loaded.

Number	GUID	Assertion	Test Description
5.1.3.8.68	0xc9460f7e, 0x2ac7, 0x4fef, 0x90, 0x50, 0xb4, 0x84, 0x36, 0x47, 0xae, 0x70	`BS.OpenProtocol – OpenProtocol()` returns `EFI_SUCCESS` with `BY_HANDLE_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol2` onto a new handle. 2. Call `OpenProtocol()` with `BY_HANDLE_PROTOCOL` in an external driver that does not follow EFI driver model. The return code should be `EFI_SUCCESS`.
5.1.3.8.69	0x16d40f9b, 0x97dc, 0x4fa9, 0xbb, 0x0b, 0x02, 0xf5, 0xae, 0x72, 0x13, 0xa7	`BS.OpenProtocol – OpenProtocol()` returns `EFI_SUCCESS` with `BY_HANDLE_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol2` onto a new handle. 2. Call `OpenProtocol()` with `BY_HANDLE_PROTOCOL` in an external driver that does not follow EFI driver model. The return code should be `EFI_SUCCESS`.
5.1.3.8.70	0x45c7ab50, 0xb7d2, 0x498f, 0xaa, 0xba, 0x6e, 0xbe, 0xb4, 0xee, 0x6b, 0x7d	`BS.OpenProtocol – OpenProtocol()` returns `EFI_SUCCESS` with `BY_HANDLE_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol2` onto a new handle. 2. Call `OpenProtocol()` with `BY_HANDLE_PROTOCOL` in an external driver that does not follow EFI driver model. The return code should be `EFI_SUCCESS`.
5.1.3.8.71	0x3aa76227, 0xfaf6, 0x4ca2, 0x99, 0x9a, 0xf2, 0x9a, 0x4b, 0x86, 0xb6, 0x6f	`BS.OpenProtocol – OpenProtocol()` returns `EFI_SUCCESS` with `BY_HANDLE_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` with `BY_HANDLE_PROTOCOL` in an external driver that does not follow EFI driver model. 3. Call `OpenProtocol()` with `BY_HANDLE_PROTOCOL` again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.8.72	0x346eeba8, 0xae42, 0x4b9d, 0xae, 0x3b, 0xca, 0xd6, 0x39, 0x88, 0xb8, 0xcb	`BS.OpenProtocol – OpenProtocol()` returns `EFI_SUCCESS` with `BY_HANDLE_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` with `BY_HANDLE_PROTOCOL` in an external driver that does not follow EFI driver model. 3. Call `OpenProtocol()` with `BY_HANDLE_PROTOCOL` again. The return code should be `EFI_SUCCESS`.
5.1.3.8.73	0x1954dbdd, 0xb7f2, 0x485d, 0xb5, 0x22, 0x04, 0xd6, 0x4c, 0x05, 0x8c, 0x5e	`BS.OpenProtocol – OpenProtocol()` returns `EFI_SUCCESS` with `BY_HANDLE_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a new handle. 2. Call `OpenProtocol()` with `BY_HANDLE_PROTOCOL` in an external driver that does not follow EFI driver model. 3. Call `OpenProtocol()` with `BY_HANDLE_PROTOCOL` again. The return code should be `EFI_SUCCESS`.
5.1.3.8.74	0x38a71272, 0x8ffb, 0x4fe2, 0xba, 0x27, 0x85, 0x77, 0xfd, 0xf3, 0x25, 0x98	`BS.OpenProtocol – OpenProtocol()` returns `EFI_SUCCESS` with `GET_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol2` onto a new handle. 2. Call `OpenProtocol()` with `GET_PROTOCOL`. The return code should be `EFI_SUCCESS`.
5.1.3.8.75	0xb312c5ab, 0xe33a, 0x441c, 0xa2, 0x82, 0xf2, 0xe1, 0xed, 0x5d, 0x8d, 0x25	`BS.OpenProtocol – OpenProtocol()` returns `EFI_SUCCESS` with `GET_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol2` onto a new handle. 2. Call `OpenProtocol()` with `GET_PROTOCOL`. The return code should be `EFI_SUCCESS`.
5.1.3.8.76	0x9dfc7f23, 0x27d6, 0x40b9, 0x8f, 0x5e, 0x42, 0x74, 0x7d, 0x8d, 0x8c, 0x48	`BS.OpenProtocol – OpenProtocol()` returns `EFI_SUCCESS` with `GET_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol2` onto a new handle. 2. Call `OpenProtocol()` with `GET_PROTOCOL`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.8.77	0x1ee34b41, 0x814f, 0x44ae, 0xb3, 0xcb, 0xe0, 0xf2, 0x65, 0x84, 0x5d, 0x9e	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with GET_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with GET_PROTOCOL in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.78	0x09afde5f, 0x30e3, 0x4197, 0x95, 0xa9, 0x01, 0xf3, 0xe9, 0xb2, 0x3f, 0xd8	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with GET_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with GET_PROTOCOL in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.79	0x6f3e8ae0, 0x822d, 0x4d41, 0x8a, 0x38, 0x40, 0xb1, 0x9d, 0xb4, 0x4f, 0x89	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with GET_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with GET_PROTOCOL in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.80	0xc7a93fd6, 0xb21d, 0x4323, 0xad, 0xd8, 0x3e, 0xbe, 0x0b, 0x9d, 0xf1, 0x7b	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with GET_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with GET_PROTOCOL in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.81	0xc7189505, 0x78a7, 0x4a5f, 0x98, 0xdb, 0xbd, 0x28, 0x71, 0x3a, 0x31, 0x94	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with GET_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with GET_PROTOCOL in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.82	0xe3fe868d, 0xf1d5, 0x437f, 0x99, 0xef, 0xd5, 0x53, 0x93, 0x8f, 0x5c, 0x64	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with GET_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with GET_PROTOCOL in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.8.83	0xd04e388d, 0x1000, 0x4743, 0x89, 0xca, 0x58, 0x21, 0xfa, 0x17, 0x4f, 0xec	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **GET_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **GET_PROTOCOL** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **GET_PROTOCOL** again. The return code should be **EFI_SUCCESS**.
5.1.3.8.84	0x6bffc888, 0xe5c5, 0x4a4f, 0xb2, 0xfe, 0x81, 0x98, 0xa8, 0x46, 0x0d, 0xc7	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **GET_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **GET_PROTOCOL** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **GET_PROTOCOL** again. The return code should be **EFI_SUCCESS**.
5.1.3.8.85	0xc9061c77, 0x922e, 0x497f, 0xbc, 0xad, 0xad, 0x04, 0x63, 0x15, 0x13, 0x52	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **GET_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **GET_PROTOCOL** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **GET_PROTOCOL** again. The return code should be **EFI_SUCCESS**.
5.1.3.8.86	0xcd2818e3, 0x5c5c, 0x4270, 0xad, 0xed, 0xa4, 0x93, 0x9f, 0x91, 0x1c, 0xa5	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **TEST_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol2** onto a new handle. 2. Call **OpenProtocol()** with **TEST_PROTOCOL**. The return code should be **EFI_SUCCESS**.
5.1.3.8.87	0xa5ab5a70, 0x518f, 0x4d2d, 0x98, 0xac, 0x0f, 0x92, 0x1e, 0x66, 0xd6, 0x17	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **TEST_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol2** onto a new handle. 2. Call **OpenProtocol()** with **TEST_PROTOCOL**. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.8.88	0x4a59b76b, 0x6425, 0x45b1, 0xbb, 0x23, 0xd2, 0x32, 0x61, 0xf6, 0xe5, 0x68	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **TEST_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol2** onto a new handle. 2. Call **OpenProtocol()** with **TEST_PROTOCOL**. The return code should be **EFI_SUCCESS**.
5.1.3.8.89	0x5f97e881, 0x959b, 0x4c3c, 0x8c, 0x25, 0xe1, 0xb6, 0xb8, 0x48, 0xe2, 0xb2	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **TEST_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol2** onto a new handle. 2. Call **OpenProtocol()** with **TEST_PROTOCOL** in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.90	0x9102ee74, 0x5fa7, 0x4436, 0xae, 0x51, 0xed, 0x15, 0x41, 0x35, 0xfe, 0xc1	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **TEST_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol2** onto a new handle. 2. Call **OpenProtocol()** with **TEST_PROTOCOL** in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.91	0x17f92140, 0x11b9, 0x4c02, 0xb6, 0x49, 0x99, 0x73, 0xb8, 0xf1, 0x8f, 0x6f	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **TEST_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol2** onto a new handle. 2. Call **OpenProtocol()** with **TEST_PROTOCOL** in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.92	0x951ac798, 0x99a7, 0x4174, 0x90, 0x0d, 0x2b, 0xf9, 0x22, 0x66, 0x5e, 0xd5	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **TEST_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol2** onto a new handle. 2. Call **OpenProtocol()** with **TEST_PROTOCOL** in an external driver that does not follow EFI driver model. The return code should be **EFI_SUCCESS**.
5.1.3.8.93	0x3d285a5e, 0xab3f, 0x47f8, 0xba, 0xbb, 0x32, 0x4b, 0x11, 0xc4, 0xef, 0x86	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **TEST_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol2** onto a new handle. 2. Call **OpenProtocol()** with **TEST_PROTOCOL** in an external driver that does not follow EFI driver model. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.8.94	0x65242c76, 0x9c09, 0x4091, 0x8e, 0xa4, 0x40, 0x80, 0xd5, 0x20, 0xc1, 0x66	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with TEST_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with TEST_PROTOCOL in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.95	0xcc1c71d3, 0xf645, 0x4c25, 0xac, 0xe4, 0x1c, 0x3f, 0x8b, 0x81, 0x12, 0x48	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with TEST_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with TEST_PROTOCOL in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with TEST_PROTOCOL again. The return code should be EFI_SUCCESS.
5.1.3.8.96	0xa6e784ed, 0x5aeb, 0x4646, 0xb6, 0xaa, 0x4e, 0x03, 0x5f, 0x03, 0xb1, 0x6b	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with TEST_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with TEST_PROTOCOL in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with TEST_PROTOCOL again. The return code should be EFI_SUCCESS.
5.1.3.8.97	0x075bcbc0, 0xcb18, 0x4965, 0xa1, 0xed, 0x52, 0x25, 0xff, 0xc8, 0x2f, 0x75	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with TEST_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with TEST_PROTOCOL in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with TEST_PROTOCOL again. The return code should be EFI_SUCCESS.
5.1.3.8.98	0x883dc6fa, 0xc66e, 0x4cf8, 0x82, 0x7f, 0xbe, 0x0c, 0xd5, 0xf4, 0x16, 0x78	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_CHILD_CONTROLLER at EFI_TPL_APPLICATION	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_CHILD_CONTROLLER. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.8.99	0x725c6b49, 0x2163, 0x439b, 0x8d, 0x8f, 0x75, 0x39, 0x33, 0x3a, 0x5a, 0xf3	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_CHILD_CONTROLLER at EFI_TPL_CALLBACK	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_CHILD_CONTROLLER. The return code should be EFI_SUCCESS.
5.1.3.8.100	0x0d82fdf1, 0x76b6, 0x455f, 0x8d, 0x25, 0x04, 0x28, 0xbc, 0xeb, 0x36, 0x6d	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_CHILD_CONTROLLER at EFI_TPL_NOTIFY	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_CHILD_CONTROLLER. The return code should be EFI_SUCCESS.
5.1.3.8.101	0x6d9e17e8, 0xab38, 0x4461, 0xa4, 0x27, 0x16, 0x5a, 0x83, 0xd7, 0x9f, 0x1f	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_CHILD_CONTROLLER at EFI_TPL_APPLICATION	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_CHILD_CONTROLLER in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.102	0x526ab525, 0x436c, 0x4b77, 0xa1, 0xce, 0x61, 0x70, 0x7f, 0x3d, 0xfb, 0xe5	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_CHILD_CONTROLLER at EFI_TPL_CALLBACK	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_CHILD_CONTROLLER in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.103	0x20527b37, 0x3ae6, 0x4320, 0x80, 0x9f, 0xa5, 0x90, 0x2c, 0x4e, 0xcf, 0x31	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_CHILD_CONTROLLER at EFI_TPL_NOTIFY	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_CHILD_CONTROLLER in an external driver that does not follow EFI driver model. The driver should be loaded.

Number	GUID	Assertion	Test Description
5.1.3.8.104	0xc12e22ab, 0xc537, 0x4067, 0x88, 0x10, 0xf3, 0xd2, 0x3e, 0x59, 0x24, 0x27	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_CHILD_CONTROLLER at EFI_TPL_APPLICATION	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_CHILD_CONTROLLER in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.105	0x48c75a14, 0x089e, 0x4313, 0x92, 0x54, 0xe0, 0xfe, 0x3a, 0x51, 0x7d, 0x2f	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_CHILD_CONTROLLER at EFI_TPL_CALLBACK	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_CHILD_CONTROLLER in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.106	0x40072ef9, 0x09c6, 0x4101, 0x99, 0x0d, 0x98, 0x5d, 0x8c, 0x0c, 0x9d, 0x9e	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_CHILD_CONTROLLER at EFI_TPL_NOTIFY	1. Install TestProtocol2 onto a new handle. 2. Call OpenProtocol() with BY_CHILD_CONTROLLER in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.107	0xd4402d18, 0x26c7, 0x4591, 0x82, 0xef, 0xe1, 0x00, 0x33, 0x12, 0xca, 0x20	BS.OpenProtocol – OpenProtocol() returns EFI_SUCCESS with BY_CHILD_CONTROLLER at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_CHILD_CONTROLLER in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_CHILD_CONTROLLER again. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.8.108	0x8d3e71ad, 0x0445, 0x4c2e, 0x9b, 0x5b, 0x52, 0x4d, 0xa5, 0xc0, 0x30, 0xdb	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **BY_CHILD_CONTROLLER** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_CHILD_CONTROLLER** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **BY_CHILD_CONTROLLER** again. The return code should be **EFI_SUCCESS**.
5.1.3.8.109	0x96561ad6, 0xb016, 0x42c8, 0xa5, 0xa7, 0xd1, 0xfa, 0xab, 0xd3, 0x5a, 0xa5	**BS.OpenProtocol – OpenProtocol()** returns **EFI_SUCCESS** with **BY_CHILD_CONTROLLER** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_CHILD_CONTROLLER** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **BY_CHILD_CONTROLLER** again. The return code should be **EFI_SUCCESS**.
5.1.3.8.110	0xf556fa41, 0x50f8, 0x4a0e, 0xa6, 0x43, 0xaa, 0x52, 0xb6, 0xb0, 0x16, 0x84	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER** in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.111	0xb7888f69, 0x56d8, 0x41ba, 0xbe, 0x94, 0xb1, 0x63, 0x21, 0xee, 0xc2, 0xb3	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER** in an external driver that does not follow EFI driver model. The driver should be loaded.

Number	GUID	Assertion	Test Description
5.1.3.8.112	0x694a60ff, 0x47d6, 0x4ab6, 0xa9, 0x3f, 0xa9, 0x1c, 0xbd, 0x4f, 0x71, 0x66	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.113	0xd913fa73, 0xf80a, 0x42aa, 0xab, 0x8a, 0xdc, 0x77, 0x1f, 0x80, 0xc0, 0x2f	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.114	0x55dd57f8, 0x31dc, 0x4e45, 0xa9, 0x3e, 0xb0, 0x8f, 0x8e, 0x0b, 0x73, 0xb9	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.115	0xb5e4107e, 0x3e8d, 0x41f3, 0xb2, 0xb2, 0xaf, 0xe4, 0x7b, 0xf8, 0x75, 0x68	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.116	0x0f4ba7fc, 0x8703, 0x4ea7, 0x92, 0xf0, 0x34, 0x0b, 0x72, 0xdd, 0x2b, 0x0f	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER again. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.117	0x748471d3, 0x378d, 0x4b4d, 0xac, 0xdb, 0x74, 0x3c, 0x1e, 0x80, 0x50, 0x09	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.118	0x9273a164, 0x52fe, 0x4348, 0x8d, 0xa2, 0x07, 0x13, 0xe8, 0x42, 0x36, 0xe9	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.119	0xccd352ac, 0x3315, 0x46c9, 0xb4, 0x87, 0xa9, 0x58, 0x0d, 0x15, 0x08, 0x31	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with EXCLUSIVE in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.120	0x151b69c4, 0xeebf, 0x4894, 0xbc, 0xb5, 0x0b, 0x01, 0x06, 0x63, 0x3d, 0x3b	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with EXCLUSIVE in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER again. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.121	0xe4281708, 0x4861, 0x4747, 0x97, 0x85, 0xfb, 0xec, 0xee, 0x2d, 0x48, 0x3e	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **EXCLUSIVE** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **BY_DRIVER** again. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.8.122	0xe76c2423, 0xd198, 0x4ee6, 0xa2, 0x9a, 0xea, 0x0c, 0xb4, 0xd4, 0x0d, 0x36	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **BY_DRIVER** again. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.8.123	0x73b010d1, 0x45f8, 0x4411, 0xae, 0xda, 0x06, 0x51, 0xe2, 0x08, 0x93, 0xf3	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **BY_DRIVER** again. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.8.124	0xc33e3bcb, 0x4671, 0x4bfc, 0x90, 0x9f, 0x5e, 0x1f, 0x0f, 0x6c, 0x43, 0x87	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **BY_DRIVER** again. The return code should be **EFI_ACCESS_DENIED**.

Number	GUID	Assertion	Test Description
5.1.3.8.125	0x480fa1d4, 0x05ee, 0x428f, 0xa0, 0xc9, 0xeb, 0x8c, 0xd8, 0x2c, 0x29, 0x58	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.126	0x9e46a76b, 0x6fbd, 0x479f, 0xb1, 0x5f, 0x5f, 0x3b, 0xa8, 0xf4, 0x61, 0x81	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.127	0xcc798b43, 0x4cd0, 0x44f6, 0x87, 0xa0, 0x1c, 0x31, 0x53, 0x7b, 0xd8, 0x1d	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.128	0xd27fef41, 0x2fce, 0x4859, 0xa4, 0x45, 0x07, 0x51, 0x30, 0xfb, 0x5a, 0x10	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.129	0xfbff7c54, 0x92cb, 0x477c, 0x80, 0xa6, 0x8f, 0x65, 0x04, 0x3f, 0x74, 0x6b	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.130	0x1079d678, 0x1e99, 0x4773, 0xb7, 0x84, 0xfd, 0xd3, 0xec, 0xb1, 0x17, 0x40	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.8.131	0x317cbcd4, 0x25ab, 0x4b66, 0x9b, 0x8c, 0xba, 0x23, 0x37, 0x99, 0x89, 0xe7	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.132	0xe7431934, 0x0670, 0x4212, 0x93, 0xc1, 0x7e, 0xa2, 0xcd, 0xa4, 0x37, 0x63	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.133	0x2e312df2, 0xfefe, 0x4ae5, 0x9d, 0x69, 0x05, 0x86, 0xfd, 0x52, 0x10, 0x5b	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.134	0x9ac02d05, 0xe755, 0x41df, 0x99, 0x3c, 0x50, 0x95, 0x5f, 0x24, 0x93, 0xac	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with EXCLUSIVE in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.135	0xce3d9684, 0x87fe, 0x47b0, 0x92, 0xee, 0xee, 0x7b, 0x92, 0x45, 0x76, 0x24	**BS.OpenProtocol – OpenProtocol()** with **EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **EXCLUSIVE** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **EXCLUSIVE** again. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.8.136	0x3432eee2, 0x767c, 0x4127, 0xb3, 0x1e, 0xdc, 0x3d, 0xe9, 0x46, 0x9b, 0xfb	**BS.OpenProtocol – OpenProtocol()** with **EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **EXCLUSIVE** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **EXCLUSIVE** again. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.8.137	0xf0aa6cf6, 0x77be, 0x45c8, 0x8c, 0x66, 0x90, 0xfa, 0xb7, 0x2b, 0x2d, 0x27	**BS.OpenProtocol – OpenProtocol()** with **EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **EXCLUSIVE** again. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.8.138	0xc9eaa206, 0x2ef4, 0x413e, 0xa9, 0xf4, 0x39, 0x6c, 0x25, 0x98, 0x73, 0x71	**BS.OpenProtocol – OpenProtocol()** with **EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **EXCLUSIVE** again. The return code should be **EFI_ACCESS_DENIED**.

Number	GUID	Assertion	Test Description
5.1.3.8.139	0x9287f725, 0xd07f, 0x48d8, 0x9f, 0x97, 0x20, 0x67, 0x7c, 0x13, 0x33, 0x15	**BS.OpenProtocol – OpenProtocol()** with **EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** in an external driver that does not follow EFI driver model. 3. Call **OpenProtocol()** with **EXCLUSIVE** again. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.8.140	0x6be45139, 0x977e, 0x4f47, 0x8f, 0x6b, 0x55, 0x76, 0x1a, 0xd6, 0xe9, 0x51	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER** in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.141	0x3b998efd, 0xdd49, 0x407a, 0x81, 0xbf, 0x2f, 0x09, 0xee, 0xa2, 0xe1, 0x86	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER** in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.142	0x11c39508, 0xd7de, 0x463f, 0x87, 0x14, 0xee, 0xe7, 0x28, 0xd7, 0x65, 0x67	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER** in an external driver that does not follow EFI driver model. The driver should be loaded.
5.1.3.8.143	0x5aa89475, 0x844a, 0x4dc9, 0xab, 0x3d, 0xf0, 0x11, 0xd8, 0xff, 0xf1, 0x85	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a new handle. 2. Call **OpenProtocol()** with **BY_DRIVER** in an external driver that does not follow EFI driver model. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.8.144	0x15bef96e, 0x8712, 0x4e4c, 0x98, 0x6e, 0xc9, 0x2f, 0x86, 0x95, 0xe8, 0x9b	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.145	0xe742b47c, 0xc569, 0x4dbf, 0x84, 0x9c, 0xc6, 0x15, 0x06, 0x96, 0x7b, 0x5d	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. The return code should be EFI_SUCCESS.
5.1.3.8.146	0xd7c5b9e3, 0x8cb3, 0x4b37, 0x96, 0x3d, 0x62, 0x89, 0xf4, 0x5f, 0x78, 0xa8	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.147	0xfdb87c6a, 0x7b50, 0x4cf3, 0x98, 0xa8, 0x7a, 0xd8, 0x5c, 0x14, 0x2b, 0x94	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.148	0xe5554329, 0x1e0e, 0x4f6e, 0x92, 0xaa, 0xcd, 0x60, 0x3f, 0x12, 0xe1, 0xcd	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.149	0x79012c79, 0x7aa1, 0x4404, 0x8a, 0x1a, 0x81, 0x33, 0x91, 0x8e, 0x38, 0xd0	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with EXCLUSIVE in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.150	0xefbca8ed, 0x9ab2, 0x474a, 0xb0, 0x93, 0xb2, 0x23, 0xbc, 0x1a, 0xe2, 0x77	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with EXCLUSIVE in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.151	0xebc46e3f, 0xfd62, 0x42b4, 0x95, 0xaf, 0x4c, 0x7a, 0x75, 0xe8, 0x9e, 0x4a	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with EXCLUSIVE in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.152	0xfad30cbf, 0xd6e6, 0x4c1c, 0x96, 0xfa, 0x68, 0xb7, 0x28, 0xff, 0x50, 0x5f	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.153	0xab90cd9e, 0x9e8b, 0x4fd3, 0x87, 0x32, 0x10, 0x04, 0x83, 0x07, 0x7c, 0x1a	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.154	0x7a6722e3, 0x1211, 0x475f, 0xa8, 0x4c, 0x07, 0xe7, 0xe6, 0xfe, 0xdc, 0xb6	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a new handle. 2. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE in an external driver that does not follow EFI driver model. 3. Call OpenProtocol() with BY_DRIVER | EXCLUSIVE again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.155	0x83a9ba94, 0xf3c2, 0x4760, 0x83, 0x10, 0x05, 0x33, 0x23, 0x0c, 0xb7, 0x64	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_APPLICATION	1. Install 3 protocols TestProtocol1 ~ 3 onto the TestHandle. 2. Connect TestDriver1 to TestHandle. TestDriver1 should be connected to TestHandle.

Number	GUID	Assertion	Test Description
5.1.3.8.156	0x7b6e0075, 0xd3c3, 0x4f8b, 0x82, 0xf5, 0xd8, 0x1f, 0x1a, 0x7a, 0xba, 0x52	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_CALLBACK	1. Install 3 protocols TestProtocol1 ~ 3 onto the TestHandle. 2. Connect TestDriver1 to TestHandle. TestDriver1 should be connected to TestHandle.
5.1.3.8.157	0x10640387, 0xdb3c, 0x43ee, 0xb3, 0x22, 0xb6, 0x7d, 0x14, 0x7f, 0x63, 0xd9	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_NOTIFY	1. Install 3 protocols TestProtocol1 ~ 3 onto the TestHandle. 2. Connect TestDriver1 to TestHandle. TestDriver1 should be connected to TestHandle.
5.1.3.8.158	0xe0026b5f, 0xbc98, 0x4090, 0xa6, 0x7d, 0xc3, 0x38, 0xe5, 0x7a, 0x2d, 0xf1	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_APPLICATION	1. Install 3 protocols TestProtocol1 ~ 3 onto the TestHandle. 2. Connect TestDriver1 to TestHandle. 3. Connect TestDriver1 to TestHandle. TestDriver1 should be started.
5.1.3.8.159	0x19bb7f70, 0x3cd8, 0x40d0, 0xbb, 0x23, 0x23, 0xa5, 0x26, 0xd8, 0x85, 0x9a	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_CALLBACK	1. Install 3 protocols TestProtocol1 ~ 3 onto the TestHandle. 2. Connect TestDriver1 to TestHandle. 3. Connect TestDriver1 to TestHandle. TestDriver1 should be started.
5.1.3.8.160	0xdc53e9ee, 0x0750, 0x4a79, 0x99, 0x47, 0x74, 0x54, 0x27, 0xab, 0xc0, 0xf8	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_NOTIFY	1. Install 3 protocols TestProtocol1 ~ 3 onto the TestHandle. 2. Connect TestDriver1 to TestHandle. 3. Connect TestDriver1 to TestHandle. TestDriver1 should be started.

Number	GUID	Assertion	Test Description
5.1.3.8.161	0x797d1b46, 0x6dae, 0x4b5a, 0x93, 0x6c, 0xdf, 0x8a, 0x72, 0xa5, 0xee, 0xe5	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_APPLICATION	1. Install 3 protocols TestProtocol1 ~ 3 onto the TestHandle. 2. Connect TestDriver1 to TestHandle, and open TestProtocol1 BY_DRIVER. 3. Connect TestDriver1 to TestHandle, and open TestProtocol1 BY_DRIVER again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.162	0xa170980c, 0x8d97, 0x4d09, 0x83, 0x9b, 0x5a, 0xde, 0x94, 0x98, 0x05, 0x6e	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_CALLBACK	1. Install 3 protocols TestProtocol1 ~ 3 onto the TestHandle. 2. Connect TestDriver1 to TestHandle, and open TestProtocol1 BY_DRIVER. 3. Connect TestDriver1 to TestHandle, and open TestProtocol1 BY_DRIVER again. The return code should be EFI_ACCESS_DENIED.
5.1.3.8.163	0xe752f97b, 0x97cb, 0x4607, 0x96, 0xb3, 0xb8, 0x2d, 0x60, 0xdb, 0x4f, 0x5c	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_NOTIFY	1. Install 3 protocols TestProtocol1 ~ 3 onto the TestHandle. 2. Connect TestDriver1 to TestHandle, and open TestProtocol1 BY_DRIVER. 3. Connect TestDriver1 to TestHandle, and open TestProtocol1 BY_DRIVER again. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.164	0x8a68d655, 0xed01, 0x4d96, 0xbf, 0x66, 0x85, 0x18, 0x8c, 0x23, 0xa9, 0x19	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_APPLICATION	1. Install 3 protocols TestProtocol1 ~ 3 onto the TestHandle. 2. Connect TestDriver1 to TestHandle, and open TestProtocol1 BY_DRIVER. 3. Connect TestDriver1 to TestHandle, and open TestProtocol1 BY_DRIVER again. 4. Disconnect TestDriver1. 5. Connect TestDriver1 to TestHandle again. TestDriver1 should be started.
5.1.3.8.165	0xfb8aee98, 0x904f, 0x4f44, 0x9f, 0xb4, 0xeb, 0x40, 0xaa, 0x0c, 0x00, 0x79	BS.OpenProtocol – OpenProtocol() with BY_DRIVER at EFI_TPL_CALLBACK	1. Install 3 protocols TestProtocol1 ~ 3 onto the TestHandle. 2. Connect TestDriver1 to TestHandle, and open TestProtocol1 BY_DRIVER. 3. Connect TestDriver1 to TestHandle, and open TestProtocol1 BY_DRIVER again. 4. Disconnect TestDriver1. 5. Connect TestDriver1 to TestHandle again. TestDriver1 should be started.

Number	GUID	Assertion	Test Description
5.1.3.8.166	0xc7551a68, 0x5aee, 0x4fcf, 0x84, 0x13, 0x6e, 0xe5, 0x5b, 0xdb, 0x7d, 0xa1	`BS.OpenProtocol – OpenProtocol()` with `BY_DRIVER` at `EFI_TPL_NOTIFY`	1. Install 3 protocols `TestProtocol1` ~ 3 onto the `TestHandle`. 2. Connect `TestDriver1` to `TestHandle`, and open `TestProtocol1` `BY_DRIVER`. 3. Connect `TestDriver1` to `TestHandle`, and open `TestProtocol1` `BY_DRIVER` again. 4. Disconnect `TestDriver1`. 5. Connect `TestDriver1` to `TestHandle` again. `TestDriver1` should be started.
5.1.3.8.167	0x0a010fbc, 0x7aa1, 0x4575, 0xb3, 0xad, 0x7a, 0x18, 0xac, 0xb6, 0x9f, 0xe2	`BS.OpenProtocol – OpenProtocol()` with `BY_DRIVER` at `EFI_TPL_APPLICATION`	1. Install 3 protocols `TestProtocol1` ~ 3 onto the `TestHandle`. 2. Connect `TestDriver1` to `TestHandle`, and open `TestProtocol1` `BY_DRIVER`. 3. Connect `TestDriver1` to `TestHandle`, and open `TestProtocol1` `BY_DRIVER` again. 4. Disconnect `TestDriver1`. 5. Connect `TestDriver1` to `TestHandle`, and open `TestProtocol1` `BY_DRIVER` again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.8.168	0xb3e52ffe, 0x74fc, 0x4866, 0x86, 0x3b, 0xa7, 0x4b, 0x74, 0xe8, 0x2a, 0xcc	`BS.OpenProtocol – OpenProtocol()` with `BY_DRIVER` at `EFI_TPL_CALLBACK`	1. Install 3 protocols `TestProtocol1` ~ 3 onto the `TestHandle`. 2. Connect `TestDriver1` to `TestHandle`, and open `TestProtocol1` `BY_DRIVER`. 3. Connect `TestDriver1` to `TestHandle`, and open `TestProtocol1` `BY_DRIVER` again. 4. Disconnect `TestDriver1`. 5. Connect `TestDriver1` to `TestHandle`, and open `TestProtocol1` `BY_DRIVER` again. The return code should be `EFI_SUCCESS`.
5.1.3.8.169	0x3ad4c925, 0x4f11, 0x4a0f, 0xa2, 0x88, 0xb9, 0x8a, 0x69, 0xbe, 0xb7, 0x2b	`BS.OpenProtocol – OpenProtocol()` with `BY_DRIVER` at `EFI_TPL_NOTIFY`	1. Install 3 protocols `TestProtocol1` ~ 3 onto the `TestHandle`. 2. Connect `TestDriver1` to `TestHandle`, and open `TestProtocol1` `BY_DRIVER`. 3. Connect `TestDriver1` to `TestHandle`, and open `TestProtocol1` `BY_DRIVER` again. 4. Disconnect `TestDriver1`. 5. Connect `TestDriver1` to `TestHandle`, and open `TestProtocol1` `BY_DRIVER` again. The return code should be `EFI_SUCCESS`.
5.1.3.8.170	0x83aba934, 0x0692, 0x4016, 0x8f, 0x0c, 0x81, 0xf9, 0x2a, 0x02, 0xed, 0x0b	`BS.OpenProtocol – OpenProtocol()` with `EXCLUSIVE` at `EFI_TPL_APPLICATION`	1. Install 4 protocols `TestProtocol1` ~ 4 onto the `TestHandle`. 2. Connect `TestDriver3` to `TestHandle`. 3. Connect `TestDriver4` to `TestHandle` again. `TestDriver3` should be started.

Number	GUID	Assertion	Test Description
5.1.3.8.171	0xb60356c6, 0x15bc, 0x4064, 0xb2, 0xa9, 0x66, 0x3e, 0x04, 0x97, 0xb5, 0x8a	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_CALLBACK	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle. 3. Connect TestDriver4 to TestHandle again. TestDriver3 should be started.
5.1.3.8.172	0xb3d7daa1, 0xd69b, 0x4e88, 0xa6, 0xbb, 0x04, 0x40, 0x59, 0xcf, 0xed, 0x36	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_NOTIFY	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle. 3. Connect TestDriver4 to TestHandle again. TestDriver3 should be started.
5.1.3.8.173	0xd4e25744, 0x0bb8, 0x437f, 0xba, 0x71, 0x39, 0xf9, 0x3b, 0xc5, 0x9a, 0x19	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_APPLICATION	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle. 3. Connect TestDriver4 to TestHandle again. TestDriver4 should be started.
5.1.3.8.174	0xbd89128d, 0x9a44, 0x4807, 0xae, 0x6a, 0x9b, 0xa1, 0x59, 0x21, 0x66, 0x04	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_CALLBACK	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle. 3. Connect TestDriver4 to TestHandle again. TestDriver4 should be started.
5.1.3.8.175	0x563d5e3f, 0x426a, 0x405b, 0x8a, 0xb4, 0x2d, 0x10, 0x5d, 0x26, 0x97, 0xb2	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_NOTIFY	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle. 3. Connect TestDriver4 to TestHandle again. TestDriver4 should be started.

Number	GUID	Assertion	Test Description
5.1.3.8.176	0x472c7cc3, 0xb765, 0x4f4a, 0x87, 0xe2, 0x6b, 0xe5, 0x39, 0x38, 0x95, 0xd7	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_APPLICATION	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE, BY_DRIVER | EXCLUSIVE, BY_DRIVER and BY_DRIVER. 3. Connect TestDriver4 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE. The return code should be EFI_ACCESS_DENIED, EFI_ACCESS_DENIED, EFI_SUCCESS and EFI_ACCESS_DENIED.
5.1.3.8.177	0xde8702c3, 0xd40c, 0x429a, 0xa4, 0xc0, 0x36, 0xda, 0x2e, 0xd1, 0xa5, 0x9c	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_CALLBACK	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE, BY_DRIVER | EXCLUSIVE, BY_DRIVER and BY_DRIVER. 3. Connect TestDriver4 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE. The return code should be EFI_ACCESS_DENIED, EFI_ACCESS_DENIED, EFI_SUCCESS and EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.178	0x33839db3, 0x2c94, 0x470a, 0xa8, 0x1d, 0x3d, 0xb2, 0x88, 0x1a, 0x42, 0x42	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_NOTIFY	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE, BY_DRIVER | EXCLUSIVE, BY_DRIVER and BY_DRIVER. 3. Connect TestDriver4 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE. The return code should be EFI_ACCESS_DENIED, EFI_ACCESS_DENIED, EFI_SUCCESS and EFI_ACCESS_DENIED.
5.1.3.8.179	0xf11cc5b4, 0xfe6e, 0x48c7, 0xaf, 0xab, 0x44, 0x6d, 0x5c, 0x66, 0x51, 0xfe	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_APPLICATION	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE, BY_DRIVER | EXCLUSIVE, BY_DRIVER and BY_DRIVER. 3. Connect TestDriver4 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE. 4. Connect TestDriver4 to TestHandle. TestDriver4 should be started.

Number	GUID	Assertion	Test Description
5.1.3.8.180	0xd6d0a54f, 0x30e5, 0x42f5, 0x96, 0x7b, 0x1f, 0x8d, 0xb0, 0xa4, 0xc5, 0xbe	`BS.OpenProtocol – OpenProtocol()` with `EXCLUSIVE` at `EFI_TPL_CALLBACK`	1. Install 4 protocols `TestProtocol1` ~ 4 onto the `TestHandle`. 2. Connect `TestDriver3` to `TestHandle`, and call `OpenProtocol()` with `TestProtocol1` ~ 4 `EXCLUSIVE`, `BY_DRIVER | EXCLUSIVE`, `BY_DRIVER` and `BY_DRIVER`. 3. Connect `TestDriver4` to `TestHandle`, and call `OpenProtocol()` with `TestProtocol1` ~ 4 `EXCLUSIVE`. 4. Connect `TestDriver4` to `TestHandle`. `TestDriver4` should be started.
5.1.3.8.181	0xfa423bb7, 0x980a, 0x4638, 0x9d, 0xa1, 0xd3, 0x20, 0xc4, 0x1d, 0x6f, 0xd2	`BS.OpenProtocol – OpenProtocol()` with `EXCLUSIVE` at `EFI_TPL_NOTIFY`	1. Install 4 protocols `TestProtocol1` ~ 4 onto the `TestHandle`. 2. Connect `TestDriver3` to `TestHandle`, and call `OpenProtocol()` with `TestProtocol1` ~ 4 `EXCLUSIVE`, `BY_DRIVER | EXCLUSIVE`, `BY_DRIVER` and `BY_DRIVER`. 3. Connect `TestDriver4` to `TestHandle`, and call `OpenProtocol()` with `TestProtocol1` ~ 4 `EXCLUSIVE`. 4. Connect `TestDriver4` to `TestHandle`. `TestDriver4` should be started.

Number	GUID	Assertion	Test Description
5.1.3.8.182	0x68460dff, 0x5f3a, 0x46bb, 0x90, 0xd3, 0xec, 0x3b, 0x90, 0xc0, 0x5b, 0x11	BS.OpenProtocol – OpenProtocol() with EXCLUSIVE at EFI_TPL_APPLICATION	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE, BY_DRIVER | EXCLUSIVE, BY_DRIVER and BY_DRIVER. 3. Connect TestDriver4 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE. 4. Connect TestDriver4 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE. The return code should be EFI_SUCCESS, EFI_SUCCESS, EFI_ACCESS_DENIED and EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.8.183	0x60052ae4, 0x622a, 0x4246, 0x97, 0x10, 0xed, 0x37, 0xf1, 0xb7, 0x7a, 0xcd	`BS.OpenProtocol – OpenProtocol()` with `EXCLUSIVE` at `EFI_TPL_CALLBACK`	1. Install 4 protocols `TestProtocol1` ~ 4 onto the `TestHandle`. 2. Connect `TestDriver3` to `TestHandle`, and call `OpenProtocol()` with `TestProtocol1` ~ 4 `EXCLUSIVE`, `BY_DRIVER | EXCLUSIVE`, `BY_DRIVER` and `BY_DRIVER`. 3. Connect `TestDriver4` to `TestHandle`, and call `OpenProtocol()` with `TestProtocol1` ~ 4 `EXCLUSIVE`. 4. Connect `TestDriver4` to `TestHandle`, and call `OpenProtocol()` with `TestProtocol1` ~ 4 `EXCLUSIVE`. The return code should be `EFI_SUCCESS`, `EFI_SUCCESS`, `EFI_ACCESS_DENIED` and `EFI_ACCESS_DENIED`.

Number	GUID	Assertion	Test Description
5.1.3.8.184	0x1585ecb8, 0x2066, 0x4089, 0xa7, 0x29, 0x95, 0xee, 0x19, 0x8b, 0x15, 0xab	**BS.OpenProtocol – OpenProtocol()** with **EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**, **BY_DRIVER | EXCLUSIVE**, **BY_DRIVER** and **BY_DRIVER**. 3. Connect **TestDriver4** to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. 4. Connect **TestDriver4** to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. The return code should be **EFI_SUCCESS**, **EFI_SUCCESS**, **EFI_ACCESS_DENIED** and **EFI_ACCESS_DENIED**.
5.1.3.8.185	0x1708f46c, 0xa0ea, 0x4fc9, 0x8d, 0xb6, 0x16, 0xfc, 0x17, 0x2d, 0x49, 0x2c	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**. 3. Connect **TestDriver4** to **TestHandle** again. **TestDriver3** should be started.
5.1.3.8.186	0xdc300053, 0x5377, 0x407f, 0x8a, 0x70, 0x20, 0x2e, 0x63, 0x01, 0xd7, 0x54	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**. 3. Connect *TestDriver5* to **TestHandle** again. **TestDriver3** should be started.

Number	GUID	Assertion	Test Description
5.1.3.8.187	0xae0696f6, 0x4ee1, 0x4de7, 0x9c, 0x4d, 0x4d, 0x7b, 0x3a, 0xa6, 0x4f, 0xe8	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**. 3. Connect *TestDriver5* to **TestHandle** again. **TestDriver3** should be started.
5.1.3.8.188	0xe2c08d3a, 0x218e, 0x411c, 0x95, 0xcf, 0x38, 0x85, 0xb3, 0x75, 0xe6, 0xa7	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**. 3. Connect *TestDriver5* to **TestHandle** again. *TestDriver5* should be started.
5.1.3.8.189	0xcda7ab9f, 0x66db, 0x4d0c, 0xb2, 0x1d, 0x92, 0x8d, 0x6c, 0xcd, 0x63, 0x9d	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**. 3. Connect *TestDriver5* to **TestHandle** again. *TestDriver5* should be started.
5.1.3.8.190	0xfc0c893e, 0x307c, 0x403f, 0xbe, 0x98, 0xaf, 0xc6, 0x6b, 0xee, 0xfb, 0xa2	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**. 3. Connect *TestDriver5* to **TestHandle** again. *TestDriver5* should be started.

Number	GUID	Assertion	Test Description
5.1.3.8.191	0x34ba0d95, 0x7597, 0x4a6e, 0xa8, 0xd5, 0x78, 0x61, 0x49, 0xca, 0x9e, 0xd7	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**, **BY_DRIVER | EXCLUSIVE**, **BY_DRIVER** and **BY_DRIVER**. 3. Connect *TestDriver5* to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. The return code should be **EFI_ACCESS_DENIED**, **EFI_ACCESS_DENIED**, **EFI_SUCCESS** and **EFI_ACCESS_DENIED**.
5.1.3.8.192	0xc7d28ea7, 0x0d76, 0x4878, 0xab, 0x12, 0x0c, 0xd1, 0x06, 0xe2, 0x03, 0x3d	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**, **BY_DRIVER | EXCLUSIVE**, **BY_DRIVER** and **BY_DRIVER**. 3. Connect *TestDriver5* to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. The return code should be **EFI_ACCESS_DENIED**, **EFI_ACCESS_DENIED**, **EFI_SUCCESS** and **EFI_ACCESS_DENIED**.

Number	GUID	Assertion	Test Description
5.1.3.8.193	0x1036062c, 0x901d, 0x4ea1, 0x95, 0x8f, 0xa7, 0x38, 0xf0, 0x82, 0x74, 0x4c	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_NOTIFY	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE, BY_DRIVER | EXCLUSIVE, BY_DRIVER and BY_DRIVER. 3. Connect *TestDriver5* to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE. The return code should be EFI_ACCESS_DENIED, EFI_ACCESS_DENIED, EFI_SUCCESS and EFI_ACCESS_DENIED.
5.1.3.8.194	0x27ebea38, 0x414d, 0x45f9, 0x86, 0x7d, 0xb5, 0x71, 0xd6, 0x02, 0xd6, 0x00	BS.OpenProtocol – OpenProtocol() with BY_DRIVER | EXCLUSIVE at EFI_TPL_APPLICATION	1. Install 4 protocols TestProtocol1 ~ 4 onto the TestHandle. 2. Connect TestDriver3 to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE, BY_DRIVER | EXCLUSIVE, BY_DRIVER and BY_DRIVER. 3. Connect *TestDriver5* to TestHandle, and call OpenProtocol() with TestProtocol1 ~ 4 EXCLUSIVE. 4. Connect *TestDriver5* to TestHandle. *TestDriver5* should be started.

Number	GUID	Assertion	Test Description
5.1.3.8.195	0x3483f2b1, 0x4e0f, 0x4b94, 0x85, 0x4c, 0x41, 0x62, 0x2c, 0x94, 0xc9, 0x30	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**, **BY_DRIVER | EXCLUSIVE**, **BY_DRIVER** and **BY_DRIVER**. 3. Connect *TestDriver5* to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. 4. Connect *TestDriver5* to **TestHandle**. *TestDriver5* should be started.
5.1.3.8.196	0x7a490c15, 0xe965, 0x404d, 0xa8, 0xec, 0xd2, 0x65, 0x12, 0x2f, 0x52, 0x87	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**, **BY_DRIVER | EXCLUSIVE**, **BY_DRIVER** and **BY_DRIVER**. 3. Connect *TestDriver5* to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. 4. Connect *TestDriver5* to **TestHandle**. *TestDriver5* should be started.

Number	GUID	Assertion	Test Description
5.1.3.8.197	0xb4aeff8d, 0x1836, 0x4298, 0x9f, 0x53, 0x7f, 0x50, 0x87, 0x2a, 0x35, 0x44	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**, **BY_DRIVER | EXCLUSIVE**, **BY_DRIVER** and **BY_DRIVER**. 3. Connect *TestDriver5* to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. 4. Connect *TestDriver5* to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. The return code should be **EFI_SUCCESS**, **EFI_SUCCESS**, **EFI_SUCCESS** and **EFI_ACCESS_DENIED**.

Number	GUID	Assertion	Test Description
5.1.3.8.198	0x3ead5760, 0x74d2, 0x4780, 0x8c, 0x9d, 0x92, 0x6e, 0x02, 0x5d, 0x9a, 0x2a	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**, **BY_DRIVER | EXCLUSIVE**, **BY_DRIVER** and **BY_DRIVER**. 3. Connect *TestDriver5* to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. 4. Connect *TestDriver5* to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. The return code should be **EFI_SUCCESS**, **EFI_SUCCESS**, **EFI_SUCCESS** and **EFI_ACCESS_DENIED**.

Number	GUID	Assertion	Test Description
5.1.3.8.199	0xb4456e5c, 0x35cc, 0x49ff, 0xb2, 0x28, 0xe1, 0x99, 0xd9, 0x8a, 0xf2, 0xe8	**BS.OpenProtocol – OpenProtocol()** with **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install 4 protocols **TestProtocol1** ~ 4 onto the **TestHandle**. 2. Connect **TestDriver3** to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**, **BY_DRIVER | EXCLUSIVE**, **BY_DRIVER** and **BY_DRIVER**. 3. Connect *TestDriver5* to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. 4. Connect *TestDriver5* to **TestHandle**, and call **OpenProtocol()** with **TestProtocol1** ~ 4 **EXCLUSIVE**. The return code should be **EFI_SUCCESS**, **EFI_SUCCESS**, **EFI_SUCCESS** and **EFI_ACCESS_DENIED**.

3.3.9 CloseProtocol()

Number	GUID	Assertion	Test Description
5.1.3.9.1	0x6b30ee3e, 0x6d78, 0x4542, 0xbd, 0x82, 0x62, 0x0c, 0xeb, 0x76, 0x89, 0xcc	BS.CloseProtocol – CloseProtocol() returns EFI_INVALID_PARAMETER with invalid handle.	1. Call CloseProtocol() with invalid Handle. The return code should be EFI_INVALID_PARAMETER.
5.1.3.9.2	0x3c2ef125, 0x10e5, 0x4bb3, 0xaa, 0x70, 0xf9, 0x0e, 0x59, 0x1b, 0x2d, 0x49	BS.CloseProtocol – CloseProtocol() returns EFI_INVALID_PARAMETER with invalid agent handle.	1. Call CloseProtocol() with invalid AgentHandle. The return code should be EFI_INVALID_PARAMETER.
5.1.3.9.3	0x4c580583, 0x8720, 0x4018, 0x80, 0x3a, 0xc8, 0x89, 0x46, 0xf9, 0x00, 0x07	BS.CloseProtocol - CloseProtocol() returns EFI_INVALID_PARAMETER with invalid ControllerHandle.	1. Call CloseProtocol() with non-NULL but invalid ControllerHandle. The return code should be EFI_INVALID_PARAMETER.
5.1.3.9.4	0x1b942668, 0xc1d5, 0x4076, 0x9d, 0x42, 0x66, 0x9c, 0xca, 0x03, 0x31, 0xbf	BS.CloseProtocol – CloseProtocol() returns EFI_INVALID_PARAMETER with NULL protocol.	1. Call CloseProtocol() with NULL protocol GUID. The return code should be EFI_INVALID_PARAMETER.
5.1.3.9.5	0x35615f53, 0x7ce9, 0x491a, 0x8d, 0x3b, 0x74, 0xa4, 0x12, 0x31, 0x19, 0x1f	BS.CloseProtocol – CloseProtocol() returns EFI_NOT_FOUND with never installed protocol.	1. Call CloseProtocol() to close a protocol that is not installed on the handle. The return code should be EFI_NOT_FOUND.
5.1.3.9.6	0x60813c05, 0x9614, 0x42d6, 0xb3, 0xc1, 0x48, 0xcb, 0x7b, 0x3c, 0x5a, 0xe9	BS.CloseProtocol - CloseProtocol() returns EFI_NOT_FOUND with never opened protocol.	1. Call CloseProtocol() to close a protocol. The return code should be EFI_NOT_FOUND.
5.1.3.9.7	0x78a501c8, 0x3d70, 0x4c55, 0x99, 0x98, 0xfc, 0x8c, 0x64, 0x4c, 0xe8, 0xe0	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened BY_HANDLE_PROTOCOL at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.9.8	0x25258038, 0xc526, 0x4c50, 0xbd, 0x67, 0x61, 0x41, 0x93, 0x31, 0xf0, 0xfc	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened BY_HANDLE_PROTOCOL at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.9	0xd4d3a269, 0x2972, 0x4613, 0xb2, 0xe4, 0x40, 0x47, 0xf3, 0x1e, 0xd6, 0xe8	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened BY_HANDLE_PROTOCOL at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.10	0x3583d756, 0xee15, 0x49d2, 0xa8, 0x8d, 0xe4, 0xe0, 0x34, 0xb4, 0xe5, 0xa7	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_HANDLE_PROTOCOL at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.
5.1.3.9.11	0x8d1b0e42, 0x68c4, 0x4118, 0xa7, 0xb4, 0xb7, 0x38, 0xc8, 0xca, 0x72, 0xd5	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_HANDLE_PROTOCOL at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.
5.1.3.9.12	0x337f5477, 0xf41a, 0x4b1a, 0x87, 0x1c, 0x06, 0xcc, 0xf0, 0x99, 0xb8, 0xb4	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_HANDLE_PROTOCOL at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.

Number	GUID	Assertion	Test Description
5.1.3.9.13	0xb975f9f6, 0x7a4e, 0x44d4, 0x80, 0x37, 0xe4, 0xd1, 0x4f, 0x18, 0xb9, 0x46	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. 4. Call `CloseProtocol()` to close the protocol again. The return code should be `EFI_NOT_FOUND`.
5.1.3.9.14	0x2823a668, 0xe04f, 0x4fb6, 0xbe, 0x2a, 0x90, 0x58, 0x7f, 0x8e, 0xc5, 0x0c	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. 4. Call `CloseProtocol()` to close the protocol again. The return code should be `EFI_NOT_FOUND`.
5.1.3.9.15	0xc1c93781, 0x3316, 0x440f, 0x9b, 0x1b, 0x0f, 0xff, 0x2e, 0x0e, 0xc3, 0xe5	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. 4. Call `CloseProtocol()` to close the protocol again. The return code should be `EFI_NOT_FOUND`.
5.1.3.9.16	0xcf2eecf8, 0x864e, 0x4092, 0x9f, 0xd1, 0x2b, 0xe8, 0xd5, 0x57, 0x8e, 0xdb	`BS.CloseProtocol – CloseProtocol()` returns `EFI_SUCCESS` with opened `GET_PROTOCOL` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. The return code must be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.9.17	0x7cf10a80, 0x3057, 0x4dc3, 0xb6, 0x8a, 0x6a, 0x85, 0xfc, 0x15, 0x47, 0x15	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened GET_PROTOCOL at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.18	0x4c834cc8, 0xf8b9, 0x469c, 0x87, 0x26, 0x88, 0x2c, 0x1b, 0x32, 0xb2, 0x93	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened GET_PROTOCOL at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.19	0xb6adc12e, 0xca4a, 0x4ee1, 0xae, 0x13, 0x97, 0xea, 0x7f, 0xb2, 0x54, 0x7d	BS.CloseProtocol – CloseProtocol() closes the protocol opened GET_PROTOCOL at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.
5.1.3.9.20	0x7154668b, 0xb7a6, 0x416c, 0xb5, 0x40, 0x66, 0x82, 0x90, 0xb0, 0x73, 0x91	BS.CloseProtocol – CloseProtocol() closes the protocol opened GET_PROTOCOL at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.
5.1.3.9.21	0x5fce55ec, 0x6a72, 0x468b, 0x9c, 0x5c, 0x6a, 0x55, 0x87, 0x13, 0xfd, 0xba	BS.CloseProtocol – CloseProtocol() closes the protocol opened GET_PROTOCOL at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.

Number	GUID	Assertion	Test Description
5.1.3.9.22	0x530fbeb7, 0xaf17, 0x4184, 0x82, 0x22, 0x84, 0x15, 0xfd, 0x36, 0x62, 0x35	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `GET_PROTOCOL` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. 4. Call `CloseProtocol()` to close the protocol again. The return code should be `EFI_NOT_FOUND`.
5.1.3.9.23	0x1d2c0ca2, 0x64b8, 0x49bd, 0x81, 0x52, 0x04, 0x23, 0x39, 0xb7, 0x94, 0xbd	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `GET_PROTOCOL` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. 4. Call `CloseProtocol()` to close the protocol again. The return code should be `EFI_NOT_FOUND`.
5.1.3.9.24	0xc6e9a0d6, 0x964c, 0x4f62, 0xa9, 0xa2, 0x8b, 0x5a, 0xef, 0x0b, 0x4d, 0x9e	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `GET_PROTOCOL` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. 4. Call `CloseProtocol()` to close the protocol again. The return code should be `EFI_NOT_FOUND`.
5.1.3.9.25	0xd0252221, 0xed8e, 0x4b29, 0x94, 0x1d, 0xdd, 0x77, 0x34, 0xb0, 0x46, 0x38	`BS.CloseProtocol – CloseProtocol()` returns `EFI_SUCCESS` with opened `TEST_PROTOCOL` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. The return code must be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.9.26	0xf13e1252, 0x4a59, 0x457c, 0x81, 0xe3, 0x8d, 0xe8, 0x98, 0x51, 0x0c, 0xbc	`BS.CloseProtocol – CloseProtocol()` returns `EFI_SUCCESS` with opened `TEST_PROTOCOL` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. The return code must be `EFI_SUCCESS`.
5.1.3.9.27	0x444d4e4f, 0x1f92, 0x4d0f, 0xbd, 0x94, 0x55, 0x8a, 0x18, 0x04, 0x54, 0xb9	`BS.CloseProtocol – CloseProtocol()` returns `EFI_SUCCESS` with opened `TEST_PROTOCOL` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. The return code must be `EFI_SUCCESS`.
5.1.3.9.28	0xdc29e780, 0x458c, 0x4768, 0xbd, 0x74, 0x38, 0x2f, 0x5e, 0x18, 0x1d, 0xcd	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `TEST_PROTOCOL` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. `TestProtocol` should not be opened.
5.1.3.9.29	0xf593bade, 0xdf33, 0x434c, 0xa4, 0x09, 0x2f, 0xda, 0x04, 0xb2, 0x9a, 0x37	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `TEST_PROTOCOL` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. `TestProtocol` should not be opened.
5.1.3.9.30	0xbd6838e1, 0x229a, 0x405b, 0xa8, 0xcd, 0x80, 0xb9, 0xd3, 0x02, 0xd2, 0x69	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `TEST_PROTOCOL` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. `TestProtocol` should not be opened.

Number	GUID	Assertion	Test Description
5.1.3.9.31	0x7be802be, 0xc38c, 0x41ca, 0x86, 0xb5, 0x44, 0x99, 0x2b, 0x90, 0x69, 0x73	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `TEST_PROTOCOL` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. 4. Call `CloseProtocol()` to close the protocol again. The return code should be `EFI_NOT_FOUND`.
5.1.3.9.32	0x6b61aade, 0xdf67, 0x4867, 0x96, 0xe8, 0x81, 0x18, 0x82, 0x05, 0x85, 0x3b	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `TEST_PROTOCOL` at `EFI_TPL_CALLBACK`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. 4. Call `CloseProtocol()` to close the protocol again. The return code should be `EFI_NOT_FOUND`.
5.1.3.9.33	0xd235784c, 0x06dd, 0x4dcf, 0x92, 0x13, 0xde, 0xc3, 0xe6, 0x03, 0xf2, 0x37	`BS.CloseProtocol – CloseProtocol()` closes the protocol opened `TEST_PROTOCOL` at `EFI_TPL_NOTIFY`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `CloseProtocol()` to close the protocol. 4. Call `CloseProtocol()` to close the protocol again. The return code should be `EFI_NOT_FOUND`.
5.1.3.9.34	0xc86b323b, 0xb7d3, 0x491f, 0x9b, 0x05, 0xfc, 0x6b, 0x59, 0x6a, 0x93, 0xb8	`BS.CloseProtocol – CloseProtocol()` returns `EFI_SUCCESS` with opened `BY_CHILD_CONTROLLER` at `EFI_TPL_APPLICATION`.	1. Install `TestProtocol1` onto a handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `CloseProtocol()` to close the protocol. The return code must be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.9.35	0xe5a11769, 0x32f0, 0x4c86, 0xb2, 0xe9, 0x5f, 0x34, 0x63, 0xa1, 0xc7, 0xc6	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened BY_CHILD_CONTROLLER at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.36	0x4be096a6, 0x2a05, 0x4edc, 0xa9, 0x98, 0xe9, 0x99, 0xe4, 0x9e, 0xcc, 0x31	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened BY_CHILD_CONTROLLER at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.37	0x005ccabc, 0x4be9, 0x48aa, 0xa4, 0xd9, 0xcd, 0x87, 0xbe, 0xce, 0xf1, 0xed	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_CHILD_CONTROLLER at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.
5.1.3.9.38	0x5634facd, 0x0559, 0x4094, 0x97, 0xd5, 0x27, 0x8d, 0xe8, 0x0f, 0x24, 0x0c	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_CHILD_CONTROLLER at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.
5.1.3.9.39	0xe173576f, 0xc735, 0x4419, 0x95, 0x08, 0x73, 0xb3, 0x26, 0xee, 0x3e, 0x00	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_CHILD_CONTROLLER at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.

Number	GUID	Assertion	Test Description
5.1.3.9.40	0x469d7985, 0x7868, 0x456f, 0x94, 0xb7, 0xb2, 0x24, 0x90, 0x51, 0x16, 0x45	**BS.CloseProtocol – CloseProtocol()** closes the protocol opened **BY_CHILD_CONTROLLER** at **EFI_TPL_APPLICATION**.	1. Install **TestProtocol1** onto a handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_CHILD_CONTROLLER**. 3. Call **CloseProtocol()** to close the protocol. 4. Call **CloseProtocol()** to close the protocol again. The return code should be **EFI_NOT_FOUND**.
5.1.3.9.41	0x604fd72e, 0xbbc7, 0x4693, 0x8e, 0x31, 0xf4, 0x02, 0x21, 0x13, 0xce, 0x6d	**BS.CloseProtocol – CloseProtocol()** closes the protocol opened **BY_CHILD_CONTROLLER** at **EFI_TPL_CALLBACK**.	1. Install **TestProtocol1** onto a handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_CHILD_CONTROLLER**. 3. Call **CloseProtocol()** to close the protocol. 4. Call **CloseProtocol()** to close the protocol again. The return code should be **EFI_NOT_FOUND**.
5.1.3.9.42	0x7d675f3c, 0x592e, 0x4f38, 0x98, 0xe1, 0x28, 0xae, 0xaf, 0x81, 0xdc, 0xfd	**BS.CloseProtocol – CloseProtocol()** closes the protocol opened **BY_CHILD_CONTROLLER** at **EFI_TPL_NOTIFY**.	1. Install **TestProtocol1** onto a handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_CHILD_CONTROLLER**. 3. Call **CloseProtocol()** to close the protocol. 4. Call **CloseProtocol()** to close the protocol again. The return code should be **EFI_NOT_FOUND**.
5.1.3.9.43	0x51365d70, 0xd032, 0x4bb0, 0x9e, 0x2f, 0x45, 0x79, 0xe1, 0xb4, 0x3b, 0xf4	**BS.CloseProtocol – CloseProtocol()** returns **EFI_SUCCESS** with opened **BY_DRIVER** at **EFI_TPL_APPLICATION**.	1. Install **TestProtocol1** onto a handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER**. 3. Call **CloseProtocol()** to close the protocol. The return code must be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.9.44	0xb2dabae2, 0xdf68, 0x41cd, 0xbe, 0x21, 0x94, 0x0c, 0xe2, 0xf0, 0xdc, 0x65	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened BY_DRIVER at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.45	0x2cb2bbe9, 0x81b5, 0x4589, 0xa0, 0xdc, 0xd9, 0xee, 0x6c, 0xd4, 0xf4, 0x48	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened BY_DRIVER at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.46	0x747e6105, 0xab68, 0x4f7d, 0x8c, 0xed, 0x58, 0x90, 0x28, 0x3a, 0xa6, 0xaa	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_DRIVER at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.
5.1.3.9.47	0xa140992a, 0x215c, 0x4fad, 0x8f, 0x2a, 0xd1, 0x50, 0x1f, 0x47, 0x1f, 0x50	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_DRIVER at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.
5.1.3.9.48	0xa3296d1f, 0xc631, 0x42d8, 0xb6, 0xa4, 0x7c, 0x9b, 0xfe, 0xe7, 0x57, 0x83	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_DRIVER at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.

Number	GUID	Assertion	Test Description
5.1.3.9.49	0x3758f47c, 0x0041, 0x434c, 0x83, 0x76, 0x05, 0xeb, 0xba, 0x0f, 0x36, 0x49	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_DRIVER at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call CloseProtocol() to close the protocol. 4. Call CloseProtocol() to close the protocol again. The return code should be EFI_NOT_FOUND.
5.1.3.9.50	0x5bd91b68, 0x4d35, 0x4366, 0xaf, 0x0e, 0x21, 0xf2, 0xcb, 0x6b, 0xe8, 0x13	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_DRIVER at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call CloseProtocol() to close the protocol. 4. Call CloseProtocol() to close the protocol again. The return code should be EFI_NOT_FOUND.
5.1.3.9.51	0x1a02fbba, 0x35b7, 0x43c6, 0x82, 0x56, 0x90, 0x67, 0x18, 0x2f, 0xc4, 0xe0	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_DRIVER at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call CloseProtocol() to close the protocol. 4. Call CloseProtocol() to close the protocol again. The return code should be EFI_NOT_FOUND.
5.1.3.9.52	0x6f75c53a, 0x1e25, 0x4767, 0x87, 0x51, 0x77, 0x5b, 0x18, 0xbc, 0xf5, 0xb0	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened EXCLUSIVE at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.9.53	0x13bc8d9b, 0x3d19, 0x413f, 0x89, 0x28, 0xc8, 0x22, 0xb5, 0x66, 0x2e, 0x96	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened EXCLUSIVE at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.54	0xa7e0326a, 0x01a1, 0x4d41, 0x89, 0xbe, 0x37, 0x75, 0x71, 0xef, 0x88, 0x92	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened EXCLUSIVE at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.55	0xf7e85205, 0x0019, 0x42a4, 0x8d, 0xaa, 0x54, 0xf2, 0xb8, 0x94, 0x0e, 0xeb	BS.CloseProtocol – CloseProtocol() closes the protocol opened EXCLUSIVE at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.
5.1.3.9.56	0x06fba0ca, 0x5fa1, 0x48e0, 0x90, 0x9e, 0x81, 0x24, 0x76, 0x9a, 0x45, 0x41	BS.CloseProtocol – CloseProtocol() closes the protocol opened EXCLUSIVE at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.
5.1.3.9.57	0x04ef3e61, 0xd1e3, 0x474b, 0xac, 0x26, 0x1c, 0x7c, 0xac, 0x35, 0x19, 0x74	BS.CloseProtocol – CloseProtocol() closes the protocol opened EXCLUSIVE at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.

Number	GUID	Assertion	Test Description
5.1.3.9.58	0x1390eee4, 0x2409, 0x478b, 0xbc, 0x37, 0x9d, 0x17, 0x53, 0x2f, 0x68, 0x94	BS.CloseProtocol – CloseProtocol() closes the protocol opened EXCLUSIVE at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. 4. Call CloseProtocol() to close the protocol again. The return code should be EFI_NOT_FOUND.
5.1.3.9.59	0x372e9dd7, 0x4ea1, 0x4eb3, 0x91, 0x2c, 0x20, 0x94, 0x01, 0xde, 0x73, 0xa9	BS.CloseProtocol – CloseProtocol() closes the protocol opened EXCLUSIVE at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. 4. Call CloseProtocol() to close the protocol again. The return code should be EFI_NOT_FOUND.
5.1.3.9.60	0xd5da82f4, 0x43b9, 0x44f3, 0x8d, 0xb1, 0xb8, 0x2a, 0xc8, 0x20, 0x87, 0x7d	BS.CloseProtocol – CloseProtocol() closes the protocol opened EXCLUSIVE at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. 4. Call CloseProtocol() to close the protocol again. The return code should be EFI_NOT_FOUND.
5.1.3.9.61	0xc5fe3e47, 0x3dfa, 0x473f, 0x92, 0x79, 0xfe, 0x66, 0xc4, 0x0d, 0x62, 0xed	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened BY_DRIVER | EXCLUSIVE at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.9.62	0x72360334, 0x3162, 0x469c, 0x9d, 0x43, 0xa7, 0xc5, 0xba, 0xa2, 0x29, 0xa7	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened BY_DRIVER | EXCLUSIVE at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.63	0x679010d8, 0x2815, 0x4114, 0x9d, 0xbc, 0x52, 0xfb, 0x1a, 0x3d, 0x4e, 0x53	BS.CloseProtocol – CloseProtocol() returns EFI_SUCCESS with opened BY_DRIVER | EXCLUSIVE at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. The return code must be EFI_SUCCESS.
5.1.3.9.64	0xeab7d653, 0x9cde, 0x4160, 0xac, 0x7a, 0x85, 0xda, 0xc8, 0xb0, 0xd8, 0xfd	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_DRIVER | EXCLUSIVE at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.
5.1.3.9.65	0x5846a316, 0x5fc2, 0x455a, 0x88, 0xc0, 0x47, 0x85, 0xcd, 0x22, 0xe9, 0x76	BS.CloseProtocol – CloseProtocol() closes the protocol opened BY_DRIVER | EXCLUSIVE at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE. 3. Call CloseProtocol() to close the protocol. TestProtocol should not be opened.

Number	GUID	Assertion	Test Description
5.1.3.9.66	0x7c825d57, 0x616f, 0x43c4, 0x81, 0xa9, 0xd9, 0xab, 0xc1, 0x6b, 0xab, 0x8b	**BS.CloseProtocol – CloseProtocol()** closes the protocol opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**.	1. Install **TestProtocol1** onto a handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **BY_DRIVER | EXCLUSIVE**. 3. Call **CloseProtocol()** to close the protocol. **TestProtocol** should not be opened.
5.1.3.9.67	0x383627c5, 0xf2fa, 0x4b4f, 0xac, 0xa6, 0x66, 0xb2, 0xd9, 0xae, 0xe2, 0xbf	**BS.CloseProtocol – CloseProtocol()** closes the protocol opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_APPLICATION**.	1. Install **TestProtocol1** onto a handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **BY_DRIVER | EXCLUSIVE**. 3. Call **CloseProtocol()** to close the protocol. 4. Call **CloseProtocol()** to close the protocol again. The return code should be **EFI_NOT_FOUND**.
5.1.3.9.68	0x1a6476cd, 0xefa7, 0x4416, 0x94, 0x5a, 0x45, 0x44, 0xae, 0xc1, 0xd1, 0x9d	**BS.CloseProtocol – CloseProtocol()** closes the protocol opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_CALLBACK**.	1. Install **TestProtocol1** onto a handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **BY_DRIVER | EXCLUSIVE**. 3. Call **CloseProtocol()** to close the protocol. 4. Call **CloseProtocol()** to close the protocol again. The return code should be **EFI_NOT_FOUND**.
5.1.3.9.69	0x4e3cb0f2, 0xb5fc, 0x4563, 0x99, 0x6d, 0xcc, 0x44, 0xda, 0x3d, 0xf0, 0xae	**BS.CloseProtocol – CloseProtocol()** closes the protocol opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**.	1. Install **TestProtocol1** onto a handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **BY_DRIVER | EXCLUSIVE**. 3. Call **CloseProtocol()** to close the protocol. 4. Call **CloseProtocol()** to close the protocol again. The return code should be **EFI_NOT_FOUND**.

3.3.10 OpenProtocolInformation()

Number	GUID	Assertion	Test Description
5.1.3.10.1	0x5c23f55a, 0x5ea3, 0x4576, 0x9e, 0xe0, 0x77, 0xb0, 0x0d, 0x9b, 0xf8, 0x22	`BS.OpenProtocolInformation – OpenProtocolInformation()` returns `EFI_NOT_FOUND` with never installed protocol	1. Call `OpenProtocolInformation()` to attempt to retrieve open information of a protocol that is not installed on the handle. The return code should be `EFI_NOT_FOUND`.
5.1.3.10.2	0x551ffed5, 0x5e44, 0x42cc, 0xa1, 0xcc, 0xbf, 0xc8, 0x0e, 0x74, 0x98, 0xcb	`BS.OpenProtocolInformation – OpenProtocolInformation()` returns `EFI_SUCCESS` with valid parameters at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. The return code should be `EFI_SUCCESS`.
5.1.3.10.3	0xa7b17f7d, 0x001e, 0x40db, 0xb6, 0x3e, 0xfc, 0x2f, 0x37, 0xf6, 0xb5, 0xd2	`BS.OpenProtocolInformation – OpenProtocolInformation()` returns `EFI_SUCCESS` with valid parameters at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.10.4	0x8fccb668, 0xf502, 0x4020, 0x8e, 0x48, 0x07, 0x5c, 0x58, 0xfa, 0x55, 0x1a	**BS.OpenProtocolInformation – OpenProtocolInformation()** returns **EFI_SUCCESS** with valid parameters at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. The return code should be **EFI_SUCCESS**.
5.1.3.10.5	0x68534ef5, 0x8cb0, 0x402f, 0x8d, 0x15, 0xa8, 0x0d, 0x38, 0x62, 0x46, 0x27	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. The return *EntryCount* should be 4.
5.1.3.10.6	0x38e40fdd, 0x6338, 0x41da, 0xa6, 0xe2, 0x4b, 0x4b, 0x25, 0x02, 0xdb, 0x4d	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. The return *EntryCount* should be 4.

Number	GUID	Assertion	Test Description
5.1.3.10.7	0x683363d5, 0x821e, 0x4b53, 0xa3, 0x3f, 0x3c, 0x39, 0xbe, 0xfa, 0x17, 0x3b	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. The return *EntryCount* should be 4.
5.1.3.10.8	0x0ba0d7b1, 0x25cd, 0x410d, 0x8b, 0x2e, 0xf8, 0xe9, 0xc4, 0xf4, 0xe0, 0xd7	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. The return *EntryBuffer* should be the expected handle and attributes.
5.1.3.10.9	0x0f467d96, 0x2424, 0x4a85, 0x98, 0x7c, 0xa6, 0xec, 0x5f, 0xcc, 0x4a, 0x04	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. The return *EntryBuffer* should be the expected handle and attributes.

Number	GUID	Assertion	Test Description
5.1.3.10.10	0xeace4c54, 0x5bb2, 0x4419, 0x89, 0x66, 0x67, 0x3d, 0x24, 0xa8, 0x7a, 0x9e	`BS.OpenProtocolInformation – OpenProtocolInformation()` gets the open information at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. The return *`EntryBuffer`* should be the expected handle and attributes.
5.1.3.10.11	0x27a25cb1, 0xbd5e, 0x4ae3, 0xb6, 0xfd, 0xde, 0xd8, 0xb0, 0x1f, 0xc8, 0x0a	`BS.OpenProtocolInformation – OpenProtocolInformation()` gets the open information at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. 4. Open `TestProtocol1` with `BY_DRIVER`. 5. Call `OpenProtocolInformation()` again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.10.12	0x6b60a557, 0xdfc4, 0x4c1b, 0x8a, 0x5a, 0xd8, 0x10, 0x1b, 0xd3, 0x41, 0xd8	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Call OpenProtocolInformation() again. The return code should be EFI_SUCCESS.
5.1.3.10.13	0x3486a27c, 0xb5e7, 0x4d63, 0x8e, 0x24, 0x17, 0x63, 0xdd, 0xae, 0x4b, 0xd5	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Call OpenProtocolInformation() again. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.10.14	0x0d0c3286, 0xefb8, 0x43b0, 0x9b, 0x80, 0xe5, 0x50, 0x8c, 0x6b, 0xa2, 0x54	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Call OpenProtocolInformation() again. The return *EntryCount* should be 5.
5.1.3.10.15	0x5642b941, 0xf367, 0x4a1c, 0x90, 0xb7, 0xd5, 0x81, 0x50, 0x62, 0x0c, 0x10	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Call OpenProtocolInformation() again. The return *EntryCount* should be 5.

Number	GUID	Assertion	Test Description
5.1.3.10.16	0x5811c19c, 0x759f, 0x449b, 0x8f, 0xff, 0x2f, 0xf3, 0x55, 0x64, 0x26, 0xb0	`BS.OpenProtocolInformation – OpenProtocolInformation()` gets the open information at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. 4. Open `TestProtocol1` with `BY_DRIVER`. 5. Call `OpenProtocolInformation()` again. The return *`EntryCount`* should be 5.
5.1.3.10.17	0x6edfefb8, 0x06fa, 0x4aff, 0xaf, 0xbc, 0xad, 0xcc, 0x97, 0xa9, 0x18, 0x98	`BS.OpenProtocolInformation – OpenProtocolInformation()` gets the open information at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. 4. Open `TestProtocol1` with `BY_DRIVER`. 5. Call `OpenProtocolInformation()` again. The return *`EntryBuffer`* should be the expected handle and attributes.

Number	GUID	Assertion	Test Description
5.1.3.10.18	0xa8c20f63, 0x0c01, 0x421c, 0x84, 0x85, 0xbe, 0x36, 0xef, 0xe0, 0x1e, 0x6e	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Call OpenProtocolInformation() again. The return *EntryBuffer* should be the expected handle and attributes.
5.1.3.10.19	0xa926af54, 0x6ccc, 0x4360, 0xab, 0x91, 0xfa, 0x3e, 0xb0, 0x04, 0x56, 0xfb	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Call OpenProtocolInformation() again. The return *EntryBuffer* should be the expected handle and attributes.

Number	GUID	Assertion	Test Description
5.1.3.10.20	0x60f32615, 0x26de, 0x4088, 0x92, 0xf7, 0x42, 0x48, 0xc4, 0xb0, 0x15, 0x62	`BS.OpenProtocolInformation – OpenProtocolInformation()` gets the open information at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. 4. Open `TestProtocol1` with `BY_DRIVER`. 5. Close the `TestProtocol1`. 6. Open `TestProtocol1` with EXCLUSIVE. 7. Call `OpenProtocolInformation()` again. The return code should be `EFI_SUCCESS`.
5.1.3.10.21	0x88b06cc1, 0x07f3, 0x4c2c, 0xa0, 0x66, 0x17, 0x6c, 0xc0, 0xb5, 0x13, 0x52	`BS.OpenProtocolInformation – OpenProtocolInformation()` gets the open information at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. 4. Open `TestProtocol1` with `BY_DRIVER`. 5. Close the `TestProtocol1`. 6. Open `TestProtocol1` with EXCLUSIVE. 7. Call `OpenProtocolInformation()` again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.10.22	0x93cbdeae, 0x7377, 0x4c9c, 0xbb, 0x89, 0x1f, 0xa8, 0x34, 0xa1, 0xb1, 0x50	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. 4. Open **TestProtocol1** with **BY_DRIVER**. 5. Close the **TestProtocol1**. 6. Open **TestProtocol1** with **EXCLUSIVE**. 7. Call **OpenProtocolInformation()** again. The return code should be **EFI_SUCCESS**.
5.1.3.10.23	0x69f77854, 0xd208, 0x4447, 0x80, 0x55, 0xb0, 0x29, 0x0b, 0x5d, 0xdb, 0x99	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. 4. Open **TestProtocol1** with **BY_DRIVER**. 5. Close the **TestProtocol1**. 6. Open **TestProtocol1** with **EXCLUSIVE**. 7. Call **OpenProtocolInformation()** again. The return *EntryCount* should be 1.

Number	GUID	Assertion	Test Description
5.1.3.10.24	0xfdcfbc23, 0x5f95, 0x4ea0, 0xa4, 0xfe, 0xba, 0x00, 0xd7, 0xc5, 0xc4, 0xde	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Close the TestProtocol1. 6. Open TestProtocol1 with EXCLUSIVE. 7. Call OpenProtocolInformation() again. The return *EntryCount* should be 1.
5.1.3.10.25	0xc88b2499, 0x4673, 0x413c, 0x86, 0x75, 0xba, 0xa0, 0xbc, 0x10, 0x54, 0x4d	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Close the TestProtocol1. 6. Open TestProtocol1 with EXCLUSIVE. 7. Call OpenProtocolInformation() again. The return *EntryCount* should be 1.

Number	GUID	Assertion	Test Description
5.1.3.10.26	0x2c1311fb, 0xe4af, 0x4530, 0x93, 0xb7, 0xa2, 0xd5, 0x9a, 0x3f, 0xcf, 0xf7	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Close the TestProtocol1. 6. Open TestProtocol1 with EXCLUSIVE. 7. Call OpenProtocolInformation() again. The return *EntryBuffer* should be the expected handle and attributes.
5.1.3.10.27	0xddb30788, 0x7061, 0x4ea8, 0x8c, 0x84, 0x72, 0xc1, 0x84, 0x60, 0xe6, 0xef	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Close the TestProtocol1. 6. Open TestProtocol1 with EXCLUSIVE. 7. Call OpenProtocolInformation() again. The return *EntryBuffer* should be the expected handle and attributes.

Number	GUID	Assertion	Test Description
5.1.3.10.28	0x88b002c4, 0x19b1, 0x496f, 0xa7, 0x16, 0x8b, 0xaf, 0x50, 0x30, 0xf6, 0x0f	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. 4. Open **TestProtocol1** with **BY_DRIVER**. 5. Close the **TestProtocol1**. 6. Open **TestProtocol1** with **EXCLUSIVE**. 7. Call **OpenProtocolInformation()** again. The return *EntryBuffer* should be the expected handle and attributes.
5.1.3.10.29	0xdda74e1b, 0xfac7, 0x47b4, 0x8a, 0xd2, 0xb8, 0x14, 0x17, 0x38, 0x0e, 0xfc	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. 4. Open **TestProtocol1** with **BY_DRIVER**. 5. Close the **TestProtocol1**. 6. Open **TestProtocol1** with **EXCLUSIVE**. 7. Close the **TestProtocol1**. 8. Open **TestProtocol1** with **BY_DRIVER | EXCLUSIVE**. 9. Call **OpenProtocolInformation()** again. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.10.30	0xb0d45adf, 0xc9aa, 0x416e, 0xb2, 0x39, 0x4a, 0xfe, 0x3b, 0x1a, 0x43, 0xde	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Close the TestProtocol1. 6. Open TestProtocol1 with EXCLUSIVE. 7. Close the TestProtocol1. 8. Open TestProtocol1 with BY_DRIVER | EXCLUSIVE. 9. Call OpenProtocolInformation() again. The return code should be EFI_SUCCESS.
5.1.3.10.31	0x71da8c49, 0x0fe8, 0x4298, 0x80, 0xe9, 0x2e, 0x86, 0x40, 0x9b, 0x15, 0xc6	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Close the TestProtocol1. 6. Open TestProtocol1 with EXCLUSIVE. 7. Close the TestProtocol1. 8. Open TestProtocol1 with BY_DRIVER | EXCLUSIVE. 9. Call OpenProtocolInformation() again. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.10.32	0x63867ba8, 0xa4da, 0x4153, 0x93, 0xd0, 0xe2, 0x67, 0xbe, 0x35, 0x93, 0x14	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. 4. Open **TestProtocol1** with **BY_DRIVER**. 5. Close the **TestProtocol1**. 6. Open **TestProtocol1** with **EXCLUSIVE**. 7. Close the **TestProtocol1**. 8. Open **TestProtocol1** with **BY_DRIVER | EXCLUSIVE**. 9. Call **OpenProtocolInformation()** again. The return *EntryCount* should be 1.
5.1.3.10.33	0x60b01808, 0x28e7, 0x4800, 0xa8, 0x1a, 0x01, 0xa1, 0xbd, 0xec, 0xa5, 0x1f	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. 4. Open **TestProtocol1** with **BY_DRIVER**. 5. Close the **TestProtocol1**. 6. Open **TestProtocol1** with **EXCLUSIVE**. 7. Close the **TestProtocol1**. 8. Open **TestProtocol1** with **BY_DRIVER | EXCLUSIVE**. 9. Call **OpenProtocolInformation()** again. The return *EntryCount* should be 1.

Number	GUID	Assertion	Test Description
5.1.3.10.34	0x1ac2f4d5, 0x980d, 0x49a5, 0xa5, 0xd1, 0x30, 0x82, 0x7c, 0x45, 0x5c, 0x77	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. 4. Open **TestProtocol1** with **BY_DRIVER**. 5. Close the **TestProtocol1**. 6. Open **TestProtocol1** with **EXCLUSIVE**. 7. Close the **TestProtocol1**. 8. Open **TestProtocol1** with **BY_DRIVER | EXCLUSIVE**. 9. Call **OpenProtocolInformation()** again. The return *EntryCount* should be 1.

Number	GUID	Assertion	Test Description
5.1.3.10.35	0xce333372, 0x126d, 0x4d25, 0x93, 0x45, 0x12, 0x1a, 0x45, 0x15, 0xb2, 0x2b	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. 4. Open **TestProtocol1** with **BY_DRIVER**. 5. Close the **TestProtocol1**. 6. Open **TestProtocol1** with **EXCLUSIVE**. 7. Close the **TestProtocol1**. 8. Open **TestProtocol1** with **BY_DRIVER | EXCLUSIVE**. 9. Call **OpenProtocolInformation()** again. The return *EntryBuffer* should be expected handle and attributes.

Number	GUID	Assertion	Test Description
5.1.3.10.36	0xdeb1b1af, 0x90ef, 0x476d, 0xa1, 0xfd, 0xc3, 0x19, 0x44, 0xed, 0x91, 0xe2	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. 4. Open **TestProtocol1** with **BY_DRIVER**. 5. Close the **TestProtocol1**. 6. Open **TestProtocol1** with **EXCLUSIVE**. 7. Close the **TestProtocol1**. 8. Open **TestProtocol1** with **BY_DRIVER | EXCLUSIVE**. 9. Call **OpenProtocolInformation()** again. The return *EntryBuffer* should be expected handle and attributes.

Number	GUID	Assertion	Test Description
5.1.3.10.37	0x6eace800, 0xbc38, 0x4766, 0xb6, 0xb7, 0xa7, 0xff, 0xb1, 0xf3, 0x64, 0x43	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, **TEST_PROTOCOL**, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. 4. Open **TestProtocol1** with **BY_DRIVER**. 5. Close the **TestProtocol1**. 6. Open **TestProtocol1** with **EXCLUSIVE**. 7. Close the **TestProtocol1**. 8. Open **TestProtocol1** with **BY_DRIVER | EXCLUSIVE**. 9. Call **OpenProtocolInformation()** again. The return *EntryBuffer* should be expected handle and attributes.

Number	GUID	Assertion	Test Description
5.1.3.10.38	0x8ca604c4, 0x0b6c, 0x40a9, 0xa5, 0x7d, 0x81, 0x22, 0x5d, 0x02, 0xb8, 0xb1	`BS.OpenProtocolInformation – OpenProtocolInformation()` gets the open information at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. 4. Open `TestProtocol1` with `BY_DRIVER`. 5. Close the `TestProtocol1`. 6. Open `TestProtocol1` with `EXCLUSIVE`. 7. Close the `TestProtocol1`. 8. Open `TestProtocol1` with `BY_DRIVER | EXCLUSIVE`. 9. Close the `TestProtocol1`. 10. Call `OpenProtocolInformation()` again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.10.39	0xfad446e9, 0x9d06, 0x4f7c, 0xbf, 0x91, 0x3b, 0x3b, 0xea, 0xc0, 0x0f, 0xcf	`BS.OpenProtocolInformation – OpenProtocolInformation()` gets the open information at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. 4. Open `TestProtocol1` with `BY_DRIVER`. 5. Close the `TestProtocol1`. 6. Open `TestProtocol1` with `EXCLUSIVE`. 7. Close the `TestProtocol1`. 8. Open `TestProtocol1` with `BY_DRIVER | EXCLUSIVE`. 9. Close the `TestProtocol1`. 10. Call `OpenProtocolInformation()` again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.10.40	0xdb121aed, 0xb553, 0x4fa0, 0x9f, 0xad, 0x12, 0x0b, 0xf4, 0x54, 0xef, 0x9e	`BS.OpenProtocolInformation – OpenProtocolInformation()` gets the open information at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. 4. Open `TestProtocol1` with `BY_DRIVER`. 5. Close the `TestProtocol1`. 6. Open `TestProtocol1` with `EXCLUSIVE`. 7. Close the `TestProtocol1`. 8. Open `TestProtocol1` with `BY_DRIVER | EXCLUSIVE`. 9. Close the `TestProtocol1`. 10. Call `OpenProtocolInformation()` again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.10.41	0xbcf00a90, 0xf775, 0x4103, 0xab, 0x4a, 0x36, 0x41, 0xea, 0xc4, 0xc7, 0xf7	`BS.OpenProtocolInformation – OpenProtocolInformation()` gets the open information at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto a handle. 2. Open `TestProtocol1` with `BY_HANDLE_PROTOCOL`, `GET_PROTOCOL`, `TEST_PROTOCOL`, and `BY_CHILD_CONTROLLER`. 3. Call `OpenProtocolInformation()` on the handle and `TestProtocol1` to retrieve the open information. 4. Open `TestProtocol1` with `BY_DRIVER`. 5. Close the `TestProtocol1`. 6. Open `TestProtocol1` with `EXCLUSIVE`. 7. Close the `TestProtocol1`. 8. Open `TestProtocol1` with `BY_DRIVER | EXCLUSIVE`. 9. Close the `TestProtocol1`. 10. Call `OpenProtocolInformation()` again. The return *`EntryCount`* should be 0.

Number	GUID	Assertion	Test Description
5.1.3.10.42	0x65097fed, 0x6b9e, 0x4365, 0x95, 0xb8, 0x7f, 0xf4, 0xfa, 0xd5, 0x89, 0xe7	BS.OpenProtocolInformation – OpenProtocolInformation() gets the open information at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto a handle. 2. Open TestProtocol1 with BY_HANDLE_PROTOCOL, GET_PROTOCOL, TEST_PROTOCOL, and BY_CHILD_CONTROLLER. 3. Call OpenProtocolInformation() on the handle and TestProtocol1 to retrieve the open information. 4. Open TestProtocol1 with BY_DRIVER. 5. Close the TestProtocol1. 6. Open TestProtocol1 with EXCLUSIVE. 7. Close the TestProtocol1. 8. Open TestProtocol1 with BY_DRIVER | EXCLUSIVE. 9. Close the TestProtocol1. 10. Call OpenProtocolInformation() again. The return *EntryCount* should be 0.

Number	GUID	Assertion	Test Description
5.1.3.10.43	0x3e749cd6, 0x0f4c, 0x49f0, 0xbc, 0xf8, 0x70, 0x66, 0xd9, 0xce, 0x08, 0x0b	**BS.OpenProtocolInformation – OpenProtocolInformation()** gets the open information at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto a handle. 2. Open **TestProtocol1** with **BY_HANDLE_PROTOCOL**, **GET_PROTOCOL**, TEST_PROTOCOL, and **BY_CHILD_CONTROLLER**. 3. Call **OpenProtocolInformation()** on the handle and **TestProtocol1** to retrieve the open information. 4. Open **TestProtocol1** with **BY_DRIVER**. 5. Close the **TestProtocol1**. 6. Open **TestProtocol1** with **EXCLUSIVE**. 7. Close the **TestProtocol1**. 8. Open **TestProtocol1** with **BY_DRIVER | EXCLUSIVE**. 9. Close the **TestProtocol1**. 10. Call **OpenProtocolInformation()** again. The return *EntryCount* should be 0.

3.3.11 ConnectController()

Number	GUID	Assertion	Test Description
5.1.3.11.1	0x5062ba7f, 0x98f8, 0x42dd, 0x98, 0x4e, 0xa3, 0xcf, 0xe7, 0x4c, 0x7a, 0x74	**BS.ConnectController – ConnectController()** returns **EFI_INVALID_PARAMETER** with invalid *ControllerHandle*	1. Call **ConnectController()** with invalid *ControllerHandle*. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.11.2	0xd2a2f8db, 0x08bc, 0x4c02, 0x87, 0x8b, 0x89, 0x02, 0xd8, 0xf0, 0x24, 0x01	**BS.ConnectController – ConnectController()** returns **EFI_NOT_FOUND** with related driver.	1. Call **InstallProtocolInterface()** to create a new handle attached with a new protocol defined by the test case. 2. Call **ConnectController()** to attempt to connect the new handle with any driver exist in current system. The return code should be **EFI_NOT_FOUND**.
5.1.3.11.3	0x90263ddb, 0x043b, 0x480a, 0x9b, 0xb4, 0x1d, 0xbb, 0x45, 0x12, 0xe0, 0x95	**BS.ConnectController – ConnectController()** returns **EFI_SUCCESS** with **NULL** driver handle and End device path at **EFI_TPL_APPLICATION**.	1. Call **ConnectController()** with a *DriverImageHandle* value of **NULL**, and a *RemainingDevicePath* value of End device path node. The return code should be **EFI_SUCCESS**.
5.1.3.11.4	0x9e334c55, 0x2d9d, 0x4c6f, 0x82, 0xed, 0x67, 0xf0, 0x68, 0x2c, 0x43, 0x79	**BS.ConnectController – ConnectController()** returns **EFI_SUCCESS** with **NULL** driver handle and End device path at **EFI_TPL_CALLBACK**.	1. Call **ConnectController()** with a *DriverImageHandle* value of **NULL**, and a *RemainingDevicePath* value of End device path node. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.11.5	0xbf4441cf, 0x401d, 0x45ed, 0xa1, 0xa9, 0xa8, 0x88, 0x80, 0x6c, 0xd8, 0x92	BS.ConnectController – ConnectController() returns EFI_SUCCESS with NULL driver handle and End device path at EFI_TPL_NOTIFY.	1. Call ConnectController() with a *DriverImageHandle* value of NULL, and a *RemainingDevicePath* value of End device path node. The return code should be EFI_SUCCESS.
5.1.3.11.6	0x3ccb67c9, 0xd8b1, 0x44e6, 0x8c, 0x47, 0x4a, 0x79, 0xe8, 0x12, 0x17, 0xe2	BS.ConnectController – ConnectController() returns EFI_SUCCESS with driver handle at EFI_TPL_APPLICATION.	1. Create a test driver to consume TestProtocol1 and install TestProtocol2. 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() with this handle and the test driver. The return code should be EFI_SUCCESS.
5.1.3.11.7	0x390d6e25, 0xf39a, 0x40d7, 0xb1, 0xdd, 0x7e, 0xcf, 0x00, 0xf6, 0xbe, 0x43	BS.ConnectController – ConnectController() returns EFI_SUCCESS with driver handle at EFI_TPL_CALLBACK.	1. Create a test driver to consume TestProtocol1 and install TestProtocol2. 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() with this handle and the test driver. The return code should be EFI_SUCCESS.
5.1.3.11.8	0x08b89696, 0xae6b, 0x4a9c, 0xa5, 0xfb, 0x8d, 0x95, 0x1f, 0x01, 0x8b, 0x08	BS.ConnectController – ConnectController() returns EFI_SUCCESS with driver handle at EFI_TPL_NOTIFY.	1. Create a test driver to consume TestProtocol1 and install TestProtocol2. 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() with this handle and the test driver. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.11.9	0x14ac9b54, 0xe7c7, 0x4858, 0x86, 0x69, 0x33, 0x23, 0x88, 0xf1, 0x66, 0xf9	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with driver handle at `EFI_TPL_APPLICATION`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2`. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver. The `TestProtocol2` should be located.
5.1.3.11.10	0x3da1683e, 0x49f1, 0x4c2f, 0x82, 0xc3, 0x84, 0x40, 0xb6, 0x73, 0xac, 0xbb	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with driver handle at `EFI_TPL_CALLBACK`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2`. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver. The `TestProtocol2` should be located.
5.1.3.11.11	0x13e0da6e, 0xe60f, 0x4bba, 0xbc, 0xb8, 0x6b, 0xe0, 0x2f, 0xd6, 0xec, 0xb5	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with driver handle at `EFI_TPL_NOTIFY`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2`. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver. The `TestProtocol2` should be located.

Number	GUID	Assertion	Test Description
5.1.3.11.12	0xed970fb7, 0xb2a8, 0x41e9, 0x95, 0xc7, 0x78, 0xe6, 0x29, 0x0e, 0x8d, 0xf1	BS.ConnectController – ConnectController() returns EFI_SUCCESS with bus driver handle at EFI_TPL_APPLICATION.	1. Create a test driver to consume TestProtocol1 and install TestProtocol2 onto 10 child handles. 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() with this handle and the test driver. The return code should be EFI_SUCCESS.
5.1.3.11.13	0x8abcac46, 0xe840, 0x496a, 0x8a, 0x8c, 0xa6, 0xc4, 0x80, 0x2a, 0x4f, 0x9f	BS.ConnectController – ConnectController() returns EFI_SUCCESS with bus driver handle at EFI_TPL_CALLBACK.	1. Create a test driver to consume TestProtocol1 and install TestProtocol2 onto 10 child handles. 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() with this handle and the test driver. The return code should be EFI_SUCCESS.
5.1.3.11.14	0xdc039a94, 0x58da, 0x4794, 0x87, 0xae, 0x8f, 0xb4, 0x9a, 0x50, 0xd6, 0xf8	BS.ConnectController – ConnectController() returns EFI_SUCCESS with bus driver handle at EFI_TPL_NOTIFY.	1. Create a test driver to consume TestProtocol1 and install TestProtocol2 onto 10 child handles. 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() with this handle and the test driver. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.11.15	0xd1ccb8e6, 0x0b71, 0x4369, 0x82, 0xec, 0x88, 0x20, 0x9e, 0x63, 0xec, 0x4c	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle at `EFI_TPL_APPLICATION`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto 10 child handles. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver. `TestProtocol2` should be located.
5.1.3.11.16	0x4fa1cf88, 0xd6b6, 0x48ed, 0xb8, 0x89, 0xaa, 0x11, 0x47, 0xb3, 0xc0, 0x8b	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle at `EFI_TPL_CALLBACK`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto 10 child handles. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver. `TestProtocol2` should be located.
5.1.3.11.17	0x62e2a15a, 0xd00b, 0x43b1, 0x92, 0x28, 0x06, 0xe0, 0x19, 0x23, 0xe7, 0x22	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle at `EFI_TPL_NOTIFY`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto 10 child handles. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver. `TestProtocol2` should be located.

Number	GUID	Assertion	Test Description
5.1.3.11.18	0x2b2076c7, 0x6555, 0x473c, 0xbd, 0xa3, 0xe6, 0xfe, 0x2e, 0x62, 0x23, 0x8e	**BS.ConnectController – ConnectController()** returns **EFI_SUCCESS** with bus driver handle at **EFI_TPL_APPLICATION**.	1. Create a test driver to consume **TestProtocol1** and install **TestProtocol2** onto 10 child handles. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** with this handle and the test driver. The count of **TestProtocol2** should be 10.
5.1.3.11.19	0xfbf6e1e7, 0x915a, 0x450c, 0x8f, 0x89, 0x4f, 0xc3, 0x28, 0x71, 0x1f, 0xf7	**BS.ConnectController – ConnectController()** returns **EFI_SUCCESS** with bus driver handle at **EFI_TPL_CALLBACK**.	1. Create a test driver to consume **TestProtocol1** and install **TestProtocol2** onto 10 child handles. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** with this handle and the test driver. The count of **TestProtocol2** should be 10.
5.1.3.11.20	0xd29d9db1, 0x8433, 0x43b5, 0x83, 0x53, 0xb2, 0xb6, 0x43, 0x28, 0x12, 0x69	**BS.ConnectController – ConnectController()** returns **EFI_SUCCESS** with bus driver handle at **EFI_TPL_NOTIFY**.	1. Create a test driver to consume **TestProtocol1** and install **TestProtocol2** onto 10 child handles. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** with this handle and the test driver. The count of **TestProtocol2** should be 10.

Number	GUID	Assertion	Test Description
5.1.3.11.21	0x93f764f7, 0x890c, 0x4939, 0xb7, 0x5b, 0xc2, 0x2a, 0x0b, 0x60, 0x15, 0xbf	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle and device path at `EFI_TPL_APPLICATION`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto 10 child handles based on different device path nodes. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver, and the specified device path. The device path should be located in the test driver.
5.1.3.11.22	0x98c2f02b, 0x0875, 0x4b69, 0xb9, 0xb8, 0xa8, 0x58, 0xfe, 0xd9, 0x28, 0xf7	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle and device path at `EFI_TPL_CALLBACK`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto 10 child handles based on different device path nodes. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver, and the specified device path. The device path should be located in the test driver.
5.1.3.11.23	0xf36c7d9b, 0x12ea, 0x4dc1, 0xad, 0xba, 0x25, 0x06, 0xb7, 0xe5, 0x39, 0x57	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle and device path at `EFI_TPL_NOTIFY`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto 10 child handles based on different device path nodes. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver, and the specified device path. The device path should be located in the test driver.

Number	GUID	Assertion	Test Description
5.1.3.11.24	0x4638f45f, 0x707c, 0x4cd5, 0x80, 0xcd, 0x9d, 0xf0, 0xeb, 0xdc, 0xc3, 0x4a	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle and device path at `EFI_TPL_APPLICATION`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto 10 child handles based on different device path nodes. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver, and the specified device path. The remaining device path node is the same as the input.
5.1.3.11.25	0xe9cc5de6, 0x3847, 0x4af8, 0xa9, 0x41, 0x39, 0x39, 0xc9, 0x30, 0x85, 0x12	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle and device path at `EFI_TPL_CALLBACK`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto 10 child handles based on different device path nodes. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver, and the specified device path. The remaining device path node is the same as the input.
5.1.3.11.26	0xfa25dafa, 0xf36b, 0x45f6, 0x88, 0x59, 0x85, 0xd8, 0x8e, 0xde, 0x10, 0x6b	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle and device path at `EFI_TPL_NOTIFY`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto 10 child handles based on different device path nodes. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` with this handle and the test driver, and the specified device path. The remaining device path node is the same as the input.

Number	GUID	Assertion	Test Description
5.1.3.11.27	0x08eda2de, 0xcd07, 0x42b6, 0x85, 0xcb, 0x68, 0x75, 0x69, 0x5e, 0xee, 0x61	**BS.ConnectController – ConnectController()** returns **EFI_SUCCESS** with bus driver handle non-recursively at **EFI_TPL_APPLICATION**.	1. Create a test driver to consume **TestProtocol1** and install **TestProtocol2** onto a child handle. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** to connect them with non-recursively. The return code should be **EFI_SUCCESS**.
5.1.3.11.28	0x8b053397, 0x4ef1, 0x44b6, 0xb5, 0x06, 0xff, 0x31, 0xc1, 0x29, 0x7a, 0xc5	**BS.ConnectController – ConnectController()** returns **EFI_SUCCESS** with bus driver handle non-recursively at **EFI_TPL_CALLBACK**.	1. Create a test driver to consume **TestProtocol1** and install **TestProtocol2** onto a child handle. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** to connect them with non-recursively. The return code should be **EFI_SUCCESS**.
5.1.3.11.29	0xe76ab343, 0x1c15, 0x4464, 0xa9, 0xae, 0x15, 0x19, 0x1f, 0x54, 0x20, 0x6b	**BS.ConnectController – ConnectController()** returns **EFI_SUCCESS** with bus driver handle non-recursively at **EFI_TPL_NOTIFY**.	1. Create a test driver to consume **TestProtocol1** and install **TestProtocol2** onto a child handle. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** to connect them with non-recursively. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.11.30	0x797d540f, 0x0b07, 0x40c2, 0x9a, 0x92, 0xdb, 0xe8, 0xae, 0x42, 0xaa, 0xa7	**BS.ConnectController – ConnectController()** returns **EFI_SUCCESS** with bus driver handle non-recursively at **EFI_TPL_APPLICATION**.	1. Create a test driver to consume **TestProtocol1** and install **TestProtocol2** onto a child handle. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** to connect them with non-recursively. **TestProtocol2** should be located.
5.1.3.11.31	0xe9083c7c, 0x0ec6, 0x4d4e, 0x82, 0xaa, 0x37, 0xc7, 0x15, 0xf0, 0x1b, 0x2b	**BS.ConnectController – ConnectController()** returns **EFI_SUCCESS** with bus driver handle non-recursively at **EFI_TPL_CALLBACK**.	1. Create a test driver to consume **TestProtocol1** and install **TestProtocol2** onto a child handle. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** to connect them with non-recursively. **TestProtocol2** should be located.
5.1.3.11.32	0x2661fc3b, 0x060e, 0x459b, 0xb6, 0x9e, 0x9a, 0xbd, 0xf3, 0x8d, 0x18, 0x78	**BS.ConnectController – ConnectController()** returns **EFI_SUCCESS** with bus driver handle non-recursively at **EFI_TPL_NOTIFY**.	1. Create a test driver to consume **TestProtocol1** and install **TestProtocol2** onto a child handle. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** to connect them with non-recursively. **TestProtocol2** should be located.

Number	GUID	Assertion	Test Description
5.1.3.11.33	0xff8e9b83, 0x3056, 0x4460, 0xaf, 0xcf, 0x00, 0xea, 0x49, 0x7f, 0x3b, 0x88	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle non-recursively at `EFI_TPL_APPLICATION`.	1. Create a test driver1 to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a test driver2 to associate with the child handle created by test driver1, and install `TestProtocol3` on the handle. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect them with non-recursively. `TestProtocol3` should not be located.
5.1.3.11.34	0x3dab87dd, 0x3300, 0x4bd1, 0xbe, 0x7d, 0x8a, 0xbc, 0x7f, 0x2d, 0x7c, 0xec	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle non-recursively at `EFI_TPL_CALLBACK`.	1. Create a test driver1 to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a test driver2 to associate with the child handle created by test driver1, and install `TestProtocol3` on the handle. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect them with non-recursively. `TestProtocol3` should not be located.

Number	GUID	Assertion	Test Description
5.1.3.11.35	0x05746bbf, 0x24ec, 0x4a9b, 0x87, 0xf8, 0xc1, 0xe1, 0xa3, 0x59, 0x9a, 0x85	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle non-recursively at `EFI_TPL_NOTIFY`.	1. Create a test driver1 to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a test driver2 to associate with the child handle created by test driver1, and install `TestProtocol3` on the handle. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect them with non-recursively. `TestProtocol3` should not be located.
5.1.3.11.36	0xe5ac854a, 0xed36, 0x4a52, 0x8b, 0xf5, 0xa2, 0xcf, 0x38, 0x72, 0x87, 0xef	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle recursively at `EFI_TPL_APPLICATION`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect them with recursively. The return code should be `EFI_SUCCESS`.
5.1.3.11.37	0xff98ccd3, 0xabd4, 0x40f5, 0xa8, 0x61, 0xba, 0xaf, 0x44, 0x1b, 0x85, 0x16	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle recursively at `EFI_TPL_CALLBACK`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect them with recursively. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.11.38	0x8e783e67, 0x9591, 0x4a2b, 0x92, 0x1c, 0x88, 0xf5, 0x01, 0x57, 0x6f, 0x60	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle recursively at `EFI_TPL_NOTIFY`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect them with recursively. The return code should be `EFI_SUCCESS`.
5.1.3.11.39	0xac33fc14, 0x5103, 0x4f74, 0x9e, 0x45, 0xe5, 0x2e, 0xa2, 0x34, 0xa6, 0x05	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle recursively at `EFI_TPL_APPLICATION`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect them with recursively. `TestProtocol2` should be located.
5.1.3.11.40	0x6c322336, 0xa1c9, 0x44a5, 0xbd, 0xe7, 0x28, 0x4b, 0xb8, 0x0e, 0xb3, 0x9c	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle recursively at `EFI_TPL_CALLBACK`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect them with recursively. `TestProtocol2` should be located.

Number	GUID	Assertion	Test Description
5.1.3.11.41	0xcddb22e1, 0x257e, 0x46a8, 0x97, 0xb2, 0xcc, 0x42, 0x24, 0x7b, 0x95, 0x27	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle recursively at `EFI_TPL_NOTIFY`.	1. Create a test driver to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect them with recursively. `TestProtocol2` should be located.
5.1.3.11.42	0xde796be2, 0xa687, 0x4853, 0xb8, 0x23, 0xd4, 0x6f, 0x45, 0x04, 0xb5, 0xf2	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle recursively at `EFI_TPL_APPLICATION`.	1. Create a test driver1 to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a test driver2 to associate with the child handle created by test driver1, and install `TestProtocol3` on the handle. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect them with recursively. `TestProtocol3` should be located.

Number	GUID	Assertion	Test Description
5.1.3.11.43	0xbf767b24, 0x2947, 0x4be2, 0x94, 0xd2, 0x19, 0x00, 0x2b, 0x43, 0x4c, 0x55	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle recursively at `EFI_TPL_CALLBACK`.	1. Create a test driver1 to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a test driver2 to associate with the child handle created by test driver1, and install `TestProtocol3` on the handle. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect them with recursively. `TestProtocol3` should be located.
5.1.3.11.44	0x7f316b06, 0xe1ee, 0x47da, 0xb6, 0x67, 0x3b, 0xc4, 0xc9, 0x10, 0x3c, 0xd7	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with bus driver handle recursively at `EFI_TPL_NOTIFY`.	1. Create a test driver1 to consume `TestProtocol1` and install `TestProtocol2` onto a child handle. 2. Create a test driver2 to associate with the child handle created by test driver1, and install `TestProtocol3` on the handle. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect them with recursively. `TestProtocol3` should be located.

Number	GUID	Assertion	Test Description
5.1.3.11.45	0x917ecceb, 0x5338, 0x4d26, 0xbf, 0x7e, 0x59, 0xee, 0xc8, 0x28, 0x05, 0x28	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with multiple drivers at `EFI_TPL_APPLICATION`.	1. Create three test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol2` and install `TestProtocol3`, and the last one consume `TestProtocol3` and install `TestProtocol4`. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle with 3 test drivers. The return code should be `EFI_SUCCESS`.
5.1.3.11.46	0x51c7c310, 0xde21, 0x4de3, 0xb7, 0x42, 0x58, 0x72, 0x7c, 0x0b, 0x56, 0x04	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with multiple drivers at `EFI_TPL_CALLBACK`.	1. Create three test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol2` and install `TestProtocol3`, and the last one consume `TestProtocol3` and install `TestProtocol4`. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle with 3 test drivers. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.11.47	0xf5b2a58b, 0x2066, 0x457b, 0xbf, 0x12, 0xaf, 0x16, 0xc9, 0x67, 0xf4, 0xbd	BS.ConnectController – ConnectController() returns EFI_SUCCESS with multiple drivers at EFI_TPL_NOTIFY.	1. Create three test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol2 and install TestProtocol3, and the last one consume TestProtocol3 and install TestProtocol4. 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() to connect the handle with 3 test drivers. The return code should be EFI_SUCCESS.
5.1.3.11.48	0x36fba4aa, 0xd674, 0x48ae, 0x80, 0x79, 0x00, 0xc4, 0x33, 0x03, 0x92, 0x79	BS.ConnectController – ConnectController() returns EFI_SUCCESS with multiple drivers at EFI_TPL_APPLICATION.	1. Create three test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol2 and install TestProtocol3, and the last one consume TestProtocol3 and install TestProtocol4. 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() to connect the handle with 3 test drivers. TestProtocol2 ~ 4 should be located.

Number	GUID	Assertion	Test Description
5.1.3.11.49	0x51ffd5da, 0x49d0, 0x40bf, 0xaf, 0xe9, 0x50, 0xaa, 0x2f, 0x08, 0x6a, 0xf8	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with multiple drivers at `EFI_TPL_CALLBACK`.	1. Create three test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol2` and install `TestProtocol3`, and the last one consume `TestProtocol3` and install `TestProtocol4`. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle with 3 test drivers. `TestProtocol2` ~ 4 should be located.
5.1.3.11.50	0xe3c583a5, 0xa3da, 0x4e4e, 0xaf, 0x5d, 0x65, 0xb6, 0x1b, 0x18, 0xe9, 0x11	`BS.ConnectController – ConnectController()` returns `EFI_SUCCESS` with multiple drivers at `EFI_TPL_NOTIFY`.	1. Create three test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol2` and install `TestProtocol3`, and the last one consume `TestProtocol3` and install `TestProtocol4`. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle with 3 test drivers. `TestProtocol2` ~ 4 should be located.

Number	GUID	Assertion	Test Description
5.1.3.11.51	0x1b08dc10, 0xc423, 0x4a3a, 0x84, 0x84, 0xf0, 0x73, 0x02, 0xf7, 0x12, 0x8b	**BS.ConnectController – ConnectController()** connects driver list in order at **EFI_TPL_APPLICATION**.	1. Create two test drivers, the first one consume **TestProtocol1** and install **TestProtocol2**, the second one consume **TestProtocol1** and install **TestProtocol3**. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** to connect the handle with 2 test drivers. The return code should be **EFI_SUCCESS**.
5.1.3.11.52	0x079ebac7, 0xcc02, 0x4472, 0x95, 0xc4, 0xc0, 0x5f, 0x10, 0x05, 0x5c, 0xc1	**BS.ConnectController – ConnectController()** connects driver list in order at **EFI_TPL_CALLBACK**.	1. Create two test drivers, the first one consume **TestProtocol1** and install **TestProtocol2**, the second one consume **TestProtocol1** and install **TestProtocol3**. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** to connect the handle with 2 test drivers. The return code should be **EFI_SUCCESS**.
5.1.3.11.53	0x44ab5c2d, 0x0898, 0x4ac9, 0xa0, 0x96, 0x7c, 0x91, 0x96, 0x74, 0xf9, 0xe4	**BS.ConnectController – ConnectController()** connects driver list in order at **EFI_TPL_NOTIFY**.	1. Create two test drivers, the first one consume **TestProtocol1** and install **TestProtocol2**, the second one consume **TestProtocol1** and install **TestProtocol3**. 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** to connect the handle with 2 test drivers. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.11.54	0xb5b557e9, 0x1023, 0x4110, 0xbd, 0x49, 0xeb, 0x9a, 0x2e, 0x58, 0x81, 0xd3	`BS.ConnectController – ConnectController()` connects driver list in order at `EFI_TPL_APPLICATION`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle with 2 test drivers. `TestProtocol2` should be located, `TestProtocol3` could not.
5.1.3.11.55	0x36fa3b30, 0x2aed, 0x4bae, 0xb6, 0x3c, 0x35, 0x34, 0xba, 0x88, 0x54, 0xc0	`BS.ConnectController – ConnectController()` connects driver list in order at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle with 2 test drivers. `TestProtocol2` should be located, `TestProtocol3` could not.

Number	GUID	Assertion	Test Description
5.1.3.11.56	0xe0b288b9, 0x2e75, 0x4314, 0x99, 0x56, 0xc3, 0xf8, 0xdf, 0x4f, 0x6b, 0x9e	BS.ConnectController – ConnectController() connects driver list in order at EFI_TPL_NOTIFY.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3. 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() to connect the handle with 2 test drivers. TestProtocol2 should be located, TestProtocol3 could not.
5.1.3.11.57	0x9528d695, 0xffd5, 0x4ec9, 0x9c, 0x23, 0x3c, 0x45, 0x1c, 0x81, 0x70, 0xa4	BS.ConnectController – ConnectController() connects driver list in order described in EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL at EFI_TPL_APPLICATION.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3. 2. Install a EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL and list the second driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.11.58	0x4e710111, 0x1f35, 0x41eb, 0x86, 0xc0, 0x09, 0x24, 0xd6, 0xc4, 0x4d, 0xdf	BS.ConnectController – ConnectController() connects driver list in order described in EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL at EFI_TPL_CALLBACK.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3. 2. Install a EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL and list the second driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle. The return code should be EFI_SUCCESS.
5.1.3.11.59	0x1216a391, 0xdd69, 0x4e1e, 0xa7, 0x95, 0x76, 0x90, 0xa4, 0x01, 0x59, 0x14	BS.ConnectController – ConnectController() connects driver list in order described in EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL at EFI_TPL_NOTIFY.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3. 2. Install a EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL and list the second driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.11.60	0x207a93c8, 0x9c2c, 0x496f, 0xad, 0x9f, 0xe9, 0xf7, 0x1c, 0xfd, 0xd4, 0xfd	BS.ConnectController – ConnectController() connects driver list in order described in EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL at EFI_TPL_APPLICATION.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3. 2. Install a EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL and list the second driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle. TestProtocol3 should be located, TestProtocol2 could not.
5.1.3.11.61	0x9ee6b3f3, 0xbe55, 0x465d, 0xad, 0xdb, 0xd5, 0x52, 0xc0, 0xd0, 0xff, 0x39	BS.ConnectController – ConnectController() connects driver list in order described in EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL at EFI_TPL_CALLBACK.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3. 2. Install a EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL and list the second driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle. TestProtocol3 should be located, TestProtocol2 could not.

Number	GUID	Assertion	Test Description
5.1.3.11.62	0x9ae537a6, 0xa090, 0x41d3, 0x8c, 0xe1, 0x3e, 0x7f, 0x07, 0x30, 0x21, 0x16	`BS.ConnectController – ConnectController()` connects driver list in order described in `EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL` at `EFI_TPL_NOTIFY`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install a `PLATFORM_DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle. `TestProtocol3` should be located, `TestProtocol2` could not.
5.1.3.11.63	0xe7408bd3, 0xfe38, 0x4298, 0x87, 0x8b, 0x9a, 0x46, 0x39, 0x3a, 0x3d, 0x39	`BS.ConnectController – ConnectController()` connects driver list in order described in `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` at `EFI_TPL_APPLICATION`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.11.64	0x8a1955bd, 0xe50e, 0x4c19, 0x85, 0x9d, 0xcc, 0x29, 0x13, 0xc0, 0x1c, 0x23	**BS.ConnectController – ConnectController()** connects driver list in order described in **EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL** at **EFI_TPL_CALLBACK**.	1. Create two test drivers, the first one consume **TestProtocol1** and install **TestProtocol2**, the second one consume **TestProtocol1** and install **TestProtocol3**. 2. Install a **EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL** and list the second driver first. 3. Create a new handle and install **TestProtocol1** on this handle. 4. Call **ConnectController()** to connect the handle. The return code should be **EFI_SUCCESS**.
5.1.3.11.65	0x9047b56d, 0x3169, 0x4f87, 0x88, 0x45, 0xf0, 0x65, 0x89, 0xbb, 0x62, 0xcb	**BS.ConnectController – ConnectController()** connects driver list in order of Bus Specific Driver Override at **EFI_TPL_NOTIFY**.	1. Create two test drivers, the first one consume **TestProtocol1** and install **TestProtocol2**, the second one consume **TestProtocol1** and install **TestProtocol3**. 2. Install an **EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL** and list the second driver first. 3. Create a new handle and install **TestProtocol1** on this handle. 4. Call **ConnectController()** to connect the handle. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.11.66	0xe50b3169, 0xbb9f, 0x45b1, 0xb0, 0xf3, 0x4c, 0x4f, 0xab, 0x88, 0xc9, 0x67	`BS.ConnectController – ConnectController()` connects driver list in order of Bus Specific Driver Override at `EFI_TPL_APPLICATION`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle. `TestProtocol3` should be located, `TestProtocol2` could not.
5.1.3.11.67	0xec307bd4, 0x904d, 0x4a0f, 0xbf, 0x74, 0x47, 0xf6, 0x87, 0x1e, 0x43, 0x5c	`BS.ConnectController – ConnectController()` connects driver list in order of Bus Specific Driver Override at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle. `TestProtocol3` should be located, `TestProtocol2` could not.

Number	GUID	Assertion	Test Description
5.1.3.11.68	0x507332b3, 0xe897, 0x421a, 0xa3, 0x62, 0xe9, 0x0a, 0x38, 0x18, 0xe2, 0x76	`BS.ConnectController – ConnectController()` connects driver list in order described in `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` at `EFI_TPL_NOTIFY`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle. `TestProtocol3` should be located, `TestProtocol2` could not.
5.1.3.11.69	0xdb605bb5, 0x0720, 0x4d47, 0xb4, 0x29, 0xde, 0xd1, 0xbe, 0xd5, 0x4a, 0x87	`BS.ConnectController – ConnectController()` connects driver list in order of Driver Binding Version at `EFI_TPL_APPLICATION`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`, and its Driver Binding Version is higher than the first one. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.11.70	0x53fa4d60, 0x6ab1, 0x418f, 0x8b, 0xdf, 0x50, 0x43, 0x90, 0xae, 0xd2, 0x9d	`BS.ConnectController – ConnectController()` connects driver list in order of Driver Binding Version at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`, and its Driver Binding Version is higher than the first one. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle. The return code should be `EFI_SUCCESS`.
5.1.3.11.71	0x4be4a695, 0xe6cd, 0x4b44, 0xb5, 0x73, 0x9a, 0x53, 0x0a, 0x6b, 0x57, 0xae	`BS.ConnectController – ConnectController()` connects driver list in order of Driver Binding Version at `EFI_TPL_NOTIFY`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`, and its Driver Binding Version is higher than the first one. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.11.72	0x2a342c0d, 0x32f9, 0x4380, 0xb5, 0x5d, 0x9f, 0x0b, 0xca, 0xd5, 0xc1, 0x44	`BS.ConnectController – ConnectController()` connects driver list in order of Driver Binding Version at `EFI_TPL_APPLICATION`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`, and its Driver Binding Version is higher than the first one. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle. `TestProtocol3` should be located, `TestProtocol2` could not.
5.1.3.11.73	0xd5831426, 0x6631, 0x46ca, 0x92, 0x72, 0x76, 0xca, 0x3d, 0xd7, 0x67, 0x3b	`BS.ConnectController – ConnectController()` connects driver list in order of Driver Binding Version at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`, and its Driver Binding Version is higher than the first one. 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle. `TestProtocol3` should be located, `TestProtocol2` could not.

Number	GUID	Assertion	Test Description
5.1.3.11.74	0x43c3a632, 0xeaea, 0x4ae2, 0x84, 0x88, 0x2e, 0x01, 0x94, 0x34, 0xd8, 0x28	BS.ConnectController – ConnectController() connects driver list in order of Driver Binding Version at EFI_TPL_NOTIFY.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3, and its Driver Binding Version is higher than the first one. 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() to connect the handle. TestProtocol3 should be located, TestProtocol2 could not.
5.1.3.11.75	0x2d951d03, 0xd6f6, 0x4ca3, 0x9b, 0xcd, 0x9f, 0x96, 0xb3, 0x3a, 0x65, 0x5b	BS.ConnectController – *Handle* list's priority is higher than Platform Driver Override at EFI_TPL_APPLICATION.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3. 2. Install an EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL and list the second driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and the two test drivers. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.11.76	0xc9f37982, 0x7df2, 0x4187, 0xa6, 0x6c, 0xf0, 0x94, 0x1c, 0xf7, 0x8b, 0x7f	`BS.ConnectController – Handle` list's priority is higher than Platform Driver Override at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL`and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. The return code should be `EFI_SUCCESS`.
5.1.3.11.77	0x3484123c, 0xb134, 0x4ff4, 0x81, 0x92, 0xba, 0xa3, 0x96, 0x84, 0xea, 0x45	`BS.ConnectController – Handle` list's priority is higher than Platform Driver Override at `EFI_TPL_NOTIFY`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.11.78	0x214b4f8a, 0x2d44, 0x4de0, 0xb1, 0x94, 0x93, 0xe0, 0xf3, 0x0f, 0xe6, 0x9a	`BS.ConnectController – Handle` list's priority is higher than Platform Driver Override at `EFI_TPL_APPLICATION`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL`and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. `TestProtocol2` should be located, `TestProtocol3` could not.
5.1.3.11.79	0xa99252b2, 0x9657, 0x45f7, 0x84, 0x53, 0xdd, 0x8c, 0x80, 0xaf, 0xd8, 0x71	`BS.ConnectController – Handle` list's priority is higher than Platform Driver Override at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. `TestProtocol2` should be located, `TestProtocol3` could not.

Number	GUID	Assertion	Test Description
5.1.3.11.80	0x0bf6828c, 0xb3f1, 0x460e, 0xa4, 0xd9, 0xd0, 0x73, 0xbd, 0x19, 0xd2, 0xcb	BS.ConnectController – Handle list's priority is higher than Platform Driver Override at EFI_TPL_NOTIFY.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3. 2. Install an EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOLand list the second driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and the two test drivers. TestProtocol2 should be located, TestProtocol3 could not.
5.1.3.11.81	0xf7ebadd8, 0x67bc, 0x4193, 0xbb, 0x10, 0x38, 0x46, 0xd5, 0x0b, 0x42, 0x15	BS.ConnectController – Handle list's priority is higher than Bus Specific Driver Override at EFI_TPL_APPLICATION.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3. 2. Install an EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL and list the second driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and the two test drivers. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.11.82	0x8726db63, 0x66d6, 0x490c, 0x8e, 0xc5, 0x78, 0x5f, 0xc7, 0x6d, 0xfa, 0xa5	`BS.ConnectController – Handle` list's priority is higher than Bus Specific Driver Override at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. The return code should be `EFI_SUCCESS`.
5.1.3.11.83	0xe29caa36, 0x8eef, 0x49ff, 0x9a, 0xd4, 0xff, 0x35, 0xbb, 0xa2, 0x48, 0xad	`BS.ConnectController – Handle` list's priority is higher than Bus Specific Driver Override at `EFI_TPL_NOTIFY`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.11.84	0x101b28c9, 0xe6a2, 0x4951, 0xa8, 0x83, 0x2e, 0xbf, 0xe0, 0x13, 0x30, 0xaf	`BS.ConnectController –` `Handle` list's priority is higher than Bus Specific Driver Override at `EFI_TPL_APPLICATION`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. `TestProtocol2` should be located, `TestProtocol3` could not.
5.1.3.11.85	0x81d10eed, 0xacb4, 0x4f1e, 0xa7, 0xff, 0x92, 0x4f, 0x16, 0xbc, 0x38, 0xe3	`BS.ConnectController –` `Handle` list's priority is higher than Bus Specific Driver Override at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`. 2. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. `TestProtocol2` should be located, `TestProtocol3` could not.

Number	GUID	Assertion	Test Description
5.1.3.11.86	0x161e8954, 0x9580, 0x4ef5, 0x93, 0x09, 0x32, 0xb3, 0x27, 0x85, 0x2e, 0x84	BS.ConnectController – Handle list's priority is higher than Bus Specific Driver Override at EFI_TPL_NOTIFY.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3. 2. Install an EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL and list the second driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and the two test drivers. TestProtocol2 should be located, TestProtocol3 could not.
5.1.3.11.87	0xd5a44649, 0xb901, 0x4c15, 0xbd, 0xef, 0xe6, 0x77, 0x17, 0x57, 0x76, 0xf6	BS.ConnectController – Platform Driver Override's priority is higher than Bus Specific Driver Override's at EFI_TPL_APPLICATION.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3 2. Install an EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL and list the second driver first. 3. Install an EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL and list the first driver first. 4. Create a new handle and install TestProtocol1 on this handle. 5. Call ConnectController() to connect the handle and the two test drivers. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.11.88	0x44ddbe59, 0xabfa, 0x4456, 0x8c, 0x76, 0xfd, 0x18, 0x4f, 0x65, 0xce, 0x6e	`BS.ConnectController` – Platform Driver Override's priority is higher than Bus Specific Driver Override's at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3` 2. Install an `EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the first driver first. 4. Create a new handle and install `TestProtocol1` on this handle. 5. Call `ConnectController()` to connect the handle and the two test drivers. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.11.89	0x09fd1f45, 0xa8f8, 0x45bd, 0xad, 0xa7, 0x35, 0xd1, 0x66, 0x9e, 0xf0, 0x99	**BS.ConnectController** – Platform Driver Override's priority is higher than Bus Specific Driver Override's at **EFI_TPL_NOTIFY**.	1. Create two test drivers, the first one consume **TestProtocol1** and install **TestProtocol2**, the second one consume **TestProtocol1** and install **TestProtocol3** 2. Install an **EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL** and list the second driver first. 3. Install an **EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL** and list the first driver first. 4. Create a new handle and install **TestProtocol1** on this handle. 5. Call **ConnectController()** to connect the handle and the two test drivers. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.11.90	0x4643e80e, 0xa6bf, 0x412c, 0xb4, 0xff, 0x96, 0x29, 0x28, 0x2b, 0xc8, 0x31	`BS.ConnectController –` Platform Driver Override's priority is higher than Bus Specific Driver Override's at `EFI_TPL_APPLICATION`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3` 2. Install an `EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL` and list the second driver first. 3. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the first driver first. 4. Create a new handle and install `TestProtocol1` on this handle. 5. Call `ConnectController()` to connect the handle and the two test drivers. `TestProtocol3` should be located, `TestProtocol4` could not.

Number	GUID	Assertion	Test Description
5.1.3.11.91	0x25cffdf5, 0xd252, 0x4515, 0xaf, 0x8f, 0xd8, 0xdb, 0x68, 0xf0, 0x22, 0xc3	`BS.ConnectController` – Platform Driver Override's priority is higher than Bus Specific Driver Override's at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3` 2. Install an `EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL`and list the second driver first. 3. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the first driver first. 4. Create a new handle and install `TestProtocol1` on this handle. 5. Call `ConnectController()` to connect the handle and the two test drivers. `TestProtocol3` should be located, `TestProtocol4` could not.

Number	GUID	Assertion	Test Description
5.1.3.11.92	0x555913e8, 0xba56, 0x4c68, 0x80, 0xb5, 0xa9, 0x6b, 0x8a, 0x3a, 0xfc, 0xb1	**BS.ConnectController –** Platform Driver Override's priority is higher than Bus Specific Driver Override's at **EFI_TPL_NOTIFY**.	1. Create two test drivers, the first one consume **TestProtocol1** and install **TestProtocol2**, the second one consume **TestProtocol1** and install **TestProtocol3** 2. Install an **EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL** and list the second driver first. 3. Install an **EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL** and list the first driver first. 4. Create a new handle and install **TestProtocol1** on this handle. 5. Call **ConnectController()** to connect the handle and the two test drivers. **TestProtocol3** should be located, **TestProtocol4** could not.

Number	GUID	Assertion	Test Description
5.1.3.11.93	0x5576dfdf, 0x4303, 0x41dc, 0xb4, 0xa5, 0xab, 0x49, 0xb8, 0x5e, 0x97, 0x5b	`BS.ConnectController` – Platform Driver Override's priority is higher than Driver Binding Version at `EFI_TPL_APPLICATION`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`, and its Driver Binding Version is higher than the first one. 2. Install an `EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL`and list the first driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. The return code should be `EFI_SUCCESS`.
5.1.3.11.94	0xc8facf42, 0x1aa4, 0x4507, 0x96, 0x6f, 0x7b, 0x5e, 0xd7, 0xc4, 0xd1, 0x0b	`BS.ConnectController` – Platform Driver Override's priority is higher than Driver Binding Version at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`, and its Driver Binding Version is higher than the first one. 2. Install an `EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL` and list the first driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.11.95	0xf9a48521, 0xede3, 0x4a39, 0xac, 0x5d, 0x22, 0x2c, 0x31, 0x53, 0xf5, 0x11	BS.ConnectController – Platform Driver Override's priority is higher than Driver Binding Version at EFI_TPL_NOTIFY.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3, and its Driver Binding Version is higher than the first one. 2. Install an EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL and list the first driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and the two test drivers. The return code should be EFI_SUCCESS.
5.1.3.11.96	0xdd1ab5c6, 0xf998, 0x4aae, 0x91, 0xde, 0x2d, 0xb7, 0x72, 0x9a, 0xa2, 0xc8	BS.ConnectController – Platform Driver Override's priority is higher than Driver Binding Version at EFI_TPL_APPLICATION.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3, and its Driver Binding Version is higher than the first one. 2. Install an EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL and list the first driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and the two test drivers. TestProtocol2 should be located, TestProtocol3 could not.

Number	GUID	Assertion	Test Description
5.1.3.11.97	0x75b1cb4e, 0x10b5, 0x4b97, 0x8b, 0xc7, 0xf5, 0x81, 0x6f, 0x7c, 0xcb, 0x58	`BS.ConnectController –` Platform Driver Override's priority is higher than Driver Binding Version at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`, and its Driver Binding Version is higher than the first one. 2. Install an `EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOL` and list the first driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. `TestProtocol2` should be located, `TestProtocol3` could not.

Number	GUID	Assertion	Test Description
5.1.3.11.98	0xdb926006, 0x2dda, 0x45f9, 0x95, 0xff, 0xf2, 0xd3, 0xc3, 0x64, 0x6a, 0x5c	BS.ConnectController – Platform Driver Override's priority is higher than Driver Binding Version at EFI_TPL_NOTIFY.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3, and its Driver Binding Version is higher than the first one. 2. Install an EFI_PLATFORM DRIVER_OVERRIDE_PROTOCOLand list the first driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and the two test drivers. TestProtocol2 should be located, TestProtocol3 could not.
5.1.3.11.99	0x5601264e, 0x2d2c, 0x4517, 0x8e, 0xa6, 0x69, 0x27, 0x3d, 0xd8, 0x07, 0x10	BS.ConnectController – Bus Specific Driver Override's priority is higher than Driver Binding Version at EFI_TPL_APPLICATION.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3, and its Driver Binding Version is higher than the first one. 2. Install an EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL and list the first driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and the two test drivers. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.11.100	0x6078602e, 0x4689, 0x4b00, 0x8e, 0xb6, 0xc0, 0x56, 0x0b, 0x6f, 0x8e, 0xee	BS.ConnectController – Bus Specific Driver Override's priority is higher than Driver Binding Version at EFI_TPL_CALLBACK.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3, and its Driver Binding Version is higher than the first one. 2. Install an EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL and list the first driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and the two test drivers. The return code should be EFI_SUCCESS.
5.1.3.11.101	0xa213d518, 0xade6, 0x4661, 0xa8, 0x27, 0x6a, 0x7f, 0x5a, 0xcf, 0x6b, 0x94	BS.ConnectController – Bus Specific Driver Override's priority is higher than Driver Binding Version at EFI_TPL_NOTIFY.	1. Create two test drivers, the first one consume TestProtocol1 and install TestProtocol2, the second one consume TestProtocol1 and install TestProtocol3, and its Driver Binding Version is higher than the first one. 2. Install an EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL and list the first driver first. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and the two test drivers. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.11.102	0x3f54452d, 0xe68c, 0x49ec, 0xae, 0x62, 0x9b, 0x89, 0x88, 0x94, 0xde, 0xe3	`BS.ConnectController –` Bus Specific Driver Override's priority is higher than Driver Binding Version at `EFI_TPL_APPLICATION`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`, and its Driver Binding Version is higher than the first one. 2. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the first driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. `TestProtocol2` should be located, `TestProtocol3` could not.
5.1.3.11.103	0x6a061cbc, 0x1f2a, 0x4ab1, 0x91, 0x74, 0x73, 0x86, 0x1c, 0xae, 0x54, 0x14	`BS.ConnectController –` Bus Specific Driver Override's priority is higher than Driver Binding Version at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`, and its Driver Binding Version is higher than the first one. 2. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the first driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. `TestProtocol2` should be located, `TestProtocol3` could not.

Number	GUID	Assertion	Test Description
5.1.3.11.104	0x497c37b8, 0x1371, 0x4b2c, 0xb9, 0x85, 0xd0, 0x99, 0x67, 0x6e, 0xa5, 0x79	`BS.ConnectController –` Bus Specific Driver Override's priority is higher than Driver Binding Version at `EFI_TPL_NOTIFY`.	1. Create two test drivers, the first one consume `TestProtocol1` and install `TestProtocol2`, the second one consume `TestProtocol1` and install `TestProtocol3`, and its Driver Binding Version is higher than the first one. 2. Install an `EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL` and list the first driver first. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and the two test drivers. `TestProtocol2` should be located, `TestProtocol3` could not.

3.3.12 DisconnectController()

Number	GUID	Assertion	Test Description
5.1.3.12.1	0x49160a12, 0x5137, 0x40ee, 0x8f, 0xca, 0x8f, 0x3e, 0x90, 0xe1, 0xd5, 0x24	`BS.DisconnectController – DisConnectController()` returns `EFI_INVALID_PARAMETER` with invalid *`ControllerHandle.`*	1. Call `DisConnectController()` with invalid *`ControllerHandle`*. The return code should be `EFI_INVALID_PARAMETER`.
5.1.3.12.2	0x90ab5fee, 0x4de2, 0x4136, 0x9b, 0x22, 0x34, 0x29, 0x3e, 0x60, 0x02, 0xde	`BS.DisconnectController – DisConnectController()` returns `EFI_INVALID_PARAMETER` with invalid driver image handle.	1. Call `DisConnectController()` with invalid `DriverImageHandle`. The return code should be `EFI_INVALID_PARAMETER`.
5.1.3.12.3	0x13f11092, 0xeb7f, 0x44b2, 0xba, 0x0f, 0x43, 0x19, 0x82, 0x3b, 0x63, 0xbd	`BS.DisconnectController – DisConnectController()` returns `EFI_INVALID_PARAMETER` with invalid child handle.	1. Call `DisConnectController()` with invalid *`ChildHandle`*. The return code should be `EFI_INVALID_PARAMETER`.
5.1.3.12.4	0x455218e4, 0xe706, 0x42c6, 0x83, 0x7e, 0xab, 0xd9, 0x19, 0x41, 0x86, 0x5a	`BS.DisconnectController – DisConnectController()` returns `EFI_SUCCESS` with `NULL` driver at `EFI_TPL_APPLICATION`.	1. Create a new handle and install `TestProtocol1` on this handle. 2. Call `DisConnectController()` with this handle and `NULL` driver. The return code should be `EFI_SUCCESS`.
5.1.3.12.5	0x740244c7, 0xb695, 0x48e5, 0x8e, 0x00, 0x03, 0xac, 0x0a, 0x06, 0x85, 0x54	`BS.DisconnectController – DisConnectController()` returns `EFI_SUCCESS` with `NULL` driver at `EFI_TPL_CALLBACK`.	1. Create a new handle and install `TestProtocol1` on this handle. 2. Call `DisConnectController()` with this handle and `NULL` driver. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.12.6	0x33154ee3, 0x75d0, 0x483e, 0xab, 0x48, 0x77, 0x92, 0x51, 0xf8, 0x36, 0xfd	**BS.DisconnectController – DisConnectController()** returns **EFI_SUCCESS** with **NULL** driver at **EFI_TPL_NOTIFY**.	1. Create a new handle and install **TestProtocol1** on this handle. 2. Call **DisConnectController()** with this handle and **NULL** driver. The return code should be **EFI_SUCCESS**.
5.1.3.12.7	0x60e90357, 0x8c2f, 0x46db, 0xa8, 0x50, 0xfd, 0x97, 0xd4, 0x47, 0x70, 0x90	**BS.DisconnectController – DisConnectController()** returns **EFI_SUCCESS** with unmanaged driver at **EFI_TPL_APPLICATION**.	1. Create a new handle and install **TestProtocol1** on this handle. 2. Call **DisConnectController()** with this handle and an unmanaged driver. The return code should be **EFI_SUCCESS**.
5.1.3.12.8	0xfafdc41c, 0x5454, 0x450d, 0xb6, 0x74, 0x36, 0x19, 0x61, 0x7f, 0x06, 0xc8	**BS.DisconnectController – DisConnectController()** returns **EFI_SUCCESS** with unmanaged driver at **EFI_TPL_CALLBACK**.	1. Create a new handle and install **TestProtocol1** on this handle. 2. Call **DisConnectController()** with this handle and an unmanaged driver. The return code should be **EFI_SUCCESS**.
5.1.3.12.9	0x2ffac82d, 0x3943, 0x4286, 0xa7, 0x7e, 0x51, 0xfb, 0xf3, 0xc9, 0xf8, 0x9a	**BS.DisconnectController – DisConnectController()** returns **EFI_SUCCESS** with unmanaged driver at **EFI_TPL_NOTIFY**.	1. Create a new handle and install **TestProtocol1** on this handle. 2. Call **DisConnectController()** with this handle and an unmanaged driver. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.12.10	0x0235bd32, 0x34a0, 0x4f33, 0x9b, 0x1c, 0x84, 0xd5, 0xbe, 0x61, 0x6c, 0x32	BS.DisconnectController – DisConnectController() returns EFI_SUCCESS with a managed driver at EFI_TPL_APPLICATION.	1. Create two test drivers, the first one consumes TestProtocol1 and installs TestProtocol2, the second one consumes TestProtocol2 and Installs TestProtocol3 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() to connect the handle and two test drivers. 4. Call DisConnectController() to disconnect the second driver. The return code should be EFI_SUCCESS.
5.1.3.12.11	0x727c405e, 0x1132, 0x4653, 0x89, 0x81, 0x49, 0x3a, 0x91, 0xe3, 0xe8, 0x42	BS.DisconnectController – DisConnectController() returns EFI_SUCCESS with a managed driver at EFI_TPL_CALLBACK.	1. Create two test drivers, the first one consumes TestProtocol1 and installs TestProtocol2, the second one consumes TestProtocol2 and Installs TestProtocol3 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() to connect the handle and two test drivers. 4. Call DisConnectController() to disconnect the second driver. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.12.12	0xd14c28ee, 0xb466, 0x43eb, 0x85, 0x01, 0x5f, 0x05, 0x85, 0xf1, 0x77, 0x3a	**BS.DisconnectController – DisConnectController()** returns **EFI_SUCCESS** with a managed driver at **EFI_TPL_NOTIFY**.	1. Create two test drivers, the first one consumes **TestProtocol1** and installs **TestProtocol2**, the second one consumes **TestProtocol2** and Installs **TestProtocol3** 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** to connect the handle and two test drivers. 4. Call **DisConnectController()** to disconnect the second driver. The return code should be **EFI_SUCCESS**.
5.1.3.12.13	0xc85a941b, 0x57cb, 0x42ee, 0xbb, 0x5d, 0xed, 0x1e, 0x21, 0x61, 0x9f, 0xca	**BS.DisconnectController – DisConnectController()** returns **EFI_SUCCESS** with multiple drivers at **EFI_TPL_APPLICATION**.	1. Create two test drivers, the first one consumes **TestProtocol1** and installs **TestProtocol2**, the second one consumes **TestProtocol2** and Installs **TestProtocol3** 2. Create a new handle and install **TestProtocol1** on this handle. 3. Call **ConnectController()** to connect the handle and two test drivers. 4. Call **DisConnectController()** to disconnect the handle and **NULL** driver. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.12.14	0x4894ad43, 0x77e5, 0x4f8d, 0x9f, 0x50, 0x3b, 0xc7, 0x53, 0x6d, 0xd0, 0x62	`BS.DisconnectController – DisConnectController()` returns `EFI_SUCCESS` with multiple drivers at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consumes `TestProtocol1` and installs `TestProtocol2`, the second one consumes `TestProtocol2` and Installs `TestProtocol3` 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle and two test drivers. 4. Call `DisConnectController()` to disconnect the handle and `NULL` driver. The return code should be `EFI_SUCCESS`.
5.1.3.12.15	0x6b66b89c, 0x3c58, 0x411b, 0xb8, 0xb5, 0x8d, 0x3e, 0xbe, 0x92, 0x37, 0x04	`BS.DisconnectController – DisConnectController()` returns `EFI_SUCCESS` with multiple drivers at `EFI_TPL_NOTIFY`.	1. Create two test drivers, the first one consumes `TestProtocol1` and installs `TestProtocol2`, the second one consumes `TestProtocol2` and Installs `TestProtocol3` 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle and two test drivers. 4. Call `DisConnectController()` to disconnect the handle and `NULL` driver. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.12.16	0x4aee7de8, 0x2350, 0x4072, 0x94, 0xc6, 0xd4, 0x42, 0xdb, 0xdd, 0x55, 0xc5	`BS.DisconnectController – DisConnectController()` returns `EFI_SUCCESS` with multiple drivers at `EFI_TPL_APPLICATION`.	1. Create two test drivers, the first one consumes `TestProtocol1` and installs `TestProtocol2`, the second one consumes `TestProtocol2` and Installs `TestProtocol3` 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle and two test drivers. 4. Call `DisConnectController()` to disconnect the handle and `NULL` driver. `TestProtocol2` ~ 3 should not be located.
5.1.3.12.17	0x5ce10b3a, 0x18ce, 0x4898, 0xae, 0x73, 0xbd, 0xca, 0xfc, 0xe2, 0x32, 0x5c	`BS.DisconnectController – DisConnectController()` returns `EFI_SUCCESS` with multiple drivers at `EFI_TPL_CALLBACK`.	1. Create two test drivers, the first one consumes `TestProtocol1` and installs `TestProtocol2`, the second one consumes `TestProtocol2` and Installs `TestProtocol3` 2. Create a new handle and install `TestProtocol1` on this handle. 3. Call `ConnectController()` to connect the handle and two test drivers. 4. Call `DisConnectController()` to disconnect the handle and `NULL` driver. `TestProtocol2` ~ 3 should not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.18	0x5b936fb6, 0x9ecb, 0x42e5, 0x95, 0x34, 0xcc, 0x98, 0x6e, 0xca, 0x0f, 0xaa	BS.DisconnectController – DisConnectController() returns EFI_SUCCESS with multiple drivers at EFI_TPL_NOTIFY.	1. Create two test drivers, the first one consumes TestProtocol1 and installs TestProtocol2, the second one consumes TestProtocol2 and Installs TestProtocol3 2. Create a new handle and install TestProtocol1 on this handle. 3. Call ConnectController() to connect the handle and two test drivers. 4. Call DisConnectController() to disconnect the handle and NULL driver. TestProtocol2 ~ 3 should not be located.
5.1.3.12.19	0x9311a4a0, 0xa493, 0x4451, 0xb2, 0xa1, 0x1b, 0x21, 0xef, 0x94, 0xd9, 0x11	BS.DisconnectController – DisConnectController() disconnects all child handles with Child is NULL at EFI_TPL_APPLICATION.	1. Create a test driver that consumes TestProtocol1 and installs TestProtocol2 and TestProtocol3 onto two new child handles. 2. Create two test drivers, the first one consumes TestProtocol2 and install TestProtocol4, the second one consumes TestProtocol3 and TestProtocol5. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and test driver. 5. Call DisConnectController() with Child is NULL. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.12.20	0x4fbd2f1d, 0xfeba, 0x4dc7, 0xb0, 0x30, 0x44, 0x5b, 0x13, 0xca, 0xc2, 0xaa	`BS.DisconnectController – DisConnectController()` disconnects all child handles with Child is `NULL` at `EFI_TPL_CALLBACK`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is `NULL`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.12.21	0xef305583, 0x6ed8, 0x4f3a, 0xa1, 0x43, 0x20, 0x28, 0x43, 0x9e, 0x91, 0x6a	BS.DisconnectController – DisConnectController() disconnects all child handles with Child is NULL at EFI_TPL_NOTIFY.	1. Create a test driver that consumes TestProtocol1 and installs TestProtocol2 and TestProtocol3 onto two new child handles. 2. Create two test drivers, the first one consumes TestProtocol2 and install TestProtocol4, the second one consumes TestProtocol3 and TestProtocol5. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and test driver. 5. Call DisConnectController() with Child is NULL. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.12.22	0xf196155e, 0x6d04, 0x47f8, 0xb4, 0x54, 0x89, 0xd6, 0xe7, 0x06, 0x73, 0xd2	`BS.DisconnectController – DisConnectController()` disconnects all child handles with Child is `NULL` at `EFI_TPL_APPLICATION`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is `NULL`. `TestProtocol2` ~ 5 should not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.23	0x66ce17bf, 0x834f, 0x4d17, 0xb6, 0xcf, 0x85, 0x05, 0xca, 0x01, 0xc0, 0xd8	`BS.DisconnectController – DisConnectController()` disconnects all child handles with Child is `NULL` at `EFI_TPL_CALLBACK`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is `NULL`. `TestProtocol2` ~ 5 should not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.24	0x90c42308, 0x4c75, 0x4716, 0x8e, 0xc6, 0x0f, 0x1e, 0x35, 0x8e, 0x51, 0xd9	`BS.DisconnectController – DisConnectController()` disconnects all child handles with Child is `NULL` at `EFI_TPL_NOTIFY`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and installs `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is `NULL`. `TestProtocol2` ~ 5 should not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.25	0x41ba209a, 0x9251, 0x4c6f, 0xb8, 0x56, 0x77, 0x15, 0x6d, 0x8f, 0x54, 0x29	`BS.DisconnectController – DisConnectController()` disconnects all child handles with Child is `NULL` at `EFI_TPL_APPLICATION`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is `NULL`. The bus driver should not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.26	0x3ebebd1a, 0xd252, 0x420c, 0xaa, 0xcf, 0x8e, 0x9c, 0x9c, 0xa0, 0x3a, 0x69	`BS.DisconnectController – DisConnectController()` disconnects all child handles with Child is `NULL` at `EFI_TPL_CALLBACK`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is `NULL`. The bus driver should not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.27	0x906f71a7, 0xfb1b, 0x4432, 0x94, 0x84, 0x81, 0xb7, 0x27, 0x06, 0xa5, 0x58	`BS.DisconnectController – DisConnectController()` disconnects all child handles with Child is `NULL` at `EFI_TPL_NOTIFY`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is `NULL`. The bus driver should not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.28	0x10ad8db1, 0x29c0, 0x4015, 0x9f, 0xee, 0xca, 0x53, 0x2d, 0x4d, 0xe1, 0x40	`BS.DisconnectController – DisConnectController()` disconnects related child handles with Child is not `NULL` at `EFI_TPL_APPLICATION`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is the first child. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.12.29	0xf9e8db68, 0xf1e4, 0x4705, 0xa3, 0xe1, 0xa2, 0xa6, 0x84, 0x02, 0x40, 0xad	`BS.DisconnectController – DisConnectController()` disconnects related child handles with Child is not `NULL` at `EFI_TPL_CALLBACK`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is the first child. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.12.30	0x1a42e2d7, 0xbdeb, 0x43ca, 0xb1, 0xc7, 0xff, 0x09, 0x00, 0xfd, 0x88, 0x5c	**BS.DisconnectController – DisConnectController()** disconnects related child handles with Child is not **NULL** at **EFI_TPL_NOTIFY**.	1. Create a test driver that consumes **TestProtocol1** and installs **TestProtocol2** and **TestProtocol3** onto two new child handles. 2. Create two test drivers, the first one consumes **TestProtocol2** and install **TestProtocol4**, the second one consumes **TestProtocol3** and **TestProtocol5**. 3. Create a new handle and install **TestProtocol1** on this handle. 4. Call **ConnectController()** to connect the handle and test driver. 5. Call **DisConnectController()** with Child is the first child. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.12.31	0x7119d125, 0xc346, 0x4c29, 0x88, 0x34, 0x97, 0x5a, 0xcd, 0x1b, 0x52, 0xca	**BS.DisconnectController – DisConnectController()** disconnects related child handles with Child is not **NULL** at **EFI_TPL_APPLICATION**.	1. Create a test driver that consumes **TestProtocol1** and installs **TestProtocol2** and **TestProtocol3** onto two new child handles. 2. Create two test drivers, the first one consumes **TestProtocol2** and install **TestProtocol4**, the second one consumes **TestProtocol3** and **TestProtocol5**. 3. Create a new handle and install **TestProtocol1** on this handle. 4. Call **ConnectController()** to connect the handle and test driver. 5. Call **DisConnectController()** with Child is the first child. **TestProtocol2** and **TestProtocol4** should not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.32	0xd95f9fc1, 0x0fcc, 0x4d42, 0xb9, 0x76, 0x81, 0x4a, 0xbd, 0x6c, 0x7a, 0x9b	**BS.DisconnectController – DisConnectController()** disconnects related child handles with Child is not **NULL** at **EFI_TPL_CALLBACK**.	1. Create a test driver that consumes **TestProtocol1** and installs **TestProtocol2** and **TestProtocol3** onto two new child handles. 2. Create two test drivers, the first one consumes **TestProtocol2** and install **TestProtocol4**, the second one consumes **TestProtocol3** and **TestProtocol5**. 3. Create a new handle and install **TestProtocol1** on this handle. 4. Call **ConnectController()** to connect the handle and test driver. 5. Call **DisConnectController()** with Child is the first child. **TestProtocol2** and **TestProtocol4** should not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.33	0x0800e672, 0xa39f, 0x46b6, 0x86, 0xe4, 0xf4, 0xf9, 0x7c, 0xf0, 0x6a, 0xc1	BS.DisconnectController – DisConnectController() disconnects related child handles with Child is not NULL at EFI_TPL_NOTIFY.	1. Create a test driver that consumes TestProtocol1 and installs TestProtocol2 and TestProtocol3 onto two new child handles. 2. Create two test drivers, the first one consumes TestProtocol2 and install TestProtocol4, the second one consumes TestProtocol3 and TestProtocol5. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and test driver. 5. Call DisConnectController() with Child is the first child. TestProtocol2 and TestProtocol4 could not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.34	0x96ef96af, 0x4baa, 0x4a76, 0x91, 0xb4, 0x9f, 0x7f, 0x4e, 0xec, 0xac, 0x44	`BS.DisconnectController – DisConnectController()` disconnects related child handles with Child is not `NULL` at `EFI_TPL_APPLICATION`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is the first child. The bus driver should be located.

Number	GUID	Assertion	Test Description
5.1.3.12.35	0x513580a5, 0xb1bc, 0x4855, 0x9d, 0xf6, 0xaa, 0x3b, 0xb5, 0x23, 0xf6, 0x7a	`BS.DisconnectController – DisConnectController()` disconnects related child handles with Child is not `NULL` at `EFI_TPL_CALLBACK`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is the first child. The bus driver should be located.

Number	GUID	Assertion	Test Description
5.1.3.12.36	0x98639028, 0xf0a4, 0x4a45, 0xb4, 0x23, 0x9c, 0x93, 0x37, 0x45, 0x99, 0x8f	**BS.DisconnectController – DisConnectController()** disconnects related child handles with Child is not **NULL** at **EFI_TPL_NOTIFY**.	1. Create a test driver that consumes **TestProtocol1** and installs **TestProtocol2** and **TestProtocol3** onto two new child handles. 2. Create two test drivers, the first one consumes **TestProtocol2** and install **TestProtocol4**, the second one consumes **TestProtocol3** and **TestProtocol5**. 3. Create a new handle and install **TestProtocol1** on this handle. 4. Call **ConnectController()** to connect the handle and test driver. 5. Call **DisConnectController()** with Child is the first child. The bus driver should be located.

Number	GUID	Assertion	Test Description
5.1.3.12.37	0xffb2826f, 0xf636, 0x4b4c, 0xac, 0xf3, 0x33, 0xa4, 0xb4, 0xeb, 0xcd, 0x54	BS.DisconnectController – DisConnectController() disconnects related child handles with Child is not NULL at EFI_TPL_APPLICATION.	1. Create a test driver that consumes TestProtocol1 and installs TestProtocol2 and TestProtocol3 onto two new child handles. 2. Create two test drivers, the first one consumes TestProtocol2 and install TestProtocol4, the second one consumes TestProtocol3 and TestProtocol5. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and test driver. 5. Call DisConnectController() with Child is the first child. 6. Call DisConnectController() with Child is the second child. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.12.38	0xc93237b5, 0x9662, 0x46cf, 0x89, 0x41, 0xcc, 0xf2, 0x30, 0xc7, 0x87, 0x05	`BS.DisconnectController – DisConnectController()` disconnects related child handles with Child is not `NULL` at `EFI_TPL_CALLBACK`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is the first child. 6. Call `DisConnectController()` with Child is the second child. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.12.39	0xa3b1c71b, 0xfae6, 0x4348, 0x85, 0x5e, 0x3a, 0x1b, 0xde, 0x6b, 0xd1, 0x0d	`BS.DisconnectController – DisConnectController()` disconnects related child handles with Child is not `NULL` at `EFI_TPL_NOTIFY`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is the first child. 6. Call `DisConnectController()` with Child is the second child. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.12.40	0x26ea5cb9, 0x6c10, 0x4671, 0xba, 0x04, 0xe3, 0x8a, 0x9d, 0x23, 0xc5, 0xcc	`BS.DisconnectController – DisConnectController()` disconnects related child handles with Child is not `NULL` at `EFI_TPL_APPLICATION`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is the first child. 6. Call `DisConnectController()` with Child is the second child. `TestProtocol2` ~ 5 could not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.41	0x80ec98e2, 0x0b2c, 0x4dbb, 0xa6, 0x2f, 0xe4, 0xcd, 0x3b, 0x2b, 0x83, 0x30	`BS.DisconnectController – DisConnectController()` disconnects related child handles with Child is not `NULL` at `EFI_TPL_CALLBACK`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is the first child. 6. Call `DisConnectController()` with Child is the second child. `TestProtocol2` ~ 5 could not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.42	0x8d444cd1, 0x4ee6, 0x45a8, 0x8d, 0xef, 0x18, 0x67, 0x51, 0x75, 0x22, 0xa7	`BS.DisconnectController – DisConnectController()` disconnects related child handles with Child is not `NULL` at `EFI_TPL_NOTIFY`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is the first child. 6. Call `DisConnectController()` with Child is the second child. `TestProtocol2` ~ 5 could not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.43	0x8cd9bfbf, 0x021f, 0x469f, 0xbc, 0xb3, 0x9a, 0xff, 0x5e, 0x90, 0x36, 0x4b	`BS.DisconnectController – DisConnectController()` disconnects related child handles with Child is not `NULL` at `EFI_TPL_APPLICATION`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is the first child. 6. Call `DisConnectController()` with Child is the second child. The bus driver could not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.44	0xc3f9ef08, 0xb346, 0x4c61, 0xa4, 0xc4, 0x6f, 0x31, 0x9c, 0xb0, 0xc0, 0xfc	`BS.DisconnectController – DisConnectController()` disconnects related child handles with Child is not `NULL` at `EFI_TPL_CALLBACK`.	1. Create a test driver that consumes `TestProtocol1` and installs `TestProtocol2` and `TestProtocol3` onto two new child handles. 2. Create two test drivers, the first one consumes `TestProtocol2` and install `TestProtocol4`, the second one consumes `TestProtocol3` and `TestProtocol5`. 3. Create a new handle and install `TestProtocol1` on this handle. 4. Call `ConnectController()` to connect the handle and test driver. 5. Call `DisConnectController()` with Child is the first child. 6. Call `DisConnectController()` with Child is the second child. The bus driver could not be located.

Number	GUID	Assertion	Test Description
5.1.3.12.45	0xd0b46a61, 0x8708, 0x447b, 0x8c, 0xb8, 0x38, 0x60, 0x6a, 0x13, 0x4a, 0x64	BS.DisconnectController – DisConnectController() disconnects related child handles with Child is not NULL at EFI_TPL_NOTIFY.	1. Create a test driver that consumes TestProtocol1 and installs TestProtocol2 and TestProtocol3 onto two new child handles. 2. Create two test drivers, the first one consumes TestProtocol2 and install TestProtocol4, the second one consumes TestProtocol3 and TestProtocol5. 3. Create a new handle and install TestProtocol1 on this handle. 4. Call ConnectController() to connect the handle and test driver. 5. Call DisConnectController() with Child is the first child. 6. Call DisConnectController() with Child is the second child. The bus driver could not be located.

3.3.13 ProtocolsPerHandle()

Number	GUID	Assertion	Test Description
5.1.3.13.1	0xbd6c7a67, 0x0398, 0x496c, 0x8e, 0x28, 0x9d, 0xf9, 0x73, 0xb6, 0x5d, 0x0b	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` returns `EFI_INVALID_PARAMETER` with invalid handle	1. Call `ProtocolsPerHandle()` with `NULL` handle or invalid handle. The return code should be `EFI_INVALID_PARAMETER`.
5.1.3.13.2	0xebd50604, 0x8586, 0x43d8, 0xb5, 0xc8, 0x5a, 0x93, 0xa8, 0x01, 0xd1, 0x7a	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` returns `EFI_INVALID_PARAMETER` with `NULL` protocol buffer	1. Call `ProtocolsPerHandle()` with `NULL` protocol buffer (type is EFI_GUID***). The return code should be `EFI_INVALID_PARAMETER`.
5.1.3.13.3	0x0b12494f, 0xd484, 0x4cb7, 0xa9, 0x9d, 0xaf, 0x20, 0x03, 0x3f, 0x2d, 0xec	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` returns `EFI_INVALID_PARAMETER` with `Buffer` count `NULL`	1. Call `ProtocolsPerHandle()` with pointer to buffer count value of `NULL`. The return code should be `EFI_INVALID_PARAMETER`.
5.1.3.13.4	0xfea682e9, 0x5bb0, 0x4309, 0xa5, 0xbd, 0x90, 0xae, 0x8a, 0x8c, 0xaf, 0x6e	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` returns `EFI_SUCCESS` with valid parameter at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` ~ `TestProtocol4` onto a new handle. 2. Call `ProtocolsPerHandle()` to retrieve protocol number and GUID array on the handle. The return code should be `EFI_SUCCESS`.
5.1.3.13.5	0xa9a8a9f5, 0x5b7d, 0x472e, 0xb1, 0xa0, 0xad, 0x80, 0x1d, 0x3a, 0xd2, 0x8a	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` returns `EFI_SUCCESS` with valid parameter at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` ~ `TestProtocol4` onto a new handle. 2. Call `ProtocolsPerHandle()` to retrieve protocol number and GUID array on the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.13.6	0xd7b10222, 0x8df7, 0x4746, 0xbb, 0x35, 0xb2, 0x4a, 0x0a, 0xd6, 0xbc, 0x70	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` returns `EFI_SUCCESS` with valid parameter at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` ~ `TestProtocol4` onto a new handle. 2. Call `ProtocolsPerHandle()` to retrieve protocol number and GUID array on the handle. The return code should be `EFI_SUCCESS`.
5.1.3.13.7	0x8f3ade4b, 0x242c, 0x4ed7, 0x8a, 0x9f, 0x30, 0x84, 0xf4, 0x6c, 0x8e, 0x73	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` gets all protocols on the handle at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` ~ `TestProtocol4` onto a new handle. 2. Call `ProtocolsPerHandle()` to retrieve protocol number and GUID array on the handle. `TestProtocol1` ~ `TestProtocol4` should be returned.
5.1.3.13.8	0x6460ddb3, 0x61f4, 0x4072, 0xbb, 0xe5, 0x7c, 0x2d, 0x3a, 0xee, 0x31, 0x7f	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` gets all protocols on the handle at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` ~ `TestProtocol4` onto a new handle. 2. Call `ProtocolsPerHandle()` to retrieve protocol number and GUID array on the handle. `TestProtocol1` ~ `TestProtocol4` should be returned.
5.1.3.13.9	0x05f7ae94, 0x9646, 0x43f0, 0xa5, 0x8b, 0x9c, 0x4e, 0x1c, 0x78, 0x3f, 0x43	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` gets all protocols on the handle at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` ~ `TestProtocol4` onto a new handle. 2. Call `ProtocolsPerHandle()` to retrieve protocol number and GUID array on the handle. `TestProtocol1` ~ `TestProtocol4` should be returned.

Number	GUID	Assertion	Test Description
5.1.3.13.10	0x995133c6, 0xda8e, 0x4aa4, 0x87, 0xeb, 0xf8, 0x2f, 0xe7, 0xd5, 0xd5, 0x03	BS.ProtocolsPerHandle – ProtocolsPerHandle() returns EFI_SUCCESS with valid parameter at EFI_TPL_APPLICATION	1. Install TestProtocol1 ~ TestProtocol4 onto a new handle. 2. Call ProtocolsPerHandle() to retrieve protocol number and GUID array on the handle. 3. install TestProtocol5 onto the new handle. 4. Call ProtocolsPerHandle() again. The return code should be EFI_SUCCESS.
5.1.3.13.11	0x4fd61cf7, 0xcab6, 0x4f67, 0x96, 0x0c, 0x56, 0x62, 0xa6, 0x90, 0x31, 0xaa	BS.ProtocolsPerHandle – ProtocolsPerHandle() returns EFI_SUCCESS with valid parameter at EFI_TPL_CALLBACK	1. Install TestProtocol1 ~ TestProtocol4 onto a new handle. 2. Call ProtocolsPerHandle() to retrieve protocol number and GUID array on the handle. 3. install TestProtocol5 onto the new handle. 4. Call ProtocolsPerHandle() again. The return code should be EFI_SUCCESS.
5.1.3.13.12	0x0001b457, 0x86f7, 0x4085, 0x8d, 0xb0, 0x2b, 0xfb, 0xad, 0xd8, 0x32, 0x08	BS.ProtocolsPerHandle – ProtocolsPerHandle() returns EFI_SUCCESS with valid parameter at EFI_TPL_NOTIFY	1. Install TestProtocol1 ~ TestProtocol4 onto a new handle. 2. Call ProtocolsPerHandle() to retrieve protocol number and GUID array on the handle. 3. install TestProtocol5 onto the new handle. 4. Call ProtocolsPerHandle() again. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.13.13	0xf69d5220, 0x5e30, 0x4ab9, 0x9d, 0x09, 0xc7, 0x50, 0x40, 0xf7, 0xbb, 0x36	BS.ProtocolsPerHandle – ProtocolsPerHandle() gets all protocols on the handle at EFI_TPL_APPLICATION	1. Install TestProtocol1 ~ TestProtocol4 onto a new handle. 2. Call ProtocolsPerHandle() to retrieve protocol number and GUID array on the handle. TestProtocol1 ~ TestProtocol5 should be returned.
5.1.3.13.14	0xfcfe375e, 0xa1ba, 0x4eaa, 0x87, 0x28, 0xaf, 0x44, 0xd5, 0xfa, 0xd3, 0x81	BS.ProtocolsPerHandle – ProtocolsPerHandle() gets all protocols on the handle at EFI_TPL_CALLBACK	1. Install TestProtocol1 ~ TestProtocol4 onto a new handle. 2. Call ProtocolsPerHandle() to retrieve protocol number and GUID array on the handle. TestProtocol1 ~ TestProtocol5 should be returned.
5.1.3.13.15	0x1d05c8b8, 0x7dae, 0x41eb, 0x87, 0x55, 0x10, 0x48, 0xfe, 0x1d, 0x49, 0xeb	BS.ProtocolsPerHandle – ProtocolsPerHandle() gets all protocols on the handle at EFI_TPL_NOTIFY	1. Install TestProtocol1 ~ TestProtocol4 onto a new handle. 2. Call ProtocolsPerHandle() to retrieve protocol number and GUID array on the handle. TestProtocol1 ~ TestProtocol5 should be returned.

Number	GUID	Assertion	Test Description
5.1.3.13.16	0x4f302ea9, 0xa047, 0x4448, 0x8b, 0xdd, 0xd1, 0x60, 0x23, 0x13, 0xa4, 0x40	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` returns `EFI_SUCCESS` with valid parameter at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` ~ `TestProtocol4` onto a new handle. 2. Call `ProtocolsPerHandle()` to retrieve protocol number and GUID array on the handle. 3. Install `TestProtocol5` onto the new handle. 4. Call `ProtocolsPerHandle()` again. 5. Uninstall `TestProtocol1` & `TestProtocol2` from the handle. 6. Call `ProtocolsPerHandle()` again. The return code should be `EFI_SUCCESS`.
5.1.3.13.17	0x24ea2098, 0x3fd2, 0x4012, 0x83, 0xe4, 0x6b, 0x65, 0xe9, 0x6d, 0xd9, 0xad	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` returns `EFI_SUCCESS` with valid parameter at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` ~ `TestProtocol4` onto a new handle. 2. Call `ProtocolsPerHandle()` to retrieve protocol number and GUID array on the handle. 3. Install `TestProtocol5` onto the new handle. 4. Call `ProtocolsPerHandle()` again. 5. Uninstall `TestProtocol1` & `TestProtocol2` from the handle. 6. Call `ProtocolsPerHandle()` again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.13.18	0xc0edf6f9, 0x3954, 0x47ea, 0x86, 0x08, 0x10, 0xb1, 0x05, 0x18, 0x50, 0xd3	BS.ProtocolsPerHandle – ProtocolsPerHandle() returns EFI_SUCCESS with valid parameter at EFI_TPL_NOTIFY	1. Install TestProtocol1 ~ TestProtocol4 onto a new handle. 2. Call ProtocolsPerHandle() to retrieve protocol number and GUID array on the handle. 3. Install TestProtocol5 onto the new handle. 4. Call ProtocolsPerHandle() again. 5. Uninstall TestProtocol1 & TestProtocol2 from the handle. 6. Call ProtocolsPerHandle() again. The return code should be EFI_SUCCESS.
5.1.3.13.19	0x4f460e70, 0xf979, 0x4ba9, 0x8b, 0x0b, 0xa4, 0x61, 0x2c, 0xc5, 0xe8, 0x6a	BS.ProtocolsPerHandle – ProtocolsPerHandle() gets all protocols on the handle at EFI_TPL_APPLICATION	1. Install TestProtocol1 ~ TestProtocol4 onto a new handle. 2. Call ProtocolsPerHandle() to retrieve protocol number and GUID array on the handle. 3. Install TestProtocol5 onto the new handle. 4. Call ProtocolsPerHandle() again. 5. Uninstall TestProtocol1 & TestProtocol2 from the handle. 6. Call ProtocolsPerHandle() again. TestProtocol3 ~ TestProtocol5 should be returned.

Number	GUID	Assertion	Test Description
5.1.3.13.20	0xe8638e2d, 0xa62c, 0x4566, 0xa4, 0xbb, 0xfe, 0x36, 0xb6, 0x33, 0xfe, 0x3e	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` gets all protocols on the handle at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` ~ `TestProtocol4` onto a new handle. 2. Call `ProtocolsPerHandle()` to retrieve protocol number and GUID array on the handle. 3. Install `TestProtocol5` onto the new handle. 4. Call `ProtocolsPerHandle()` again. 5. Uninstall `TestProtocol1` & `TestProtocol2` from the handle. 6. Call `ProtocolsPerHandle()` again. `TestProtocol3` ~ `TestProtocol5` should be returned.
5.1.3.13.21	0x0300f2e9, 0xaaaa, 0x4735, 0xb3, 0x83, 0xe9, 0xa7, 0x4a, 0x9e, 0xfb, 0x7f	`BS.ProtocolsPerHandle – ProtocolsPerHandle()` gets all protocols on the handle at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` ~ `TestProtocol4` onto a new handle. 2. Call `ProtocolsPerHandle()` to retrieve protocol number and GUID array on the handle. 3. Install `TestProtocol5` onto the new handle. 4. Call `ProtocolsPerHandle()` again. 5. Uninstall `TestProtocol1` & `TestProtocol2` from the handle. 6. Call `ProtocolsPerHandle()` again. `TestProtocol3` ~ `TestProtocol5` should be returned.

3.3.14 LocateHandleBuffer()

Number	GUID	Assertion	Test Description
5.1.3.14.1	0x4f70540a, 0xfa1e, 0x4f00, 0x9e, 0x07, 0xc9, 0xf8, 0x3c, 0xc4, 0x5a, 0xf5	**BS.LocateHandleBuffer – LocateHandleBuffer()** returns **EFI_INVALID_PARAMETER** with invalid sarch type	1. Call **LocateHandleBuffer()** with search type other than **AllHandles**, **ByRegisterNotify** and **ByProtocol**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.14.2	0xf77677d7, 0x8869, 0x453c, 0xae, 0x7f, 0xa7, 0x7d, 0x16, 0x97, 0xe9, 0xe2	**BS.LocateHandleBuffer – LocateHandleBuffer()** returns **EFI_NOT_FOUND** with never installed protocol	1. Call **LocateHandleBuffer()** to locate the handles for a never installed protocol. The return code should be **EFI_NOT_FOUND**.
5.1.3.14.3	0xf5b84647, 0xbee8, 0x46ff, 0xaf, 0xb3, 0xb3, 0xd5, 0xd5, 0xa0, 0x08, 0x38	**BS.LocateHandleBuffer – LocateHandleBuffer()** returns **EFI_INVALID_PARAMETER** with **Buffer** is **NULL** or NoHandles is **NULL**	1. Call **LocateHandleBuffer()** to locate all handles with **Buffer** is **NULL** or NoHandles is **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.1.3.14.4	0x2e9a3ce0, 0x779a, 0x4bba, 0xaa, 0x6d, 0xe5, 0xa3, 0x77, 0x89, 0x85, 0xba	**BS.LocateHandleBuffer – LocateHandleBuffer()** returns **EFI_SUCCESS** with a **Type** value of **AllHandles** at **EFI_TPL_APPLICATION**	1. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system. The return code should be **EFI_SUCCESS**.
5.1.3.14.5	0x8dd43d2b, 0xed7b, 0x4f6a, 0x9a, 0xf6, 0x16, 0x2f, 0x73, 0xc9, 0x84, 0x7b	**BS.LocateHandleBuffer – LocateHandleBuffer()** returns **EFI_SUCCESS** with a **Type** value of **AllHandles** at **EFI_TPL_CALLBACK**	1. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system. The return code should be **EFI_SUCCESS**.
5.1.3.14.6	0x3d54399c, 0x7989, 0x4ce0, 0x9d, 0xeb, 0x80, 0x78, 0x7a, 0xcc, 0xdf, 0x6b	**BS.LocateHandleBuffer – LocateHandleBuffer()** returns **EFI_SUCCESS** with a **Type** value of **AllHandles** at **EFI_TPL_NOTIFY**	1. Call **LocateHandle()** via search type **AllHandles** to retrieve all handles in the system. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.14.7	0x5e78fd28, 0x36ee, 0x4d8d, 0xb3, 0x21, 0x64, 0x06, 0xc9, 0x40, 0xc7, 0x50	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates all handles at `EFI_TPL_APPLICATION`	1. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. The return code should be `EFI_SUCCESS`.
5.1.3.14.8	0xcebea147, 0x8237, 0x4254, 0xb5, 0xec, 0xae, 0x42, 0x92, 0xbf, 0x7c, 0xe1	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates all handles at `EFI_TPL_CALLBACK`	1. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. The return code should be `EFI_SUCCESS`.
5.1.3.14.9	0xab575087, 0xdd21, 0x42fd, 0x8c, 0x66, 0x68, 0x7b, 0x7d, 0x81, 0x57, 0xa6	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates all handles at `EFI_TPL_NOTIFY`	1. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.14.10	0x18b8f641, 0x4c03, 0x4e17, 0x8b, 0x73, 0x27, 0xa5, 0x1b, 0x61, 0x29, 0x17	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates all handles at `EFI_TPL_APPLICATION`	1. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system again. The return code should be `EFI_SUCCESS`.
5.1.3.14.11	0xc22a5509, 0x92bb, 0x4dbd, 0x95, 0xaf, 0xde, 0xf0, 0xba, 0xe5, 0x27, 0x8d	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates all handles at `EFI_TPL_CALLBACK`	1. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system again. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.14.12	0xc929f6d1, 0xc810, 0x434e, 0xb2, 0x05, 0xfb, 0xf0, 0xee, 0x88, 0xe7, 0x3a	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates all handles at `EFI_TPL_NOTIFY`	1. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system again. The return code should be `EFI_SUCCESS`.
5.1.3.14.13	0x59988b38, 0x031f, 0x4405, 0x89, 0x41, 0x49, 0x33, 0x04, 0xbb, 0x3b, 0x11	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates all handles at `EFI_TPL_APPLICATION`	1. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system again. The number of handles of the system increases by 1.

Number	GUID	Assertion	Test Description
5.1.3.14.14	0xf82d253c, 0x7d51, 0x4efd, 0x90, 0x3d, 0xbb, 0x0b, 0x57, 0x34, 0xfe, 0xae	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates all handles at `EFI_TPL_CALLBACK`	1. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system again. The number of handles of the system increases by 1.
5.1.3.14.15	0x3d990f50, 0xf775, 0x46d6, 0xab, 0xba, 0xe0, 0x2e, 0x00, 0x8b, 0x58, 0x6d	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates all handles at `EFI_TPL_NOTIFY`	1. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system again. The number of handles of the system increases by 1.

Number	GUID	Assertion	Test Description
5.1.3.14.16	0x1a435f75, 0x3636, 0x423f, 0x8d, 0x9d, 0x13, 0x64, 0xc3, 0xbe, 0x2c, 0xce	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates all handles at **EFI_TPL_APPLICATION**	1. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handle. 3. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system again. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. The return code should be **EFI_SUCCESS**.
5.1.3.14.17	0xf882343e, 0x81e0, 0x4c36, 0x81, 0x3e, 0xd9, 0x19, 0xde, 0xe9, 0x9a, 0xb9	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates all handles at **EFI_TPL_CALLBACK**	1. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handle. 3. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system again. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.14.18	0x854ef303, 0xc627, 0x48c9, 0x80, 0x0a, 0xa3, 0xc6, 0x80, 0xb8, 0x65, 0xbb	BS.LocateHandleBuffer – LocateHandleBuffer() locates all handles at EFI_TPL_NOTIFY	1. Call LocateHandleBuffer() via search type AllHandles to retrieve all handles in the system. 2. Call InstallProtocolInterface() to install TestProtocol1 onto a new handle. 3. Call LocateHandleBuffer() via search type AllHandles to retrieve all handles in the system again. 4. Call UninstallProtocolInterface() to uninstall TestProtocol1. The return code should be EFI_SUCCESS.
5.1.3.14.19	0x36c035e2, 0x4ffc, 0x4144, 0x89, 0x5d, 0x67, 0x87, 0xe2, 0x8a, 0x47, 0x70	BS.LocateHandleBuffer – LocateHandleBuffer() locates all handles at EFI_TPL_APPLICATION	1. Call LocateHandleBuffer() via search type AllHandles to retrieve all handles in the system. 2. Call InstallProtocolInterface() to install TestProtocol1 onto a new handle. 3. Call LocateHandleBuffer() via search type AllHandles to retrieve all handles in the system again. 4. Call UninstallProtocolInterface() to uninstall TestProtocol1. 5. Call LocateHandleBuffer() via search type AllHandles to retrieve all handles in the system again. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.14.20	0x1771620b, 0x01ca, 0x4f40, 0xb5, 0x4a, 0x96, 0x84, 0xcb, 0xd5, 0x66, 0x99	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates all handles at **EFI_TPL_CALLBACK**	1. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handle. 3. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system again. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system again. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.14.21	0xb57efffb, 0xadc7, 0x4980, 0xb9, 0x09, 0xcb, 0x71, 0xb1, 0x57, 0x93, 0x77	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates all handles at **EFI_TPL_NOTIFY**	1. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handle. 3. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system again. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system again. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.14.22	0x5a2174e7, 0x5858, 0x4b24, 0xa5, 0x97, 0x3a, 0x85, 0x65, 0x59, 0xcc, 0x53	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates all handles at `EFI_TPL_APPLICATION`	1. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system again. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system again. The number of handles of the system decreases by 1.

Number	GUID	Assertion	Test Description
5.1.3.14.23	0x2ebaf385, 0xc0c9, 0x4ffd, 0x99, 0xe0, 0x3b, 0x62, 0xdc, 0xd8, 0x81, 0x0a	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates all handles at **EFI_TPL_CALLBACK**	1. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handle. 3. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system again. 4. Call **UninstallProtocolInterface()** to uninstall **TestProtocol1**. 5. Call **LocateHandleBuffer()** via search type **AllHandles** to retrieve all handles in the system again. The number of handles of the system decreases by 1.

Number	GUID	Assertion	Test Description
5.1.3.14.24	0xa4085bb8, 0xa805, 0x4015, 0x9a, 0x3e, 0x54, 0xe6, 0x0b, 0x79, 0x96, 0xef	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates all handles at `EFI_TPL_NOTIFY`	1. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handle. 3. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system again. 4. Call `UninstallProtocolInterface()` to uninstall `TestProtocol1`. 5. Call `LocateHandleBuffer()` via search type `AllHandles` to retrieve all handles in the system again. The number of handles of the system decreases by 1.
5.1.3.14.25	0x96ef51d8, 0x85d9, 0x4147, 0x91, 0x17, 0xe6, 0x7e, 0x40, 0xb2, 0x24, 0x5c	`BS.LocateHandleBuffer – LocateHandleBuffer()` returns `EFI_SUCCESS` with a `Type` value of `ByRegisterNotify` at `EFI_TPL_APPLICATION`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.14.26	0xaffa52a9, 0x70d8, 0x41c7, 0x86, 0x8c, 0xdb, 0x30, 0xae, 0xa6, 0x86, 0xd2	**BS.LocateHandleBuffer – LocateHandleBuffer()** returns **EFI_SUCCESS** with a **Type** value of **ByRegisterNotify** at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. The return code should be **EFI_SUCCESS**.
5.1.3.14.27	0x0e525b23, 0x9b6c, 0x4d66, 0xb0, 0xab, 0xbd, 0xf4, 0x1f, 0x57, 0xf6, 0x3a	**BS.LocateHandleBuffer – LocateHandleBuffer()** returns **EFI_SUCCESS** with a **Type** value of **ByRegisterNotify** at **EFI_TPL_NOTIFY**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. The return code should be **EFI_SUCCESS**.
5.1.3.14.28	0x9f8b22e2, 0x46b4, 0x49ee, 0x86, 0xb1, 0xe5, 0xb8, 0x77, 0x4b, 0x0f, 0x5e	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates the new register handle at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandleBuffer()** via search type **ByRegisterNotify** with the search key generated by previous **RegisterProtocolNotify**. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.14.29	0xf268e2c7, 0x3b59, 0x4592, 0x9f, 0x6a, 0x45, 0x52, 0x23, 0x8d, 0x56, 0x2c	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates the new register handle at `EFI_TPL_CALLBACK`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandleBuffer()` via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. The return code should be `EFI_SUCCESS`.
5.1.3.14.30	0xbdee4f25, 0x307c, 0x4152, 0x95, 0xd6, 0x8e, 0x2e, 0xc4, 0xa5, 0x3e, 0x1a	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates the new register handle at `EFI_TPL_NOTIFY`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandleBuffer()` via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.14.31	0x94de767d, 0x38d1, 0x4205, 0x9f, 0xf9, 0xfd, 0x71, 0xf3, 0x7e, 0x81, 0x27	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates the new register handle at `EFI_TPL_APPLICATION`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandleBuffer()` via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. The return handle number should be 1.
5.1.3.14.32	0xf0bf589a, 0xdbfc, 0x4f36, 0xa1, 0x28, 0xbb, 0x95, 0x0d, 0x65, 0xe7, 0xff	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates the new register handle at `EFI_TPL_CALLBACK`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandleBuffer()` via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. The return handle number should be 1.

Number	GUID	Assertion	Test Description
5.1.3.14.33	0x684d6623, 0x49d2, 0x4807, 0x83, 0x67, 0xa3, 0xc4, 0x0d, 0xc6, 0xdb, 0x4a	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates the new register handle at **EFI_TPL_NOTIFY**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandleBuffer()** via search type **ByRegisterNotify** with the search key generated by previous **RegisterProtocolNotify**. The return handle number should be 1.
5.1.3.14.34	0xd690f3cd, 0x52e8, 0x4fab, 0x9b, 0x01, 0x75, 0x37, 0xa4, 0x20, 0xe8, 0xd4	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates the new register handle at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandleBuffer()** via search type **ByRegisterNotify** with the search key generated by previous **RegisterProtocolNotify**. The return handle should be matched.

Number	GUID	Assertion	Test Description
5.1.3.14.35	0xe284b0bf, 0xac06, 0x45af, 0xa5, 0x73, 0x19, 0x9c, 0xd8, 0xce, 0x67, 0x44	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates the new register handle at `EFI_TPL_CALLBACK`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandleBuffer()` via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. The return handle should be matched.
5.1.3.14.36	0x03e06b5f, 0xee50, 0x46c4, 0xa2, 0xfe, 0x47, 0x63, 0xc5, 0x6e, 0x90, 0xd5	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates the new register handle at `EFI_TPL_NOTIFY`	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Call `InstallProtocolInterface()` to install `TestProtocol1` onto a new handles. 3. Call `LocateHandleBuffer()` via search type `ByRegisterNotify` with the search key generated by previous `RegisterProtocolNotify`. The return handle should be matched.

Number	GUID	Assertion	Test Description
5.1.3.14.37	0x6a2c8795, 0x5f4f, 0x4fb0, 0xae, 0x45, 0xcc, 0xab, 0x73, 0x22, 0x31, 0x78	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates the new register handle at **EFI_TPL_APPLICATION**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandleBuffer()** via search type **ByRegisterNotify** with the search key generated by previous **RegisterProtocolNotify**. 4. Call **LocateHandleBuffer()** again. The return code should be **EFI_NOT_FOUND**.
5.1.3.14.38	0x61b79601, 0xd085, 0x4733, 0x91, 0xea, 0x1c, 0x94, 0x30, 0xb1, 0x31, 0xb8	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates the new register handle at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Call **InstallProtocolInterface()** to install **TestProtocol1** onto a new handles. 3. Call **LocateHandleBuffer()** via search type **ByRegisterNotify** with the search key generated by previous **RegisterProtocolNotify**. 4. Call **LocateHandleBuffer()** again. The return code should be **EFI_NOT_FOUND**.

Number	GUID	Assertion	Test Description
5.1.3.14.39	0x8b0d77ac, 0x08d0, 0x4c8c, 0xa4, 0x0c, 0xea, 0x43, 0x46, 0xb6, 0x33, 0x86	BS.LocateHandleBuffer – LocateHandleBuffer() locates the new register handle at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() to register for TestProtocol1's installation. 2. Call InstallProtocolInterface() to install TestProtocol1 onto a new handles. 3. Call LocateHandleBuffer() via search type ByRegisterNotify with the search key generated by previous RegisterProtocolNotify. 4. Call LocateHandleBuffer() again. The return code should be EFI_NOT_FOUND.
5.1.3.14.40	0x423bb934, 0xbbe3, 0x4841, 0xb3, 0x15, 0x92, 0xa0, 0xfa, 0x85, 0x67, 0xfc	BS.LocateHandleBuffer – LocateHandleBuffer() returns EFI_SUCCESS with a Type value of ByProtocol at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto 10 new handles. InstallProtocolInterface() return code should be EFI_SUCCESS.
5.1.3.14.41	0x3b0019f3, 0x7eb6, 0x4662, 0xa9, 0x05, 0x4a, 0xe2, 0x26, 0xb4, 0x92, 0xa7	BS.LocateHandleBuffer – LocateHandleBuffer() returns EFI_SUCCESS with a Type value of ByProtocol at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto 10 new handles. InstallProtocolInterface() return code should be EFI_SUCCESS.
5.1.3.14.42	0x7e86a93d, 0x5d29, 0x4b3d, 0x82, 0x2f, 0xdd, 0x93, 0xb0, 0xb4, 0x4b, 0x22	BS.LocateHandleBuffer – LocateHandleBuffer() returns EFI_SUCCESS with a Type value of ByProtocol at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto 10 new handles. InstallProtocolInterface() return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.14.43	0x0df33644, 0x4729, 0x400e, 0xa7, 0x99, 0x84, 0x24, 0xa8, 0xd4, 0x58, 0x09	BS.LocateHandleBuffer – LocateHandleBuffer() returns EFI_SUCCESS with a Type value of ByProtocol at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto 10 new handles. 2. Call LocateHandleBuffer() via search type ByProtocol to attempt to locate all handles that support TestProtocol1. The return code should be EFI_SUCCESS.
5.1.3.14.44	0x44311df6, 0x4f7a, 0x49e1, 0x84, 0x7e, 0xdd, 0x30, 0x8c, 0x7a, 0xc5, 0x2f	BS.LocateHandleBuffer – LocateHandleBuffer() returns EFI_SUCCESS with a Type value of ByProtocol at EFI_TPL_CALLBACK	1. Install TestProtocol1 onto 10 new handles. 2. Call LocateHandleBuffer() via search type ByProtocol to attempt to locate all handles that support TestProtocol1. The return code should be EFI_SUCCESS.
5.1.3.14.45	0xd7927271, 0x3631, 0x424c, 0xad, 0x83, 0xec, 0xa5, 0x2a, 0x64, 0x5f, 0x92	BS.LocateHandleBuffer – LocateHandleBuffer() returns EFI_SUCCESS with a Type value of ByProtocol at EFI_TPL_NOTIFY	1. Install TestProtocol1 onto 10 new handles. 2. Call LocateHandleBuffer() via search type ByProtocol to attempt to locate all handles that support TestProtocol1. The return code should be EFI_SUCCESS.
5.1.3.14.46	0x0bdcd179, 0xf25c, 0x4002, 0x9c, 0x6b, 0x5e, 0xea, 0x13, 0xdc, 0xa4, 0x13	BS.LocateHandleBuffer – LocateHandleBuffer() locates handles by protocol at EFI_TPL_APPLICATION	1. Install TestProtocol1 onto 10 new handles. 2. Call LocateHandleBuffer() via search type ByProtocol to attempt to locate all handles that support TestProtocol1. The return handle number should be 10.

Number	GUID	Assertion	Test Description
5.1.3.14.47	0x8f909926, 0x153f, 0x4dc6, 0xad, 0xd3, 0x89, 0x46, 0x6b, 0x82, 0xa9, 0x68	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates handles by protocol at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandleBuffer()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return handle number should be 10.
5.1.3.14.48	0x75d8aa1b, 0x75d9, 0x4122, 0xb7, 0xa5, 0xa3, 0x8c, 0x77, 0x9f, 0xf0, 0x1e	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates handles by protocol at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandleBuffer()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return handle number should be 10.
5.1.3.14.49	0xae68a349, 0x9644, 0x4156, 0x82, 0x77, 0x44, 0x77, 0x79, 0x5b, 0xca, 0xda	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates handles by protocol at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandleBuffer()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return handles should equal to those created.
5.1.3.14.50	0x0283802c, 0x2f33, 0x46ee, 0xb6, 0xec, 0x0a, 0xe4, 0x0d, 0x70, 0xfe, 0x3e	**BS.LocateHandleBuffer – LocateHandleBuffer()** locates handles by protocol at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** onto 10 new handles. 2. Call **LocateHandleBuffer()** via search type **ByProtocol** to attempt to locate all handles that support **TestProtocol1**. The return handles should equal to those created.

Number	GUID	Assertion	Test Description
5.1.3.14.51	0x5a50388b, 0xb7e9, 0x485c, 0x8f, 0xdd, 0x1f, 0xaf, 0xe9, 0xd2, 0x45, 0x16	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates handles by protocol at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto 10 new handles. 2. Call `LocateHandleBuffer()` via search type `ByProtocol` to attempt to locate all handles that support `TestProtocol1`. The return handles should equal to those created.
5.1.3.14.52	0x9bfc5990, 0x24a6, 0x4f73, 0x8f, 0xa3, 0x5d, 0x20, 0xa6, 0xe1, 0xb9, 0x53	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates handles by protocol at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` onto 10 new handles. 2. Call `LocateHandleBuffer()` via search type `ByProtocol` to attempt to locate all handles that support `TestProtocol1`. `TestProtocol1` should be located via each return handle.
5.1.3.14.53	0xe6591929, 0xd475, 0x483c, 0xa9, 0x1b, 0x43, 0x12, 0xba, 0x4e, 0x59, 0x8d	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates handles by protocol at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` onto 10 new handles. 2. Call `LocateHandleBuffer()` via search type `ByProtocol` to attempt to locate all handles that support `TestProtocol1`. `TestProtocol1` should be located via each return handle.
5.1.3.14.54	0x746f82f2, 0x8b90, 0x451a, 0xaf, 0x0b, 0xe6, 0xaa, 0x1b, 0xed, 0x4b, 0x27	`BS.LocateHandleBuffer – LocateHandleBuffer()` locates handles by protocol at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` onto 10 new handles. 2. Call `LocateHandleBuffer()` via search type `ByProtocol` to attempt to locate all handles that support `TestProtocol1`. `TestProtocol1` should be located via each return handle.

3.3.15 LocateProtocol()

Number	GUID	Assertion	Test Description
5.1.3.15.1	0x972e9815, 0x5a39, 0x4a39, 0x98, 0x08, 0x18, 0x17, 0x23, 0x7e, 0xb9, 0x05	BS.LocateProtocol – LocateProtocol() returns EFI_INVALID_PARAMETER with NULL interface	1. Call LocateProtocol() with NULL interface (type is void **). The return code should be EFI_INVALID_PARAMETER.
5.1.3.15.2	0x336a39f9, 0x7771, 0x44f7, 0x9f, 0xc1, 0xb4, 0x1b, 0x8d, 0x6a, 0x86, 0x1f	BS.LocateProtocol – LocateProtocol() returns EFI_NOT_FOUND with never installed protocol	1. Call LocateProtocol() to attempt to locate a protocol that is never installed in the system. The return code should be EFI_NOT_FOUND.
5.1.3.15.3	0x711df728, 0x1a59, 0x4298, 0xaf, 0xf5, 0x1b, 0x6f, 0x62, 0x24, 0xa3, 0xbf	BS.LocateProtocol – LocateProtocol() returns EFI_NOT_FOUND if no new protocol installed for the Registration	1. Call RegisterNotify() to register for the specified protocol. 2. Call LocateProtocol() with Registration returned from RegisterNotify(). The return code must be EFI_NOT_FOUND.
5.1.3.15.4	0x30c4caa5, 0x90ef, 0x44e8, 0xb1, 0x80, 0x33, 0x36, 0xff, 0x36, 0x98, 0xfc	BS.LocateProtocol – LocateProtocol() returns EFI_SUCCESS with exist protocol at EFI_TPL_APPLICATION.	1. Install TestProtocol1 onto a new handle. 2. Call LocateProtocol() to locate the protocol. The return code should be EFI_SUCCESS.
5.1.3.15.5	0xbc9928fd, 0xd6ee, 0x4238, 0x97, 0x53, 0xb6, 0xda, 0x3f, 0xfb, 0x57, 0xad	BS.LocateProtocol – LocateProtocol() returns EFI_SUCCESS with exist protocol at EFI_TPL_CALLBACK.	1. Install TestProtocol1 onto a new handle. 2. Call LocateProtocol() to locate the protocol. The return code should be EFI_SUCCESS.
5.1.3.15.6	0x29194f89, 0xae18, 0x4059, 0xba, 0xa9, 0x19, 0x44, 0xb1, 0x04, 0x76, 0x03	BS.LocateProtocol – LocateProtocol() returns EFI_SUCCESS with exist protocol at EFI_TPL_NOTIFY.	1. Install TestProtocol1 onto a new handle. 2. Call LocateProtocol() to locate the protocol. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.15.7	0x8f5fde8a, 0xc855, 0x4c8e, 0x9e, 0x4d, 0x27, 0xcb, 0xf8, 0x74, 0xb3, 0xc7	**BS.LocateProtocol – LocateProtocol()** locates exist protocol at **EFI_TPL_APPLICATION**.	1. Install **TestProtocol1** onto a new handle. 2. Call **LocateProtocol()** to locate the protocol. The **TestProtocol1**'s function should be accessed and executed correctly.
5.1.3.15.8	0x6fbe36a1, 0x7d50, 0x4baa, 0xa1, 0xf4, 0x90, 0x07, 0xff, 0x6f, 0x28, 0xc2	**BS.LocateProtocol – LocateProtocol()** locates exist protocol at **EFI_TPL_CALLBACK**.	1. Install **TestProtocol1** onto a new handle. 2. Call **LocateProtocol()** to locate the protocol. The **TestProtocol1**'s function should be accessed and executed correctly.
5.1.3.15.9	0x9106e5c2, 0x6a82, 0x447e, 0xaf, 0x96, 0x2b, 0x7a, 0xb2, 0xa8, 0x70, 0xd9	**BS.LocateProtocol – LocateProtocol()** locates exist protocol at **EFI_TPL_NOTIFY**.	1. Install **TestProtocol1** onto a new handle. 2. Call **LocateProtocol()** to locate the protocol. The **TestProtocol1**'s function should be accessed and executed correctly.
5.1.3.15.10	0x70358727, 0x45c5, 0x4d79, 0xb2, 0xf8, 0xa6, 0x0a, 0x33, 0x06, 0x04, 0x49	**BS.LocateProtocol – LocateProtocol()** returns **EFI_SUCCESS** with registration key at **EFI_TPL_APPLICATION**.	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Install **TestProtocol1** onto a new handle. 3. Call **LocateProtocol()** with the registration key to attempt to retrieve **TestProtocol1**'s instance. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.15.11	0x42f3df2e, 0xa23c, 0x4f44, 0xb7, 0xb1, 0xdd, 0x62, 0x77, 0x79, 0x04, 0x58	`BS.LocateProtocol – LocateProtocol()` returns `EFI_SUCCESS` with registration key at `EFI_TPL_CALLBACK`.	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Install `TestProtocol1` onto a new handle. 3. Call `LocateProtocol()` with the registration key to attempt to retrieve `TestProtocol1`'s instance. The return code should be `EFI_SUCCESS`.
5.1.3.15.12	0x2c0ea674, 0xd3cb, 0x4a7a, 0xb1, 0x4b, 0xf4, 0xa8, 0x53, 0x0c, 0x17, 0xdd	`BS.LocateProtocol – LocateProtocol()` returns `EFI_SUCCESS` with registration key at `EFI_TPL_NOTIFY`.	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Install `TestProtocol1` onto a new handle. 3. Call `LocateProtocol()` with the registration key to attempt to retrieve `TestProtocol1`'s instance. The return code should be `EFI_SUCCESS`.
5.1.3.15.13	0xcff56950, 0x1dda, 0x4c41, 0xaa, 0x71, 0x58, 0x41, 0x27, 0xad, 0x23, 0xd9	`BS.LocateProtocol – LocateProtocol()` locates protocol with registration key at `EFI_TPL_APPLICATION`.	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Install `TestProtocol1` onto a new handle. 3. Call `LocateProtocol()` with the registration key to attempt to retrieve `TestProtocol1`'s instance. The `TestProtocol1`'s function should be accessed and executed correctly.

Number	GUID	Assertion	Test Description
5.1.3.15.14	0x47755194, 0x49e3, 0x452f, 0x9c, 0x02, 0x61, 0xa8, 0x89, 0x54, 0x5f, 0x43	**BS.LocateProtocol – LocateProtocol()** locates protocol with registration key at **EFI_TPL_CALLBACK**.	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Install **TestProtocol1** onto a new handle. 3. Call **LocateProtocol()** with the registration key to attempt to retrieve **TestProtocol1**'s instance. The **TestProtocol1**'s function should be accessed and executed correctly.
5.1.3.15.15	0xc385d8ab, 0x6038, 0x43b2, 0x82, 0x9d, 0x2d, 0xa4, 0x24, 0x62, 0x8f, 0xe6	**BS.LocateProtocol – LocateProtocol()** locates protocol with registration key at **EFI_TPL_NOTIFY**.	1. Call **RegisterProtocolNotify()** to register for **TestProtocol1**'s installation. 2. Install **TestProtocol1** onto a new handle. 3. Call **LocateProtocol()** with the registration key to attempt to retrieve **TestProtocol1**'s instance. The **TestProtocol1**'s function should be accessed and executed correctly.

Number	GUID	Assertion	Test Description
5.1.3.15.16	0xc9ed276a, 0x3d30, 0x4510, 0xa5, 0xdd, 0x93, 0x2d, 0xd8, 0x4f, 0x94, 0x9e	`BS.LocateProtocol – LocateProtocol()` locates protocol with registration key at `EFI_TPL_APPLICATION`.	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Install `TestProtocol1` onto a new handle. 3. Call `LocateProtocol()` with the registration key to attempt to retrieve `TestProtocol1`'s instance. 4. Call `LocateProtocol()` with the registration key again. The return code should be `EFI_NOT_FOUND`.
5.1.3.15.17	0x2e2d0e7e, 0x8de3, 0x4522, 0x84, 0x0d, 0x2c, 0xda, 0x60, 0xcb, 0x11, 0x5c	`BS.LocateProtocol – LocateProtocol()` locates protocol with registration key at `EFI_TPL_CALLBACK`.	1. Call `RegisterProtocolNotify()` to register for `TestProtocol1`'s installation. 2. Install `TestProtocol1` onto a new handle. 3. Call `LocateProtocol()` with the registration key to attempt to retrieve `TestProtocol1`'s instance. 4. Call `LocateProtocol()` with the registration key again. The return code should be `EFI_NOT_FOUND`.

Number	GUID	Assertion	Test Description
5.1.3.15.18	0x63940439, 0xd67c, 0x4ae0, 0xb9, 0x14, 0x90, 0xe7, 0x09, 0x40, 0x05, 0x44	BS.LocateProtocol – LocateProtocol() locates protocol with registration key at EFI_TPL_NOTIFY.	1. Call RegisterProtocolNotify() to register for TestProtocol1's installation. 2. Install TestProtocol1 onto a new handle. 3. Call LocateProtocol() with the registration key to attempt to retrieve TestProtocol1's instance. 4. Call LocateProtocol() with the registration key again. The return code should be EFI_NOT_FOUND.
5.1.3.15.19	0x3274a5c2, 0x1a28, 0x4231, 0x8f, 0x3c, 0x4a, 0xe1, 0x66, 0x41, 0x26, 0x3f	BS.LocateProtocol – LocateProtocol() returns EFI_SUCCESS with NULL protocol interface at EFI_TPL_APPLICATION.	1. Install TestNoInterfaceProtocol1 onto a new handle. 2. Call LocateProtocol() to attempt to retrieve TestNoInterfaceProtocol1's instance. The return code should be EFI_SUCCESS.
5.1.3.15.20	0x2e8a72b3, 0x4cab, 0x4e02, 0xa1, 0x7f, 0xbc, 0xda, 0x52, 0xe9, 0xe3, 0x81	BS.LocateProtocol – LocateProtocol() returns EFI_SUCCESS with NULL protocol interface at EFI_TPL_CALLBACK.	1. Install TestNoInterfaceProtocol1 onto a new handle. 2. Call LocateProtocol() to attempt to retrieve TestNoInterfaceProtocol1's instance. The return code should be EFI_SUCCESS.
5.1.3.15.21	0x712cef7b, 0xdc81, 0x466c, 0x97, 0x85, 0xad, 0xa1, 0x3b, 0x71, 0x33, 0xf5	BS.LocateProtocol – LocateProtocol() returns EFI_SUCCESS with NULL protocol interface at EFI_TPL_NOTIFY.	1. Install TestNoInterfaceProtocol1 onto a new handle. 2. Call LocateProtocol() to attempt to retrieve TestNoInterfaceProtocol1's instance. The return code should be EFI_SUCCESS.

3.3.16 InstallMultipleProtocolInterfaces()

Number	GUID	Assertion	Test Description
5.1.3.16.1	0x804b0522, 0x4ff9, 0x47cc, 0xa6, 0x2a, 0xe3, 0x27, 0xec, 0xce, 0xbe, 0x4b	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` returns `EFI_ALREADY_STARTED` with device path protocol instance already present	1. Call `InstallMultipleProtocolInterfaces()` to attempt to install multiple protocol instances at the same time, among them is a device path protocol instance that is already present in the handle database. The return code should be `EFI_ALREADY_STARTED`.
5.1.3.16.2	0x3ff2cc4e, 0xf56a, 0x44a7, 0xb4, 0x86, 0x1f, 0x7e, 0x4d, 0x63, 0x97, 0x94	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` does not install any interfaces with device path protocol instance already present	1. Call `InstallMultipleProtocolInterfaces()` to attempt to install multiple protocol instances at the same time, among them is a device path protocol instance that is already present in the handle database. All the protocol instances should not be installed onto the handle during this call.
5.1.3.16.3	0x79d79b37, 0x756f, 0x4754, 0x80, 0x43, 0x58, 0x44, 0xa7, 0x22, 0xac, 0x7d	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` returns `EFI_INVALID_PARAMETER` with invalid handle	1. Call `InstallMultipleProtocolInterfaces()` with an invalid handle. The return code should be `EFI_INVALID_PARAMETER`.
5.1.3.16.4	0xf7e5fa57, 0xb2bb, 0x4ace, 0xa3, 0x99, 0x43, 0xd2, 0x26, 0x44, 0x83, 0x4c	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` does not install any interfaces with invalid handle	1. Call `InstallMultipleProtocolInterfaces()` with an invalid handle. All protocols should not be installed onto a handle during this call.

Number	GUID	Assertion	Test Description
5.1.3.16.5	0x090defdb, 0x24a2, 0x43ff, 0xa6, 0x14, 0x75, 0x7b, 0xc2, 0xce, 0x9c, 0xdb	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** returns **EFI_INVALID_PARAMETER** with same protocol multiple times	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto a new handle. 2. Call **InstallMultipleProtocolInterfaces()** again to try to install **TestProtocol1** & **TestProtocol2** onto the same handle. The return code must be **EFI_INVALID_PARAMETER**.
5.1.3.16.6	0xdb705ca6, 0x40ca, 0x4abc, 0x92, 0x66, 0x78, 0x0d, 0x3b, 0xac, 0x62, 0x63	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** does not install any interfaces with same protocol multiple times	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto a new handle. 2. Call **InstallMultipleProtocolInterfaces()** again to try to install **TestProtocol1** & **TestProtocol2** onto the same handle. The **TestProtocol1** should still exist and **TestProtocol2** should not be installed..
5.1.3.16.7	0x12cdfc3b, 0x10b7, 0x45cc, 0x81, 0x84, 0xe6, 0x64, 0x42, 0x2c, 0xff, 0x64	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with one protocol on new handle at **EFI_TPL_APPLICATION**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto a new handle. The return code should be **EFI_SUCCESS**.
5.1.3.16.8	0x3e85df7a, 0x6128, 0x41a2, 0xa6, 0x93, 0x42, 0xba, 0xe2, 0x1c, 0xe7, 0xa6	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with one protocol on new handle at **EFI_TPL_CALLBACK**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto a new handle. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.16.9	0x0012978f, 0xb761, 0x4531, 0xbd, 0xe0, 0xbd, 0x16, 0xfd, 0x98, 0x19, 0x02	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` returns `EFI_SUCCESS` with one protocol on new handle at `EFI_TPL_NOTIFY`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. The return code should be `EFI_SUCCESS`.
5.1.3.16.10	0x8707601e, 0x4d04, 0x4a15, 0xb1, 0x53, 0x20, 0x8b, 0x9b, 0x3d, 0xc9, 0x2e	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_APPLICATION`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. A new handle should be created.
5.1.3.16.11	0x80ab6d49, 0x43f8, 0x4c1f, 0xbb, 0x64, 0x9c, 0x20, 0x99, 0x96, 0x62, 0x4a	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_CALLBACK`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. A new handle should be created.
5.1.3.16.12	0x976e2272, 0x0454, 0x4d88, 0x9e, 0xf2, 0x7a, 0x54, 0xa9, 0x76, 0x81, 0x66	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_NOTIFY`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. A new handle should be created.
5.1.3.16.13	0xd2c0eaa9, 0xaa4d, 0x447a, 0xa9, 0xd1, 0x6e, 0x0f, 0x78, 0x31, 0x17, 0x48	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_APPLICATION`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. The handle should be located via the protocol.
5.1.3.16.14	0xeb664f78, 0x8e6f, 0x4dc7, 0xb1, 0xa1, 0xd6, 0x0d, 0xf9, 0x6f, 0x1f, 0xfd	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_CALLBACK`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. The handle should be located via the protocol.
5.1.3.16.15	0x7b54fb1c, 0x1731, 0x423c, 0xa0, 0x29, 0xef, 0xd1, 0x0c, 0xb4, 0x41, 0x69	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_NOTIFY`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. The handle should be located via the protocol.

Number	GUID	Assertion	Test Description
5.1.3.16.16	0x7aaf4b71, 0xdd01, 0x4562, 0x82, 0x1a, 0x13, 0x08, 0x7d, 0x9f, 0x8a, 0x75	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_APPLICATION`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. `TestProtocol1` should be located via the handle.
5.1.3.16.17	0x5fba4597, 0x43e6, 0x4ba2, 0x80, 0x2d, 0xba, 0x56, 0xaf, 0x10, 0x06, 0x66	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_CALLBACK`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. `TestProtocol1` should be located via the handle.
5.1.3.16.18	0x9a4f2f3b, 0x5209, 0x40d3, 0x95, 0xa2, 0x9a, 0xea, 0x98, 0x19, 0x8a, 0xc0	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_NOTIFY`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. `TestProtocol1` should be located via the handle.
5.1.3.16.19	0x802b5c2e, 0x2c3c, 0x43ff, 0x9c, 0xda, 0x04, 0xf8, 0x94, 0x42, 0xb5, 0x7b	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_APPLICATION`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. `TestProtocol1`'s functions should be accessed and be executed correctly.
5.1.3.16.20	0xb7ffd827, 0x9478, 0x40c0, 0xad, 0x9b, 0x03, 0x22, 0x99, 0x2e, 0xc5, 0x97	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_CALLBACK`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. `TestProtocol1`'s functions should be accessed and be executed correctly.
5.1.3.16.21	0x77fe21e8, 0x58fd, 0x468d, 0xad, 0xbc, 0x5c, 0x4b, 0xbb, 0xe8, 0x5e, 0x59	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs one protocol on new handle at `EFI_TPL_NOTIFY`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1` onto a new handle. `TestProtocol1`'s functions should be accessed and be executed correctly.

Number	GUID	Assertion	Test Description
5.1.3.16.22	0xbccb1238, 0xd969, 0x4a35, 0xa1, 0xc4, 0x74, 0x5c, 0xb1, 0x79, 0x63, 0x26	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() returns EFI_SUCCESS with one protocol on an existing handle at EFI_TPL_APPLICATION	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 onto an existing handle. The return code should be EFI_SUCCESS.
5.1.3.16.23	0xd56ff74a, 0x1305, 0x43ad, 0x9f, 0xd6, 0x17, 0x8d, 0x7b, 0x67, 0x50, 0x66	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() returns EFI_SUCCESS with one protocol on new handle at EFI_TPL_CALLBACK	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 onto a new handle. The return code should be EFI_SUCCESS.
5.1.3.16.24	0xa6ebc379, 0x5753, 0x40b4, 0x81, 0xb4, 0x9c, 0xdc, 0x79, 0x6c, 0xe9, 0x5d	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() returns EFI_SUCCESS with one protocol on an existing handle at EFI_TPL_NOTIFY	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 onto an existing handle. The return code should be EFI_SUCCESS.
5.1.3.16.25	0x41b1e88c, 0x0162, 0x4dfd, 0xb1, 0x14, 0x89, 0x97, 0xeb, 0xed, 0x64, 0x11	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs one protocol on an existing handle at EFI_TPL_APPLICATION	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 onto an existing handle. No new handle should be created.
5.1.3.16.26	0x2d864f91, 0xdddc, 0x4f34, 0xb9, 0x4d, 0x90, 0x0a, 0xef, 0x44, 0x9c, 0xd3	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs one protocol on an existing handle at EFI_TPL_CALLBACK	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 onto an existing handle. No new handle should be created.
5.1.3.16.27	0x6e1e752c, 0x9320, 0x4d73, 0x87, 0x30, 0xce, 0x76, 0x65, 0x27, 0x24, 0x20	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs one protocol on an existing handle at EFI_TPL_NOTIFY	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 onto an existing handle. No new handle should be created.

Number	GUID	Assertion	Test Description
5.1.3.16.28	0xbd4c5e34, 0x43d5, 0x4145, 0xb5, 0x29, 0x36, 0xf9, 0xf5, 0x2d, 0xb2, 0x58	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs one protocol on an existing handle at **EFI_TPL_APPLICATION**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto an existing handle. The handle should be located via the protocol.
5.1.3.16.29	0x74d0c8f7, 0x1e32, 0x4b4c, 0x87, 0x71, 0xbd, 0xce, 0x1d, 0x7d, 0xe8, 0xce	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs one protocol on an existing handle at **EFI_TPL_CALLBACK**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto an existing handle. The handle should be located via the protocol.
5.1.3.16.30	0xc27c0e00, 0x4d66, 0x44b8, 0xad, 0x3c, 0x50, 0x94, 0x62, 0x30, 0xaf, 0x31	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs one protocol on an existing handle at **EFI_TPL_NOTIFY**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto an existing handle. The handle should be located via the protocol.
5.1.3.16.31	0xb97d0b30, 0xc4a2, 0x44f4, 0xb4, 0xf4, 0x94, 0x3c, 0xd9, 0x82, 0x10, 0x7a	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs one protocol on an existing handle at **EFI_TPL_APPLICATION**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto an existing handle. **TestProtocol1** should be located via the handle.
5.1.3.16.32	0xbb4f764c, 0x301e, 0x4781, 0x9b, 0x70, 0x23, 0x0b, 0xaf, 0x4e, 0xf5, 0xda	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs one protocol on an existing handle at **EFI_TPL_CALLBACK**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto an existing handle. **TestProtocol1** should be located via the handle.
5.1.3.16.33	0x4c51e23d, 0x18c8, 0x4f8a, 0xa8, 0x54, 0xe2, 0xbf, 0x57, 0xcb, 0x15, 0xfe	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs one protocol on an existing handle at **EFI_TPL_NOTIFY**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto an existing handle. **TestProtocol1** should be located via the handle.

Number	GUID	Assertion	Test Description
5.1.3.16.34	0x96bbdd38, 0x6e66, 0x417d, 0xa8, 0x7e, 0xf1, 0x0f, 0x2f, 0xa6, 0x3c, 0xd6	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs one protocol on an existing handle at **EFI_TPL_APPLICATION**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto an existing handle. **TestProtocol1**'s functions should be accessed and be executed correctly.
5.1.3.16.35	0x9647fb47, 0xb854, 0x495b, 0xbc, 0xff, 0xf8, 0xed, 0x80, 0xe9, 0xe5, 0xd8	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs one protocol on an existing handle at **EFI_TPL_CALLBACK**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto an existing handle. **TestProtocol1**'s functions should be accessed and be executed correctly.
5.1.3.16.36	0x8902c01f, 0x9215, 0x4902, 0xa3, 0x70, 0xd3, 0x11, 0xda, 0xfc, 0xc2, 0xa8	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs one protocol on an existing handle at **EFI_TPL_NOTIFY**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto an existing handle. **TestProtocol1**'s functions should be accessed and be executed correctly.
5.1.3.16.37	0xe851fe59, 0xf599, 0x4b56, 0xa3, 0xa8, 0xf1, 0xde, 0x3f, 0x29, 0xd6, 0xbf	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with multiple protocols on new handle at **EFI_TPL_APPLICATION**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1**, **TestProtocol2**, **TestProtocol3** onto a new handle. The return code should be **EFI_SUCCESS**.
5.1.3.16.38	0x45b4418e, 0x997e, 0x4050, 0xbc, 0xc4, 0x70, 0xed, 0x4b, 0xf0, 0x67, 0x9e	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with multiple protocols on new handle at **EFI_TPL_CALLBACK**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1**, **TestProtocol2**, **TestProtocol3** onto a new handle. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.16.39	0x6621263d, 0x39b8, 0x410c, 0xa7, 0x9b, 0x35, 0xcf, 0x38, 0xaf, 0xa3, 0xdb	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` returns `EFI_SUCCESS` with multiple protocols on new handle at `EFI_TPL_NOTIFY`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3` onto a new handle. The return code should be `EFI_SUCCESS`.
5.1.3.16.40	0x295381f4, 0x3106, 0x408b, 0xa0, 0x88, 0x4e, 0xa3, 0x1c, 0x8b, 0x57, 0x9b	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs multiple protocols on new handle at `EFI_TPL_APPLICATION`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3` onto a new handle. A new handle should be created.
5.1.3.16.41	0x092c02d7, 0xf796, 0x4a45, 0xa9, 0xc8, 0x01, 0xc3, 0x69, 0xa2, 0x93, 0x78	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs multiple protocols on new handle at `EFI_TPL_CALLBACK`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3` onto a new handle. A new handle should be created.
5.1.3.16.42	0x3e9922bb, 0xc501, 0x402b, 0xa0, 0x01, 0xf3, 0x2e, 0xc9, 0xeb, 0x37, 0x72	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs multiple protocols on new handle at `EFI_TPL_NOTIFY`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3` onto a new handle. A new handle should be created.
5.1.3.16.43	0x1b5a97be, 0xa885, 0x4878, 0x94, 0xf4, 0x62, 0x51, 0x82, 0x8e, 0xea, 0xb0	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs multiple protocols on new handle at `EFI_TPL_APPLICATION`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3` onto a new handle. The handle should be located via each protocol.

Number	GUID	Assertion	Test Description
5.1.3.16.44	0x031f8b77, 0xf024, 0x4979, 0x99, 0x5f, 0x19, 0x8a, 0x82, 0xac, 0x4c, 0x0f	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs multiple protocols on new handle at **EFI_TPL_CALLBACK**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto a new handle. The handle should be located via each protocol.
5.1.3.16.45	0x65008362, 0x42ee, 0x4599, 0x8b, 0x51, 0xd0, 0xcc, 0x3d, 0x05, 0x14, 0xf3	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs multiple protocols on new handle at **EFI_TPL_NOTIFY**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto a new handle. The handle should be located via each protocol.
5.1.3.16.46	0xe79a6e38, 0x3451, 0x4f7c, 0x96, 0xc9, 0x05, 0xaa, 0x94, 0x7d, 0x1a, 0x45	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs one protocol on new handle at **EFI_TPL_APPLICATION**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto a new handle. Each protocol should be located via the handle.
5.1.3.16.47	0x2239ef0b, 0x833a, 0x4525, 0x9a, 0x9f, 0x00, 0x2a, 0x31, 0xbf, 0x3a, 0x01	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs multiple protocols on new handle at **EFI_TPL_CALLBACK**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto a new handle. Each protocol should be located via the handle.
5.1.3.16.48	0xad472682, 0xdc2a, 0x4cca, 0x8a, 0x53, 0x47, 0xcb, 0x65, 0x44, 0x92, 0xcf	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** installs multiple protocols on new handle at **EFI_TPL_NOTIFY**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** onto a new handle. Each protocol should be located via the handle.
5.1.3.16.49	0x86b364b6, 0xef09, 0x4e65, 0xb5, 0x6a, 0xb8, 0x87, 0x92, 0xc2, 0xc2, 0xbb	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with multiple protocols on an existing handle at **EFI_TPL_APPLICATION**	1. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1**, **TestProtocol2**, **TestProtocol3** onto an existing handle. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.16.50	0x6fa7054c, 0xd436, 0x42d6, 0x8b, 0x73, 0x79, 0xaf, 0xf6, 0x63, 0xa4, 0x1d	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` returns `EFI_SUCCESS` with multiple protocols on an existing handle at `EFI_TPL_CALLBACK`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3` onto an existing handle. The return code should be `EFI_SUCCESS`.
5.1.3.16.51	0x241337ae, 0x527d, 0x4a10, 0x8b, 0x56, 0x30, 0xdd, 0xa1, 0x52, 0x42, 0xf4	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` returns `EFI_SUCCESS` with multiple protocols on an existing handle at `EFI_TPL_NOTIFY`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3` onto an existing handle. The return code should be `EFI_SUCCESS`.
5.1.3.16.52	0xf1d61967, 0xba05, 0x4d4b, 0xa1, 0x90, 0x55, 0x39, 0x23, 0x3a, 0xfa, 0x92	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs multiple protocols on an existing handle at `EFI_TPL_APPLICATION`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3` onto an existing handle. No new handle should be created.
5.1.3.16.53	0x9b2ee3a0, 0x7f21, 0x4b94, 0xa0, 0x11, 0x5a, 0x2e, 0x8f, 0xd9, 0x96, 0x9d	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs multiple protocols on an existing handle at `EFI_TPL_CALLBACK`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3` onto an existing handle. No new handle should be created.
5.1.3.16.54	0x946a0349, 0x1233, 0x452e, 0xa0, 0x10, 0xa3, 0x19, 0xfe, 0x02, 0x4c, 0xb4	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs multiple protocols on an existing handle at `EFI_TPL_NOTIFY`	1. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, `TestProtocol3` onto an existing handle. No new handle should be created.

Number	GUID	Assertion	Test Description
5.1.3.16.55	0xd342993b, 0x753e, 0x466b, 0x9f, 0x92, 0x4f, 0x97, 0xf7, 0x6e, 0x74, 0x72	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs multiple protocols on an existing handle at EFI_TPL_APPLICATION	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1, TestProtocol2, TestProtocol3 onto an existing handle. The handle should be located via each protocol.
5.1.3.16.56	0x2e2cfed3, 0xba41, 0x4d40, 0x8e, 0xdd, 0xc5, 0xc5, 0xa0, 0x3d, 0xe9, 0xc1	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs multiple protocols on an existing handle at EFI_TPL_CALLBACK	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 onto an existing handle. The handle should be located via each protocol.
5.1.3.16.57	0x48783e17, 0x8143, 0x4af9, 0xa2, 0x28, 0x96, 0x55, 0x37, 0x00, 0xe2, 0x53	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs multiple protocols on an existing handle at EFI_TPL_NOTIFY	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 onto an existing handle. The handle should be located via each protocol.
5.1.3.16.58	0x835818d1, 0x1c63, 0x408e, 0xb9, 0xf7, 0x34, 0x54, 0xe9, 0x06, 0x59, 0xe2	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs one protocol on an existing handle at EFI_TPL_APPLICATION	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 onto an existing handle. Each protocol should be located via the handle.
5.1.3.16.59	0x03169da7, 0xfc5f, 0x43f6, 0x97, 0x53, 0x4a, 0x7e, 0x50, 0x90, 0xeb, 0x13	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs multiple protocols on an existing handle at EFI_TPL_CALLBACK	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 onto an existing handle. Each protocol should be located via the handle.
5.1.3.16.60	0xf45687b9, 0xec94, 0x4cc1, 0x98, 0xb6, 0x39, 0xc7, 0x8a, 0x0e, 0x8f, 0xee	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs multiple protocols on an existing handle at EFI_TPL_NOTIFY	1. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 onto an existing handle. Each protocol should be located via the handle.

Number	GUID	Assertion	Test Description
5.1.3.16.61	0xcd6ff9e0, 0xc307, 0x4b0f, 0x8b, 0xb1, 0xdb, 0x3c, 0x4a, 0x07, 0x0e, 0xc9	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** returns **EFI_ALREADY_STARTED** with same device path at **EFI_TPL_APPLICATION**	1. Call **InstallProtocolInterface()** to install a device path onto a new handle. 2. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1**, **TestProtocol2**, and the same device path as the one installed before onto another new handle. The return code should be **EFI_ALREADY_STARTED**.
5.1.3.16.62	0xd6a218f1, 0xda1c, 0x4030, 0xbc, 0xdf, 0x1b, 0xdc, 0x1f, 0x9f, 0xd5, 0x92	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** returns **EFI_ALREADY_STARTED** with same device path at **EFI_TPL_CALLBACK**	1. Call **InstallProtocolInterface()** to install a device path onto a new handle. 2. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1**, **TestProtocol2**, and the same device path as the one installed before onto another new handle. The return code should be **EFI_ALREADY_STARTED**.
5.1.3.16.63	0xe310ae92, 0xf894, 0x4fdd, 0xbe, 0xd4, 0xbf, 0x1b, 0x70, 0x0f, 0x4c, 0xad	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** returns **EFI_ALREADY_STARTED** with same device path at **EFI_TPL_NOTIFY**	1. Call **InstallProtocolInterface()** to install a device path onto a new handle. 2. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1**, **TestProtocol2**, and the same device path as the one installed before onto another new handle. The return code should be **EFI_ALREADY_STARTED**.

Number	GUID	Assertion	Test Description
5.1.3.16.64	0x571c7046, 0x58f0, 0x45a8, 0x86, 0x8d, 0xf1, 0x16, 0xd7, 0x02, 0xe7, 0x54	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs same device path at EFI_TPL_APPLICATION	1. Call InstallProtocolInterface() to install a device path onto a new handle. 2. Call InstallMultipleProtocolInterfaces() to install TestProtocol1, TestProtocol2, and the same device path as the one installed before onto another new handle. No new handle should be created.
5.1.3.16.65	0xbabbef02, 0x5645, 0x4284, 0xb7, 0x18, 0x18, 0xbe, 0xaa, 0x51, 0x52, 0xbf	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs same device path at EFI_TPL_CALLBACK	1. Call InstallProtocolInterface() to install a device path onto a new handle. 2. Call InstallMultipleProtocolInterfaces() to install TestProtocol1, TestProtocol2, and the same device path as the one installed before onto another new handle. No new handle should be created.
5.1.3.16.66	0x093b4b63, 0xcbad, 0x425a, 0xb0, 0xc5, 0xe6, 0xc1, 0x27, 0x4a, 0xba, 0x06	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() installs same device path at EFI_TPL_NOTIFY	1. Call InstallProtocolInterface() to install a device path onto a new handle. 2. Call InstallMultipleProtocolInterfaces() to install TestProtocol1, TestProtocol2, and the same device path as the one installed before onto another new handle. No new handle should be created.

Number	GUID	Assertion	Test Description
5.1.3.16.67	0xccf096ed, 0x327c, 0x44f7, 0xb2, 0xf1, 0x8d, 0xe4, 0x8d, 0x21, 0xfc, 0x54	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs same device path at `EFI_TPL_APPLICATION`	1. Call `InstallProtocolInterface()` to install a device path onto a new handle. 2. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, and the same device path as the one installed before onto another new handle. Each protocol should not be located.
5.1.3.16.68	0x386fcc7f, 0xf776, 0x4284, 0x90, 0x60, 0x16, 0x96, 0xa4, 0x4e, 0x37, 0x73	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs same device path at `EFI_TPL_CALLBACK`	1. Call `InstallProtocolInterface()` to install a device path onto a new handle. 2. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, and the same device path as the one installed before onto another new handle. Each protocol should not be located.
5.1.3.16.69	0x8bb68afb, 0x4656, 0x4bce, 0x80, 0x67, 0x60, 0x70, 0xda, 0x89, 0x04, 0x13	`BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()` installs same device path at `EFI_TPL_NOTIFY`	1. Call `InstallProtocolInterface()` to install a device path onto a new handle. 2. Call `InstallMultipleProtocolInterfaces()` to install `TestProtocol1`, `TestProtocol2`, and the same device path as the one installed before onto another new handle. Each protocol should not be located.

Number	GUID	Assertion	Test Description
5.1.3.16.70	0x42662a65, 0x4966, 0x4d14, 0x90, 0x53, 0xc9, 0x7d, 0x57, 0x0e, 0xcc, 0x3a	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() notifies the register function at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() to register some notify functions for TestProtocol1 and TestProtocol2. The return code should be EFI_SUCCESS.
5.1.3.16.71	0x288f4c75, 0xc1dc, 0x438d, 0x92, 0xe3, 0x13, 0xf4, 0x02, 0xff, 0xfe, 0x24	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() notifies the register function at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() to register some notify functions for TestProtocol1 and TestProtocol2. The return code should be EFI_SUCCESS.
5.1.3.16.72	0x6c1e2c2c, 0x7004, 0x4764, 0xb5, 0xce, 0x07, 0xe5, 0x0b, 0x08, 0xca, 0x38	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() notifies the register function at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() to register some notify functions for TestProtocol1 and TestProtocol2. The return code should be EFI_SUCCESS.
5.1.3.16.73	0xe25facbd, 0xd42f, 0x44f4, 0x8a, 0xa6, 0x2d, 0x17, 0x94, 0x34, 0x03, 0x61	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() notifies the register function at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() to register some notify functions for TestProtocol1 and TestProtocol2. 2. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 & TestProtocol2 at the same time. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.3.16.74	0xf40536b7, 0x0b97, 0x477d, 0x91, 0x86, 0x40, 0x64, 0x01, 0x60, 0x95, 0xa4	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() notifies the register function at EFI_TPL_CALLBACK	1. Call RegisterProtocolNotify() to register some notify functions for TestProtocol1 and TestProtocol2. 2. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 & TestProtocol2 at the same time. The return code should be EFI_SUCCESS.
5.1.3.16.75	0xd1fc105e, 0x8c44, 0x408a, 0xbc, 0x58, 0x42, 0xfa, 0x71, 0x8c, 0x64, 0xe6	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() notifies the register function at EFI_TPL_NOTIFY	1. Call RegisterProtocolNotify() to register some notify functions for TestProtocol1 and TestProtocol2. 2. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 & TestProtocol2 at the same time. The return code should be EFI_SUCCESS.
5.1.3.16.76	0xa1479f29, 0x960b, 0x493c, 0xb9, 0xd3, 0xfc, 0x07, 0x45, 0x90, 0x66, 0xcd	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() notifies the register function at EFI_TPL_APPLICATION	1. Call RegisterProtocolNotify() to register some notify functions for TestProtocol1 and TestProtocol2. 2. Call InstallMultipleProtocolInterfaces() to install TestProtocol1 & TestProtocol2 at the same time. All events notify functions should be invoked, and each was invoked once.

Number	GUID	Assertion	Test Description
5.1.3.16.77	0xbe2a26f3, 0xaa13, 0x43d9, 0x84, 0x8d, 0x0c, 0x09, 0xfd, 0x7f, 0xfe, 0x1b	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** notifies the register function at **EFI_TPL_CALLBACK**	1. Call **RegisterProtocolNotify()** to register some notify functions for **TestProtocol1** and **TestProtocol2**. 2. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** & **TestProtocol2** at the same time. All events notify functions should be invoked, and each was invoked once.
5.1.3.16.78	0x6c3b6ba1, 0xcd59, 0x4385, 0x96, 0x35, 0x29, 0x78, 0xf7, 0x24, 0x98, 0x97	**BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces()** notifies the register function at **EFI_TPL_NOTIFY**	1. Call **RegisterProtocolNotify()** to register some notify functions for **TestProtocol1** and **TestProtocol2**. 2. Call **InstallMultipleProtocolInterfaces()** to install **TestProtocol1** & **TestProtocol2** at the same time. All events notify functions should be invoked, and each was invoked once.
5.1.3.16.79	0x4242e59c, 0x7370, 0x4a87, 0x83, 0x8c, 0x66, 0xdf, 0xf0, 0x66, 0xe0, 0x1e	BS.InstallMultipleProtocolInterfaces – InstallMultipleProtocolInterfaces() returns EFI_INVALID_PARAMETER when handle is **NULL**	1. Call InstallMultipleProtocolInterfaces() with an NULL handle. The return code should be EFI_INVALID_PARAMETER.

3.3.17 UninstallMultipleProtocolInterfaces()

Number	GUID	Assertion	Test Description
5.1.3.17.1	0x2f6ac49a, 0x0f2d, 0x4392, 0xa0, 0xa6, 0x91, 0x80, 0xc9, 0xd2, 0x31, 0x77	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` returns `EFI_INVALID_PARAMETER` with a non-existent protocol	1. Call `UnInstallMultipleProtocolInterfaces()` to attempt to uninstall multiple protocol instances at the same time, among them is a protocol instance that does not exist on the handle. The return code should be `EFI_INVALID_PARAMETER`.
5.1.3.17.2	0x914d9c49, 0x0e54, 0x429a, 0x88, 0xc7, 0x93, 0xdb, 0xdc, 0x7d, 0xe0, 0x35	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` does not uninstall any interfaces with a non-existent protocol	1. Call `UnInstallMultipleProtocolInterfaces()` to attempt to uninstall multiple protocol instances at the same time, among them is a protocol instance that does not exist on the handle. All the other protocol instances should not be uninstalled from the handle during this call.
5.1.3.17.3	0x9b15125f, 0xec64, 0x4626, 0xbf, 0x69, 0x99, 0xc0, 0x2c, 0x20, 0x5f, 0xd5	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` returns `EFI_SUCCESS` with non-opened protocol at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.17.4	0xb9b20241, 0x96ce, 0x4742, 0xb1, 0x7b, 0x91, 0x9e, 0xdb, 0x96, 0x31, 0x85	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` returns `EFI_SUCCESS` with non-opened protocol at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` from the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.17.5	0xd33209ff, 0x9d19, 0x4d8e, 0xa6, 0xb7, 0x67, 0x1f, 0x10, 0xa1, 0x1a, 0x7a	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` returns `EFI_SUCCESS` with non-opened protocol at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.17.6	0x5076952f, 0x17c6, 0x4e8a, 0xb2, 0x49, 0x14, 0x0c, 0xd2, 0x87, 0x82, 0x38	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` uninstalls non-opened protocol at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` from the handle. The handle should still exist.
5.1.3.17.7	0x6caad6f1, 0xe004, 0x45f2, 0x8a, 0x13, 0xd6, 0x3c, 0xe5, 0xb3, 0x36, 0xe7	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` uninstalls non-opened protocol at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` from the handle. The handle should still exist.
5.1.3.17.8	0x797bfd7c, 0xa7ce, 0x4fc7,0x9b, 0xc8, 0x17, 0x17, 0x00, 0x80, 0xd4, 0xdc	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` uninstalls non-opened protocol at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` from the handle. The handle should still exist.
5.1.3.17.9	0x89837cb3, 0x93a0, 0x4b57, 0xbe, 0x97, 0xc7, 0x24, 0x19, 0x09, 0x38, 0x11	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` uninstalls non-opened protocol at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` from the handle. `TestProtocol1` should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.10	0x829c1f46, 0xc17b, 0x4a2d, 0x96, 0x52, 0x56, 0xcc, 0x78, 0x0d, 0xc4, 0xa8	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() uninstalls non-opened protocol at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 from the handle. TestProtocol1 should not exist.
5.1.3.17.11	0x89717ad9, 0x3bec, 0x4ab4, 0xa3, 0x21, 0x5e, 0xac, 0xb9, 0x74, 0xa7, 0x53	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() uninstalls non-opened protocol at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 from the handle. TestProtocol1 should not exist.
5.1.3.17.12	0x90862ff0, 0x93a4, 0x43fe, 0xac, 0x10, 0x4a, 0xf3, 0x39, 0x4d, 0x8f, 0xa4	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() uninstalls non-opened protocol at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 from the handle. TestProtocol1 should not be located from the handle.
5.1.3.17.13	0xf686a16d, 0x8f7d, 0x419d, 0x85, 0x21, 0x77, 0xda, 0x3f, 0x76, 0x6d, 0x73	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() uninstalls non-opened protocol at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 from the handle. TestProtocol1 should not be located from the handle.

Number	GUID	Assertion	Test Description
5.1.3.17.14	0xf95014de, 0x823b, 0x47a0, 0x90, 0x90, 0xeb, 0x8a, 0xdd, 0x95, 0x6f, 0x8d	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() uninstalls non-opened protocol at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 from the handle. TestProtocol1 should not be located from the handle.
5.1.3.17.15	0xeecfa186, 0xb839, 0x4dd2, 0x90, 0x52, 0x15, 0xb5, 0x08, 0x86, 0x10, 0x0a	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() uninstalls non-opened protocol at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 from the handle. TestProtocol2 should still exist on the handle.
5.1.3.17.16	0x2d914b4e, 0xe621, 0x4b8e, 0x89, 0xdf, 0x1b, 0x20, 0x65, 0x63, 0x7d, 0x11	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() uninstalls non-opened protocol at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 from the handle. TestProtocol2 should still exist on the handle.
5.1.3.17.17	0xe854db23, 0x0e8d, 0x436e, 0x92, 0x89, 0xe2, 0xae, 0x58, 0xa6, 0xd6, 0x83	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() uninstalls non-opened protocol at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 from the handle. TestProtocol2 should still exist on the handle.

Number	GUID	Assertion	Test Description
5.1.3.17.18	0x2d0ec682, 0xe6b7, 0x46e5, 0x8e, 0x23, 0x40, 0xfd, 0x1b, 0x22, 0x46, 0x0a	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() returns EFI_SUCCESS with all protocols at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 and TestProtocol2 from the handle. The return code should be EFI_SUCCESS.
5.1.3.17.19	0x182f395c, 0x92a9, 0x4122, 0xae, 0x28, 0x91, 0xd1, 0x57, 0xd6, 0x0a, 0x0e	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() returns EFI_SUCCESS with all protocols at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 and TestProtocol2 from the handle. The return code should be EFI_SUCCESS.
5.1.3.17.20	0x0eafb9e0, 0xfab2, 0x4a07, 0x95, 0xf0, 0x42, 0x61, 0xaa, 0x7a, 0xdb, 0x43	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() returns EFI_SUCCESS with all protocols at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 and TestProtocol2 from the handle. The return code should be EFI_SUCCESS.
5.1.3.17.21	0x63dd3860, 0x4f05, 0x4f97, 0xa8, 0x2c, 0xca, 0xfa, 0xfc, 0x25, 0xc0, 0x19	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() uninstalls all protocols at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 and TestProtocol2 from the handle. The handle should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.22	0x3ee0e86e, 0xcbae, 0x46d2, 0x95, 0x74, 0x23, 0x1f, 0x68, 0xc8, 0xeb, 0xa6	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** uninstalls all protocols at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** and **TestProtocol2** from the handle. The handle should not exist.
5.1.3.17.23	0xab66814a, 0x96ca, 0x4bd6, 0xb7, 0x3b, 0x72, 0x64, 0x9a, 0xc7, 0x98, 0x2e	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** uninstalls all protocols at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** and **TestProtocol2** from the handle. The handle should not exist.
5.1.3.17.24	0xabdfff35, 0x3c96, 0x4fc3, 0x96, 0xe2, 0x45, 0x84, 0x30, 0x20, 0xb2, 0xb4	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** uninstalls all protocols at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** and **TestProtocol2** from the handle. **TestProtocol1** should not exist.
5.1.3.17.25	0xb21f77dc, 0x6bab, 0x4be6, 0x83, 0xa1, 0xaa, 0xfb, 0x6b, 0x58, 0xa3, 0xaa	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** uninstalls all protocols at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** and **TestProtocol2** from the handle. **TestProtocol1** should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.26	0x7ce55ebf, 0x02d4, 0x41fb, 0x89, 0xcd, 0x68, 0xae, 0xbe, 0x73, 0xd9, 0x8c	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** uninstalls all protocols at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** and **TestProtocol2** from the handle. **TestProtocol1** should not exist.
5.1.3.17.27	0x0f0c7f75, 0x6373, 0x4a9e, 0x82, 0xfa, 0x63, 0x8d, 0x18, 0xad, 0x8d, 0x5f	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** uninstalls all protocols at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** and **TestProtocol2** from the handle. **TestProtocol2** should not exist.
5.1.3.17.28	0x8dc31981, 0xd08f, 0x45bf, 0xa1, 0xb0, 0xcd, 0xdb, 0xca, 0x1f, 0x23, 0x03	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** uninstalls all protocols at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** and **TestProtocol2** from the handle. **TestProtocol2** should not exist.
5.1.3.17.29	0x21f85a43, 0x2402, 0x45b1, 0xa6, 0x2a, 0x52, 0x07, 0x5b, 0x09, 0xfa, 0x75	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** uninstalls all protocols at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** and **TestProtocol2** from the handle. **TestProtocol2** should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.30	0xdb5ad6f9, 0xeda1, 0x4c61, 0xa8, 0x9c, 0xc5, 0x4b, 0x1e, 0xe2, 0xc2, 0x4c	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` returns `EFI_SUCCESS` with opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.17.31	0x8b8801d0, 0xe0b2, 0x41f3, 0xab, 0x90, 0xb1, 0xe2, 0xdc, 0xd5, 0xd2, 0x9b	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` returns `EFI_SUCCESS` with opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.17.32	0x5e941370, 0xd65c, 0x4f5a, 0xa1, 0x63, 0x98, 0x26, 0xd7, 0x4a, 0x2a, 0x43	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` returns `EFI_SUCCESS` with opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.17.33	0x9e0fa47a, 0x1038, 0x48f9, 0xac, 0x67, 0x64, 0x00, 0x76, 0xc7, 0xca, 0xa3	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The handle should not exist.
5.1.3.17.34	0xa5d03ea1, 0xd059, 0x436b, 0x9d, 0xd4, 0xf9, 0x3b, 0xf6, 0xe8, 0xc5, 0xcf	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The handle should not exist.
5.1.3.17.35	0xe9020be2, 0x07cb, 0x49c2, 0x92, 0x60, 0x72, 0xf3, 0x03, 0xac, 0x2c, 0xd5	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The handle should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.36	0xdd900c24, 0xcafa, 0x43ae, 0xa2, 0xdd, 0x3d, 0x6b, 0xc8, 0x9c, 0x75, 0x0a	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened BY_HANDLE_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. TestProtocol1 should not exist.
5.1.3.17.37	0xd4edb27f, 0x6ba2, 0x485c, 0x85, 0xc1, 0x5b, 0x61, 0xb7, 0x70, 0xc2, 0x7e	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened BY_HANDLE_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. TestProtocol1 should not exist.
5.1.3.17.38	0xb29b4a3b, 0x7aa3, 0x4840, 0x80, 0xc5, 0x18, 0xd8, 0x72, 0x56, 0xe6, 0x69	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened BY_HANDLE_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_HANDLE_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. TestProtocol1 should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.39	0x1366ce7c, 0xc588, 0x4e13, 0x91, 0x1d, 0x56, 0xb9, 0x2b, 0x24, 0x56, 0x45	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol2` should not exist.
5.1.3.17.40	0xb9f4ddf8, 0x388a, 0x48df, 0xb6, 0x13, 0x1f, 0xf9, 0x57, 0x70, 0x2e, 0x71	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol2` should not exist.
5.1.3.17.41	0x33dfbc47, 0xe974, 0x404e, 0xa0, 0x55, 0x5b, 0x7c, 0x06, 0x84, 0x7a, 0x95	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_HANDLE_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_HANDLE_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol2` should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.42	0x04f5c8a0, 0xfb6d, 0x4bff, 0x85, 0x13, 0x62, 0xfc, 0x36, 0x3d, 0xca, 0x6b	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with opened **GET_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **GET_PROTOCOL**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The return code should be **EFI_SUCCESS**.
5.1.3.17.43	0x55675511, 0x86c1, 0x4605, 0x85, 0xd4, 0xd5, 0x08, 0x0d, 0x7e, 0xe5, 0xc1	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with opened **GET_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **GET_PROTOCOL**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The return code should be **EFI_SUCCESS**.
5.1.3.17.44	0x4a756cdd, 0x2034, 0x48be, 0x91, 0xd5, 0xb1, 0x39, 0x3c, 0xf4, 0x17, 0xeb	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with opened **GET_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **GET_PROTOCOL**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.17.45	0xbed332bb, 0x7e6f, 0x4484, 0xb7, 0x68, 0x92, 0xe0, 0x2f, 0x03, 0x1c, 0x2e	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened GET_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The handle should not exist.
5.1.3.17.46	0x7f3e829a, 0x8aa8, 0x4f54, 0x91, 0x11, 0x2f, 0xa8, 0xfa, 0xce, 0xca, 0xae	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened GET_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The handle should not exist.
5.1.3.17.47	0xbbe591cc, 0xc1f8, 0x44ac, 0x96, 0x4d, 0xec, 0x95, 0x55, 0x60, 0x92, 0x04	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened GET_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The handle should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.48	0xe29553ba, 0xff64, 0x4c70, 0xa5, 0x8b, 0x7e, 0xcd, 0x35, 0xe6, 0x3c, 0x8b	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `GET_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should not exist.
5.1.3.17.49	0x81a05ca7, 0x53a2, 0x4cea, 0x9b, 0x83, 0x47, 0xa7, 0x01, 0xbd, 0x0b, 0x88	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `GET_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should not exist.
5.1.3.17.50	0xb497e879, 0x7273, 0x4827, 0xb1, 0x7c, 0x12, 0x09, 0x27, 0xfd, 0x65, 0x75	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `GET_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `GET_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.51	0x82d2a7f1, 0x6b7e, 0x475e, 0xa1, 0x55, 0x79, 0x38, 0xb1, 0xda, 0xae, 0x25	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened GET_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. TestProtocol2 should not exist.
5.1.3.17.52	0x5f578aa8, 0x74c0, 0x4cba, 0xbc, 0x0e, 0x38, 0x8a, 0x71, 0xf8, 0xc7, 0xd3	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened GET_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. TestProtocol2 should not exist.
5.1.3.17.53	0xc3e5a292, 0xb6fc, 0x41ff, 0xba, 0x39, 0xbe, 0xbc, 0x39, 0x13, 0xdb, 0x00	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened GET_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 GET_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. TestProtocol2 should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.54	0x6c67d8c2, 0x38f5, 0x4674, 0xb2, 0x88, 0x12, 0x63, 0x23, 0x84, 0x21, 0x84	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with opened **TEST_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 TEST_PROTOCOL**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The return code should be **EFI_SUCCESS**.
5.1.3.17.55	0xeb211a93, 0xa179, 0x4894, 0xb4, 0x6b, 0x47, 0xc8, 0xce, 0xe3, 0x1d, 0xff	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with opened **TEST_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 TEST_PROTOCOL**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The return code should be **EFI_SUCCESS**.
5.1.3.17.56	0x0025c42e, 0x8a4f, 0x4dc5, 0x83, 0xe1, 0xf5, 0x1a, 0xe5, 0x7a, 0x4a, 0xaf	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with opened **TEST_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 TEST_PROTOCOL**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.17.57	0x40abad92, 0x6ce5, 0x4caa, 0xad, 0xa1, 0x49, 0x7c, 0x8c, 0xb0, 0x18, 0xd9	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened TEST_PROTOCOL at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The handle should not exist.
5.1.3.17.58	0xa6a482ae, 0x9a8a, 0x4ace, 0x89, 0x24, 0x50, 0x40, 0x5b, 0xb8, 0x92, 0x7b	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened TEST_PROTOCOL at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The handle should not exist.
5.1.3.17.59	0x88ac2d9d, 0x7d4d, 0x4ca3, 0x94, 0x39, 0x54, 0x6d, 0x63, 0x0a, 0x67, 0x07	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened TEST_PROTOCOL at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 TEST_PROTOCOL. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The handle should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.60	0xb325707b, 0x0e09, 0x4315, 0xad, 0x51, 0x71, 0xe9, 0x61, 0x60, 0x2a, 0xdd	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `TEST_PROTOCOL` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should not exist.
5.1.3.17.61	0x624ec4ef, 0x1715, 0x47c4, 0xa4, 0xcb, 0x14, 0x10, 0x12, 0xd7, 0x56, 0x76	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `TEST_PROTOCOL` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should not exist.
5.1.3.17.62	0x2678e3eb, 0xd510, 0x4632, 0x9e, 0xd7, 0xc1, 0xba, 0xd3, 0x12, 0x94, 0x04	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `TEST_PROTOCOL` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `TEST_PROTOCOL`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.63	0x9f6a0688, 0xe31b, 0x4df6, 0x8d, 0x7c, 0x91, 0xef, 0x8f, 0xb4, 0xae, 0xfa	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **TEST_PROTOCOL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 TEST_PROTOCOL**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. **TestProtocol2** should not exist.
5.1.3.17.64	0xda7d27db, 0xa358, 0x4f49, 0xb1, 0x24, 0x90, 0x97, 0x53, 0xe1, 0xe6, 0xda	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **TEST_PROTOCOL** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 TEST_PROTOCOL**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. **TestProtocol2** should not exist.
5.1.3.17.65	0xa0b02f70, 0xdc35, 0x49dc, 0x94, 0x3a, 0xe6, 0xe4, 0xe7, 0x7a, 0x0f, 0x40	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **TEST_PROTOCOL** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 TEST_PROTOCOL**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. **TestProtocol2** should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.66	0x6d5d96e5, 0x87a3, 0x4fe3, 0x86, 0xcb, 0x89, 0x7f, 0x48, 0xae, 0x39, 0x06	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() returns EFI_ACCESS_DENIED with opened BY_CHILD_CONTROLLER at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The return code should be EFI_ACCESS_DENIED.
5.1.3.17.67	0x87af92f4, 0x0886, 0x42bd, 0x9a, 0xfe, 0xb7, 0x3e, 0x56, 0xbd, 0x71, 0x88	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() returns EFI_ACCESS_DENIED with opened BY_CHILD_CONTROLLER at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The return code should be EFI_ACCESS_DENIED.
5.1.3.17.68	0x0767027f, 0xa432, 0x4a7f, 0xa3, 0xb6, 0xd8, 0x9d, 0xdd, 0x68, 0x6e, 0xe8	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() returns EFI_ACCESS_DENIED with opened BY_CHILD_CONTROLLER at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.17.69	0xc1088f51, 0x8698, 0x4315, 0x81, 0x7d, 0xd0, 0x6b, 0xbd, 0x7a, 0xca, 0x99	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened BY_CHILD_CONTROLLER at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The handle should still exist.
5.1.3.17.70	0x0126d268, 0x232e, 0x4d9c, 0xb4, 0x8e, 0xc5, 0xef, 0x56, 0x2e, 0x19, 0x25	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened BY_CHILD_CONTROLLER at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The handle should still exist.
5.1.3.17.71	0x59913cd8, 0xb53a, 0x4854, 0xa6, 0x4d, 0x9f, 0x98, 0xd2, 0x1a, 0x1a, 0xa6	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened BY_CHILD_CONTROLLER at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_CHILD_CONTROLLER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The handle should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.72	0xd33680d1, 0xc401, 0x4439, 0xac, 0xde, 0x5b, 0xb1, 0xa2, 0xda, 0xf6, 0x95	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should still exist.
5.1.3.17.73	0x9ecbe3f6, 0x5c1e, 0x472d, 0x86, 0x22, 0xff, 0x1c, 0x8f, 0xcf, 0xbe, 0x6a	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should still exist.
5.1.3.17.74	0x00f7a9f3, 0x5910, 0x4fea, 0x87, 0xd1, 0xf0, 0x80, 0xaa, 0x2b, 0x7b, 0x56	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.75	0xe44995b9, 0x2c57, 0x4f99, 0x82, 0xa5, 0xb9, 0xee, 0xc7, 0x18, 0xcd, 0x79	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol2` should still exist.
5.1.3.17.76	0xc5f403a8, 0x06a1, 0x49d1, 0x86, 0x1f, 0x4c, 0xa7, 0x4b, 0x4f, 0x45, 0x44	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol2` should still exist.
5.1.3.17.77	0x7538063b, 0x1934, 0x4408, 0x87, 0x33, 0x57, 0xf1, 0xb6, 0x54, 0x33, 0x47	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol2` should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.78	0x285ea572, 0xbede, 0x4238, 0x85, 0xd6, 0x6c, 0x71, 0x0c, 0x3f, 0xcc, 0x28	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.17.79	0x059b49dc, 0x7694, 0x441c, 0xa8, 0xa2, 0xe3, 0xd0, 0x31, 0xcd, 0x82, 0xa0	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.17.80	0x1fa7aa80, 0x84d2, 0x4eb5, 0xb7, 0xcb, 0x0f, 0xe2, 0x41, 0x5b, 0x31, 0x30	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_CHILD_CONTROLLER` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_CHILD_CONTROLLER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.17.81	0x6af7091b, 0x2db6, 0x4f09, 0xa1, 0xfe, 0xdd, 0x5e, 0x87, 0xf4, 0x82, 0xbb	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` returns `EFI_ACCESS_DENIED` with opened `EXCLUSIVE` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_ACCESS_DENIED`.
5.1.3.17.82	0xf589893d, 0x3d46, 0x4be3, 0xaa, 0x9a, 0x42, 0x1e, 0x3d, 0xcd, 0xfd, 0x35	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` returns `EFI_ACCESS_DENIED` with opened `EXCLUSIVE` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_ACCESS_DENIED`.

Number	GUID	Assertion	Test Description
5.1.3.17.83	0xe05ca4d7, 0xa705, 0x4270, 0x99, 0xbb, 0x10, 0x8d, 0x8c, 0x1f, 0xc8, 0x0c	**BS.UninstallMultiplePr otocolInterfaces – UninstallMultipleProto colInterfaces()** returns **EFI_ACCESS_DENIED** with opened **EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The return code should be **EFI_ACCESS_DENIED**.
5.1.3.17.84	0x621782bb, 0x2da2, 0x4344, 0xae, 0x2b, 0x69, 0xc0, 0xe8, 0xe6, 0x8f, 0xdf	**BS.UninstallMultiplePr otocolInterfaces – UninstallMultipleProto colInterfaces()** uninstalls opened **EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The handle should still exist.
5.1.3.17.85	0x28749f75, 0xc7c3, 0x4e55, 0xbc, 0xa1, 0xb2, 0xfb, 0x80, 0x77, 0x26, 0x0c	**BS.UninstallMultiplePr otocolInterfaces – UninstallMultipleProto colInterfaces()** uninstalls opened **EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The handle should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.86	0x193a9bdd, 0x6b07, 0x44e7, 0xb6, 0x53, 0x60, 0x42, 0x78, 0xca, 0xdb, 0x1a	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `EXCLUSIVE` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The handle should still exist.
5.1.3.17.87	0x5460bae6, 0x94af, 0x4bd9, 0x97, 0x8f, 0x46, 0x71, 0xda, 0x2a, 0x63, 0xa5	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `EXCLUSIVE` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should still exist.
5.1.3.17.88	0x748b6ed2, 0xf1f7, 0x4b40, 0xaa, 0x7e, 0xc0, 0xbc, 0xfc, 0x25, 0x28, 0x5e	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `EXCLUSIVE` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.89	0xbb86b8cd, 0x124e, 0x4bde, 0x89, 0xa6, 0xe3, 0xc7, 0x8d, 0x12, 0x48, 0x2b	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. **TestProtocol1** should still exist.
5.1.3.17.90	0xf800d1fe, 0xb548, 0x4d37, 0xb0, 0x22, 0x1e, 0x45, 0xd7, 0xe2, 0xae, 0xb0	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. **TestProtocol2** should still exist.
5.1.3.17.91	0xc2ab2631, 0x012d, 0x4d14, 0x81, 0x4f, 0x1c, 0xda, 0xf2, 0xa6, 0x3b, 0xfa	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. **TestProtocol2** should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.92	0xd995de48, 0xe12e, 0x4854, 0x86, 0x6c, 0x59, 0xd2, 0xf7, 0x6f, 0x6e, 0xb0	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `EXCLUSIVE` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol2` should still exist.
5.1.3.17.93	0x5c04c757, 0x9313, 0x4afa, 0xaf, 0x23, 0xe9, 0xae, 0x6f, 0x74, 0x28, 0xc5	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `EXCLUSIVE` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.17.94	0xb72382d7, 0xb6c7, 0x4532, 0x97, 0x7c, 0x6b, 0xfc, 0xe0, 0x42, 0xe4, 0xcc	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. 4. Call **CloseProtocol()** to close **TestProtocol1**. 5. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The return code should be **EFI_SUCCESS**.
5.1.3.17.95	0x09522d19, 0x6020, 0x4b2e, 0xa9, 0x64, 0xe0, 0x39, 0xf5, 0xfd, 0x36, 0x10	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. 4. Call **CloseProtocol()** to close **TestProtocol1**. 5. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.17.96	0x0ffd3c72, 0xe720, 0x4181, 0x88, 0x15, 0x3a, 0x7e, 0x68, 0x83, 0x9c, 0x1c	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() returns EFI_ACCESS_DENIED with opened BY_DRIVER at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The return code should be EFI_ACCESS_DENIED.
5.1.3.17.97	0xd6a17500, 0x9dcd, 0x48e3, 0xa1, 0x60, 0x81, 0x09, 0x53, 0xb8, 0x2f, 0x24	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() returns EFI_ACCESS_DENIED with opened BY_DRIVER at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The return code should be EFI_ACCESS_DENIED.
5.1.3.17.98	0x1e4e4e42, 0x9a65, 0x4780, 0x84, 0x8b, 0x0f, 0xd2, 0xe5, 0xc1, 0x77, 0x9a	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() returns EFI_ACCESS_DENIED with opened BY_DRIVER at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.1.3.17.99	0x7b51f145, 0x4444, 0x49a2, 0xaf, 0x26, 0xc5, 0x98, 0xd9, 0xee, 0x18, 0x65	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The handle should still exist.
5.1.3.17.100	0x6963ae6e, 0x0740, 0x4bae, 0x8c, 0x2a, 0xe6, 0x99, 0x13, 0xbe, 0x2b, 0x40	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The handle should still exist.
5.1.3.17.101	0x98baf1ed, 0xb864, 0x4858, 0x89, 0x55, 0x39, 0x39, 0x6e, 0x94, 0x04, 0x09	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The handle should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.102	0xe6c1e016, 0x6faf, 0x4ee0, 0x83, 0xa9, 0x7d, 0x73, 0x5c, 0x3f, 0x4b, 0xbc	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened BY_DRIVER at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. TestProtocol1 should still exist.
5.1.3.17.103	0x10205361, 0x03c6, 0x4c8a, 0x89, 0x53, 0x8d, 0x8f, 0xc0, 0x00, 0xac, 0x4a	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened BY_DRIVER at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. TestProtocol1 should still exist.
5.1.3.17.104	0x6fbe1f14, 0xe6f5, 0x4e57, 0x95, 0xd5, 0xa4, 0x6d, 0xd9, 0x86, 0x76, 0x3f	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened BY_DRIVER at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. TestProtocol1 should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.105	0xe1b6ee4c, 0x79a9, 0x432d, 0xb7, 0xda, 0x68, 0x57, 0x05, 0xf0, 0x4d, 0x13	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol2` should still exist.
5.1.3.17.106	0xdfb2e951, 0xc3d8, 0x4f27, 0x87, 0x9d, 0xfc, 0xd6, 0x1a, 0x6d, 0x77, 0xe9	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol2` should still exist.
5.1.3.17.107	0xeb1621e3, 0x498e, 0x4b15, 0x82, 0xc5, 0x7b, 0x91, 0x71, 0xb5, 0xd0, 0x0a	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol2` should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.108	0x4bc1f888, 0xad45, 0x4708, 0xb6, 0x5d, 0xde, 0x51, 0xa7, 0x0d, 0xb8, 0xd2	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.17.109	0x458919a9, 0x41a3, 0x47a5, 0xa0, 0x90, 0xbd, 0xaf, 0xd2, 0x14, 0x1a, 0x59	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.17.110	0xdea8772d, 0x6898, 0x4605, 0x8e, 0x7b, 0xc1, 0x84, 0x08, 0x03, 0xbf, 0x95	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.17.111	0x9d70878c, 0xfe99, 0x47a1, 0xae, 0x69, 0x74, 0x26, 0x67, 0x71, 0x72, 0x59	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` returns `EFI_ACCESS_DENIED` with opened `BY_DRIVER | EXCLUSIVE` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER | EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_ACCESS_DENIED`.

Number	GUID	Assertion	Test Description
5.1.3.17.112	0x52490623, 0x3656, 0x4885, 0x8d, 0xed, 0x03, 0xa3, 0x3e, 0x51, 0xe6, 0x45	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() returns EFI_ACCESS_DENIED with opened BY_DRIVER | EXCLUSIVE at EFI_TPL_CALLBACK	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The return code should be EFI_ACCESS_DENIED.
5.1.3.17.113	0xb68e1e7c, 0x84a7, 0x4f2f, 0xbc, 0x6f, 0x21, 0x44, 0xf9, 0x6a, 0x06, 0xb5	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() returns EFI_ACCESS_DENIED with opened BY_DRIVER | EXCLUSIVE at EFI_TPL_NOTIFY	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The return code should be EFI_ACCESS_DENIED.
5.1.3.17.114	0x73a6e8ac, 0xd67e, 0x41bd, 0xad, 0x5b, 0x1b, 0xca, 0x32, 0x67, 0xda, 0x67	BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() uninstalls opened BY_DRIVER | EXCLUSIVE at EFI_TPL_APPLICATION	1. Install TestProtocol1 & TestProtocol2 onto new handle. 2. Call OpenProtocol() to open TestProtocol1 BY_DRIVER | EXCLUSIVE. 3. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocol1 & TestProtocol2 from the handle. The handle should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.115	0x06c1eafd, 0xf83a, 0x4a77, 0x90, 0x9b, 0xfb, 0x44, 0x53, 0x9b, 0x2f, 0xfe	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER | EXCLUSIVE` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER | EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The handle should still exist.
5.1.3.17.116	0x24822324, 0xbd2e, 0x4487, 0xbc, 0x9b, 0x85, 0x36, 0x15, 0xb7, 0xaf, 0xb5	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER | EXCLUSIVE` at `EFI_TPL_NOTIFY`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER | EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The handle should still exist.
5.1.3.17.117	0x190a11f5, 0x10ab, 0x40c3, 0x98, 0x19, 0x79, 0x75, 0xc3, 0x5f, 0xe6, 0xdd	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER | EXCLUSIVE` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER | EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. `TestProtocol1` should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.118	0x22f6d0c0, 0xf42f, 0x4867, 0x88, 0x75, 0xdd, 0x3f, 0x8d, 0x77, 0x8e, 0x22	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER | EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. **TestProtocol1** should still exist.
5.1.3.17.119	0x6b48156e, 0x6adc, 0x4ba7, 0xbd, 0x5b, 0xc4, 0x83, 0x08, 0x37, 0x28, 0x50	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER | EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. **TestProtocol1** should still exist.
5.1.3.17.120	0x0705d119, 0x04b6, 0x4cfa,0x9e, 0x1e, 0x00, 0x4e, 0xd0, 0x54, 0xd9, 0x05	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_APPLICATION**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER | EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. **TestProtocol2** should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.121	0x111c2fe1, 0x1c44, 0x42c8, 0x88, 0x76, 0x48, 0x0f, 0xd3, 0x0c, 0xa1, 0x5a	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_CALLBACK**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **BY_DRIVER | EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. **TestProtocol2** should still exist.
5.1.3.17.122	0x132ccf99, 0x64f8, 0x4d31, 0xa5, 0x46, 0x36, 0xde, 0x50, 0xdf, 0xb1, 0xbc	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1** **BY_DRIVER | EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. **TestProtocol2** should still exist.

Number	GUID	Assertion	Test Description
5.1.3.17.123	0x0670739d, 0xf6a6, 0x4cb6, 0xa4, 0x22, 0xb8, 0xd6, 0xed, 0x2e, 0x53, 0xb2	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER | EXCLUSIVE` at `EFI_TPL_APPLICATION`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER | EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.
5.1.3.17.124	0xcf9ddc59, 0x3d57, 0x4dfe, 0xa6, 0x3a, 0x51, 0x3d, 0x26, 0x14, 0x0e, 0xa8	`BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()` uninstalls opened `BY_DRIVER | EXCLUSIVE` at `EFI_TPL_CALLBACK`	1. Install `TestProtocol1` & `TestProtocol2` onto new handle. 2. Call `OpenProtocol()` to open `TestProtocol1` `BY_DRIVER | EXCLUSIVE`. 3. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. 4. Call `CloseProtocol()` to close `TestProtocol1`. 5. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocol1` & `TestProtocol2` from the handle. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.3.17.125	0x7031defc, 0xdaba, 0x48ab, 0x80, 0x84, 0x34, 0xf3, 0xbd, 0xd8, 0xff, 0x8e	**BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces()** uninstalls opened **BY_DRIVER | EXCLUSIVE** at **EFI_TPL_NOTIFY**	1. Install **TestProtocol1** & **TestProtocol2** onto new handle. 2. Call **OpenProtocol()** to open **TestProtocol1 BY_DRIVER | EXCLUSIVE**. 3. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. 4. Call **CloseProtocol()** to close **TestProtocol1**. 5. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocol1** & **TestProtocol2** from the handle. The return code should be **EFI_SUCCESS**.
5.1.3.17.126	0x49245471, 0xcd0c, 0x4b67, 0x86, 0x2e, 0x40, 0xdf, 0x7b, 0x7e, 0xa5, 0x2d	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** returns **EFI_SUCCESS** with two **NULL** at **EFI_TPL_APPLICATION**	1. Install **TestProtocolNoInterface1** & **TestProtocolNoInterface2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocolNoInterface1** & **TestProtocolNoInterface2** from the handle. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.3.17.127	0x4d809155, 0xadba, 0x425d, 0x89, 0x0a, 0x03, 0xbc, 0x2d, 0xfb, 0x91, 0x58	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() returns EFI_SUCCESS with two NULL at EFI_TPL_CALLBACK	1. Install TestProtocolNoInterface1 & TestProtocolNoInterface2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocolNoInterface1 & TestProtocolNoInterface2 from the handle. The return code should be EFI_SUCCESS.
5.1.3.17.128	0x9e5bb648, 0xec5f, 0x4fb5, 0xad, 0x5f, 0xcf, 0xc1, 0x36, 0x56, 0xbc, 0xd2	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() returns EFI_SUCCESS with two NULL at EFI_TPL_NOTIFY	1. Install TestProtocolNoInterface1 & TestProtocolNoInterface2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocolNoInterface1 & TestProtocolNoInterface2 from the handle. The return code should be EFI_SUCCESS.
5.1.3.17.129	0xb4aedbe9, 0xa3bf, 0x4a57, 0x99, 0x35, 0x27, 0xed, 0x5b, 0xd1, 0x74, 0xc9	BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces() uninstalls two NULL interfaces at EFI_TPL_APPLICATION	1. Install TestProtocolNoInterface1 & TestProtocolNoInterface2 onto new handle. 2. Call UnInstallMultipleProtocolInterfaces() to remove TestProtocolNoInterface1 & TestProtocolNoInterface2 from the handle. The handle should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.130	0x1471a8dd, 0x6290, 0x429f, 0x8e, 0xe0, 0x6c, 0x96, 0xb7, 0xcb, 0x17, 0x62	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` uninstalls two `NULL` interfaces at `EFI_TPL_CALLBACK`	1. Install `TestProtocolNoInterface1` & `TestProtocolNoInterface2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocolNoInterface1` & `TestProtocolNoInterface2` from the handle. The handle should not exist.
5.1.3.17.131	0x05142fe9, 0x964e, 0x47fd, 0x80, 0xdf, 0x99, 0x0c, 0x12, 0x56, 0x79, 0x2c	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` uninstalls two `NULL` interfaces at `EFI_TPL_NOTIFY`	1. Install `TestProtocolNoInterface1` & `TestProtocolNoInterface2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocolNoInterface1` & `TestProtocolNoInterface2` from the handle. The handle should not exist.
5.1.3.17.132	0x5bf9b76d, 0x543e, 0x43e5, 0xae, 0x72, 0x70, 0xaa, 0x21, 0x0b, 0x7f, 0x51	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` uninstalls two `NULL` interfaces at `EFI_TPL_APPLICATION`	1. Install `TestProtocolNoInterface1` & `TestProtocolNoInterface2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocolNoInterface1` & `TestProtocolNoInterface2` from the handle. `TestProtocolNoInterface1` should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.133	0x2ec74865, 0x37c0, 0x4c4e, 0xa5, 0x34, 0x9a, 0x95, 0x4c, 0x89, 0x1a, 0xe9	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** uninstalls two **NULL** interfaces at **EFI_TPL_CALLBACK**	1. Install **TestProtocolNoInterface1** & **TestProtocolNoInterface2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocolNoInterface1** & **TestProtocolNoInterface2** from the handle. **TestProtocolNoInterface1** should not exist.
5.1.3.17.134	0x67249190, 0x20dc, 0x460f, 0xbd, 0x71, 0xb1, 0x07, 0xef, 0x0e, 0x1a, 0xaa	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** uninstalls two **NULL** interfaces at **EFI_TPL_NOTIFY**	1. Install **TestProtocolNoInterface1** & **TestProtocolNoInterface2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocolNoInterface1** & **TestProtocolNoInterface2** from the handle. **TestProtocolNoInterface1** should not exist.
5.1.3.17.135	0xc7f4b9f2, 0xc755, 0x4bb4, 0xa2, 0x92, 0xc6, 0xa4, 0x52, 0x91, 0xf8, 0xbd	**BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()** uninstalls two **NULL** interfaces at **EFI_TPL_APPLICATION**	1. Install **TestProtocolNoInterface1** & **TestProtocolNoInterface2** onto new handle. 2. Call **UnInstallMultipleProtocolInterfaces()** to remove **TestProtocolNoInterface1** & **TestProtocolNoInterface2** from the handle. **TestProtocolNoInterface2** should not exist.

Number	GUID	Assertion	Test Description
5.1.3.17.136	0x1e93f309, 0x862d, 0x4add, 0x89, 0xb9, 0xc3, 0xa7, 0x58, 0x61, 0x98, 0x69	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` uninstalls two `NULL` interfaces at `EFI_TPL_CALLBACK`	1. Install `TestProtocolNoInterface1` & `TestProtocolNoInterface2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocolNoInterface1` & `TestProtocolNoInterface2` from the handle. `TestProtocolNoInterface2` should not exist.
5.1.3.17.137	0x445c2395, 0x8bda, 0x4e5e, 0xab, 0x07, 0x82, 0x3b, 0x18, 0x7e, 0x52, 0xd8	`BS.UninstallMultipleProtocolInterfaces – UnInstallMultipleProtocolInterfaces()` uninstalls two `NULL` interfaces at `EFI_TPL_NOTIFY`	1. Install `TestProtocolNoInterface1` & `TestProtocolNoInterface2` onto new handle. 2. Call `UnInstallMultipleProtocolInterfaces()` to remove `TestProtocolNoInterface1` & `TestProtocolNoInterface2` from the handle. `TestProtocolNoInterface2` should not exist.

3.4 Image Services Test

Reference Document:

UEFI Specification, Image Services Section.

Table 4. Image Functions

Name	Type	Description
LoadImage()	Boot	Loads an EFI image into memory.
StartImage()	Boot	Transfers control to a loaded image's entry point.
UnloadImage()	Boot	Unloads an image.
EFI_IMAGE_ENTRY_POINT	Boot	Prototype of an EFI Image's entry point.
Exit()	Boot	Exits the image's entry point.
ExitBootServices()	Boot	Terminates boot services.

3.4.1 LoadImage()

Number	GUID	Assertion	Test Description
5.1.4.1.1	0x8d5f5a0d, 0x225e, 0x4383, 0x9d, 0x14, 0x27, 0x46, 0xd7, 0x48, 0xb7, 0xa3	`BS.LoadImage – LoadImage()` returns `EFI_INVALID_PARAMETER` with invalid `ParentImageHandle`.	1. Call `LoadImage()` with a `ParentImageHandle` value of `NULL` or an invalid image handle, The return code must be `EFI_INVALID_PARAMETER`.
5.1.4.1.2	0xb04da351, 0xe5a5, 0x43a3, 0x88, 0x98, 0x41, 0x37, 0xbb, 0xba, 0x7e, 0x86	`BS.LoadImage – LoadImage()` returns `EFI_INVALID_PARAMETER` with `NULL FilePath`.	1. Call `LoadImage()` with a `FilePath` value of `NULL`, The return code must be `EFI_INVALID_PARAMETER`.
5.1.4.1.3	0x755f66bd, 0xad6e, 0x4fa3, 0xb5, 0xaf, 0xd9, 0xdd, 0x22, 0xa8, 0x38, 0x58	`BS.LoadImage – LoadImage()` returns `EFI_NOT_FOUND` with irrelevant `FilePath`.	1. Call `LoadImage()` with the `FilePath` that could not be parsed to locate the proper protocol for loading the image file. The return code must be `EFI_NOT_FOUND`.
5.1.4.1.4	0x4556a0d5, 0xb928, 0x4777, 0x8e, 0xce, 0x6d, 0xbd, 0x80, 0x88, 0xf8, 0x78	`BS.LoadImage – LoadImage()` returns `EFI_NOT_FOUND` with a non-existent `FilePath`.	1. Call `LoadImage()` with a `FilePath` that actually does not exist in the system. The return code must be `EFI_NOT_FOUND`.
5.1.4.1.5	0xcc78f02e, 0x8b50, 0x4f9d, 0xb2, 0x92, 0x59, 0x10, 0xac, 0x2a, 0x22, 0x02	`BS.LoadImage – LoadImage()` returns `EFI_INVALID_PARAMETER` with `NULL ImageHandle`.	1. Call `LoadImage()` with the `NULL ImageHandle`, The return code must be `EFI_INVALID_PARAMETER`.
5.1.4.1.6	0x279ca318, 0x4859, 0x4c3f, 0xb7, 0x75, 0x06, 0x58, 0x7d, 0xdc, 0x7e, 0x56	`BS.LoadImage – LoadImage()` returns `EFI_LOAD_ERROR` with 0 length `Buffer`.	1. Call `LoadImage()` with the `SourceSize` as 0, The return code must be `EFI_LOAD_ERROR`.

Number	GUID	Assertion	Test Description
5.1.4.1.7	0x2881c2cc, 0x28aa, 0x4335, 0x8a, 0x9f, 0x5c, 0x90, 0x5d, 0x5f, 0x9d, 0xfc	**BS.LoadImage – LoadImage()** loads image from disk device that supports Simple File System Protocol.	1. Create an EFI application, an EFI boot services driver, and an EFI runtime services driver onto disk device. 2. Call **LoadImage()** to load each image. The return code should be **EFI_SUCCESS**.
5.1.4.1.8	0x8bdfd438, 0x06b0, 0x43a6, 0xab, 0x5b, 0x51, 0x83, 0x39, 0xfd, 0x8f, 0x87	**BS.LoadImage – LoadImage()** loads image from disk device that supports Simple File System Protocol.	1. Create an EFI application, an EFI boot services driver, and an EFI runtime services driver onto disk device. 2. Call **LoadImage()** to load each image. **EFI_LOADED_IMAGE_PROTOCOL** should be located from each return **ImageHandle**.
5.1.4.1.9	0xa44b3d57, 0xa2a3, 0x41ee, 0xb5, 0xa3, 0x59, 0x5f, 0xab, 0xfc, 0x5c, 0x76	**BS.LoadImage – LoadImage()** loads image from disk device that supports Simple File System Protocol.	1. Create an EFI application, an EFI boot services driver, and an EFI runtime services driver onto disk device. 2. Call **LoadImage()** to load each image. The memory type of code and data for EFI application must be **EfiLoaderCode** and **EfiLoaderData**. For EFI boot services must be **EfiBootServicesCode** and **EfiBootServicesData**. For EFI runtime services must be **EfiRuntimeServicesCode** and **EfiRuntimeServicesData**.

Number	GUID	Assertion	Test Description
5.1.4.1.10	0x7d5540a9, 0x9bbd, 0x4f33, 0xaf, 0xf3, 0x84, 0xbc, 0xc5, 0xbe, 0x83, 0x0a	`BS.LoadImage – LoadImage()` loads image from memory.	1. Create an EFI application, an EFI boot services driver, and an EFI runtime services driver, and then load them to memory. 2. Call `LoadImage()` to load each image. The return code should be `EFI_SUCCESS`.
5.1.4.1.11	0xb382d195, 0x2231, 0x4c6a, 0xa3, 0x42, 0x3d, 0xde, 0x8f, 0x7c, 0x39, 0xe0	`BS.LoadImage – LoadImage()` loads image from memory.	1. Create an EFI application, an EFI boot services driver, and an EFI runtime services driver, and then load them to memory. 2. Call `LoadImage()` to load each image. `EFI_LOADED_IMAGE_PROTOCOL` should be located from each return `ImageHandle`.
5.1.4.1.12	0xd59292f3, 0x68bd, 0x4b2e, 0xb0, 0xa5, 0x9b, 0x8c, 0x39, 0x52, 0xcf, 0x9e	`BS.LoadImage – LoadImage()` loads image from memory.	1. Create an EFI application, an EFI boot services driver, and an EFI runtime services driver, and then load them to memory. 2. Call `LoadImage()` to load each image. The memory type of code and data for EFI application must be `EfiLoaderCode` and `EfiLoaderData`. For EFI boot services must be `EfiBootServicesCode` and `EfiBootServicesData`. For EFI runtime services must be `EfiRuntimeServicesCode` and `EfiRuntimeServicesData`.

Number	GUID	Assertion	Test Description
5.1.4.1.13	0x1272dcf7, 0xdd42, 0x4f3f, 0x90, 0x55, 0x7d, 0x6f, 0x3e, 0x8b, 0xba, 0x1f	**BS.LoadImage – LoadImage()** ignores **FilePath** with non-**NULL** **SourceBuffer**.	1. Create an EFI application and an EFI boot services driver onto the disk device, and then load the Application to memory. 2. Call **LoadImage()** with a **FilePath** value of the path of the EFI boot services driver, and the **SourceBuffer** to the EFI application's memory. The return code should be **EFI_SUCCESS**.
5.1.4.1.14	0x21759ccc, 0x092c, 0x4a43, 0x8a, 0xcc, 0x8f, 0xa7, 0xb0, 0x69, 0x91, 0x29	**BS.LoadImage – LoadImage()** ignores **FilePath** with non-**NULL** **SourceBuffer**.	1. Create an EFI application and an EFI boot services driver onto the disk device, and then load the Application to memory. 2. Call **LoadImage()** with a **FilePath** value of the path of the EFI boot services driver, and the **SourceBuffer** to the EFI application's memory. **EFI_LOADED_IMAGE_PROTOCOL** should be located from the return **ImageHandle**.
5.1.4.1.15	0x90f0c29a, 0x19f4, 0x4350, 0xa5, 0xc1, 0x1a, 0xe6, 0x9e, 0x45, 0x09, 0xaf	**BS.LoadImage – LoadImage()** ignores **FilePath** with non-**NULL** **SourceBuffer**.	1. Create an EFI application and an EFI boot services driver onto the disk device, and then load the Application to memory. 2. Call **LoadImage()** with a **FilePath** value of the path of the EFI boot services driver, and the **SourceBuffer** to the EFI application's memory. The memory type of code and data should be **EfiLoaderCode** and **EfiLoaderData**.

Number	GUID	Assertion	Test Description
5.1.4.1.16	0xfc86a302, 0xd59b, 0x4f58, 0x9f, 0x8f, 0x83, 0xab, 0x31, 0x4c, 0x5f, 0x0a	**`BS.LoadImage – LoadImage()`** does not return **`EFI_SUCCESS`** with corrupt image file.	1. Call **`LoadImage()`** with the images whose format was corrupt or not understood by the EFI loader. The return code should not be **`EFI_SUCCESS`**.
5.1.4.1.17	0xb51a788f, 0xa7f1, 0x4332, 0x9b, 0xaf, 0x64, 0xe6, 0x4d, 0x74, 0x42, 0xd9	**`BS.LoadImage – LoadImage()`** returns **`EFI_OUT_OF_RESOURCES`** with very large image.	1. Call **`LoadImage()`** with a very large image. The return code should be **`EFI_OUT_OF_RESOURCES`**.
5.1.4.1.18	0x37126638, 0x5217, 0x4f39, 0x9d, 0x82, 0x40, 0xa3, 0x74, 0xb5, 0x74, 0xf6	**`BS.LoadImage – LoadImage()`** loads image via **`EFI_LOAD_FILE_PROTOCOL`**.	1. Create a **`EFI_LOAD_FILE_PROTOCOL`** in a test driver and start it. 2. Create three device paths related to the **`EFI_LOAD_FILE_PROTOCOL`** and bind with an EFI application, an EFI boot services driver, and an EFI runtime services driver. 3. Call **`LoadImage()`** to load those images. The return code should be **`EFI_SUCCESS`**.
5.1.4.1.19	0x0c0a89fc, 0x9b1f, 0x443a, 0xb0, 0x62, 0x5a, 0xfa, 0xb5, 0x19, 0xac, 0x12	**`BS.LoadImage – LoadImage()`** loads image via **`EFI_LOAD_FILE_PROTOCOL`**.	1. Create a **`EFI_LOAD_FILE_PROTOCOL`** in a test driver and start it. 2. Create three device paths related to the **`EFI_LOAD_FILE_PROTOCOL`** and bind with an EFI application, an EFI boot services driver, and an EFI runtime services driver. 3. Call **`LoadImage()`** to load those images. **`EFI_LOADED_IMAGE_PROTOCOL`** should be located from the image handle.

Number	GUID	Assertion	Test Description
5.1.4.1.20	0x55383e9d, 0xc035, 0x4b36, 0x93, 0x9e, 0xb5, 0x6b, 0x1e, 0x81, 0xdc, 0xb9	BS.LoadImage – LoadImage() loads image via EFI_LOAD_FILE_PROTOCOL .	1. Create a EFI_LOAD_FILE_PROTOCOL in a test driver and start it. 2. Create three device paths related to the EFI_LOAD_FILE_PROTOCOL and bind with an EFI application, an EFI boot services driver, and an EFI runtime services driver. 3. Call LoadImage() to load those images. The memory type of code and data for EFI application must be EfiLoaderCode and EfiLoaderData. For EFI boot services must be EfiBootServicesCode and EfiBootServicesData. For EFI runtime services must be EfiRuntimeServicesCode and EfiRuntimeServicesData.
5.1.4.1.21	0x589fe1c3, 0xf0f3, 0x486e, 0x90, 0x45, 0x3, 0xba, 0x6d, 0xe2, 0x3b, 0x8c	BS.LoadImage - LoadImage() load valid hii image from memory; return code should be EFI_SUCCESS	1. Create a valid hii image and then load it to memory 2. Call LoadImage() to load the image; the return code should be EFI_SUCCESS.
5.1.4.1.22	0x1d8b160c, 0x7601, 0x47c9, 0x81, 0x2, 0x68, 0xc0, 0xf8, 0x1, 0x31, 0x4b	BS.LoadImage - LoadImage() load valid hii image from memory, return code should be EFI_SUCCESS	1. Create a valid hii image and 2. Call LoadImage() to load hii image. EFI_HII_PACKAGE_LIST_PROTOCOL should be installed on ImageHandle.
5.1.4.1.23	0xf5268bb3, 0xff27, 0x492b, 0x91, 0x4f, 0xec, 0x98, 0x20, 0xa2, 0x14, 0xc8	BS.LoadImage - LoadImage() load invalid hii image or Application/BsDriver/RuntimeDriver image from memory; return code should be EFI_SUCCESS	1. Create invalid hii or Application/BsDriver/RuntimeDriver images 2. Call LoadImage() to load each image; the return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.4.1.24	0xa40cacae, 0x81d7, 0x4eb6, 0xad, 0x4f, 0x2e, 0xda, 0x48, 0x92, 0xe1, 0xc	**BS.LoadImage - LoadImage()** Invoke **BS.HandleProtocol()** and verify whether **EFI_HII_PACKAGE_LIST_PROTOCOL** installed on the **ImageHandle**, and the return value should be **EFI_UNSUPPORTED**	1. Verify whether the **ImageHandle** installed on **EFI_HII_PACKAGE_LIST_PROTOCOL** and return value should be **EFI_UNSUPPORTED**.

3.4.2 StartImage()

Number	GUID	Assertion	Test Description
5.1.4.2.1	0x67ba6fae, 0x9758, 0x4edb, 0x9d, 0x4d, 0x1a, 0xe8, 0xc9, 0x82, 0x0f, 0x1e	BS.StartImage – StartImage() returns EFI_INVALID_PARAMETER with invalid ImageHandle.	1. Call StartImage() with NULL or invalid image handle. The return code must be EFI_INVALID_PARAMETER.
5.1.4.2.2	0xb217ffee, 0xac38, 0x4590, 0x92, 0x2b, 0x56, 0x6c, 0x2f, 0xb8, 0x04, 0x7b	BS.StartImage – StartImage() starts an EFI application.	1. Create an EFI application that installs and uninstalls Protocol1, and opens Protocol2. 2. Register a notification for Protocol1's installation. 3. Load and Start the EFI application. The return code should be EFI_SUCCESS.
5.1.4.2.3	0x6999d70b, 0x3226, 0x41c1, 0x85, 0xef, 0x0a, 0x47, 0x31, 0x31, 0xd3, 0x0a	BS.StartImage – StartImage() starts an EFI application.	1. Create an EFI application that installs and uninstalls Protocol1, and opens Protocol2. 2. Register a notification for Protocol1's installation. 3. Load and Start the EFI application. The notify function should be invoked.
5.1.4.2.4	0x63223117, 0x0d3a, 0x468b, 0x8f, 0xb5, 0x1a, 0x8c, 0xbf, 0x51, 0xd6, 0x29	BS.StartImage – StartImage() starts an EFI application.	1. Create an EFI application that installs and uninstalls Protocol1, and opens Protocol2. 2. Register a notification for Protocol1's installation. 3. Load and Start the EFI application. Protocol2 should be opened.
5.1.4.2.5	0x1015f20e, 0x1d8f, 0x4793, 0xa7, 0xbc, 0x3a, 0xff, 0xe7, 0xdd, 0xfb, 0xdc	BS.StartImage – StartImage() starts an EFI boot services driver.	1. Create an EFI boot services driver that installs Protocol1, opens Protocol2, and returns EFI_SUCCESS. 2. Register a notification for Protocol1's installation. 3. Load and Start the EFI boot services driver. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.4.2.6	0x943ddc91, 0xf767, 0x4b77, 0x95, 0x31, 0xc6, 0x30, 0xac, 0xbe, 0xf6, 0x18	**BS.StartImage – StartImage()** starts an EFI boot services driver.	1. Create an EFI boot services driver that installs **Protocol1**, opens **Protocol2**, and returns **EFI_SUCCESS**. 2. Register a notification for **Protocol1**'s installation. 3. Load and Start the EFI boot services driver. The notify function should be invoked.
5.1.4.2.7	0x80c0983a, 0x2ed4, 0x4492, 0xbd, 0x2b, 0x38, 0xa3, 0xaf, 0xa5, 0xde, 0x9e	**BS.StartImage – StartImage()** starts an EFI boot services driver.	1. Create an EFI boot services driver that installs **Protocol1**, opens **Protocol2**, and returns **EFI_SUCCESS**. 2. Register a notification for **Protocol1**'s installation. 3. Load and Start the EFI boot services driver. **Protocol1** should be located.
5.1.4.2.8	0x0c2676e7, 0x66e8, 0x48ea, 0xa9, 0x35, 0x98, 0xd8, 0x25, 0x3f, 0x87, 0xd9	**BS.StartImage – StartImage()** starts an EFI boot services driver.	1. Create an EFI boot services driver that installs **Protocol1**, and open **Protocol2**. 2. Register a notification for **Protocol1**'s installation. 3. Load and Start the EFI boot services driver. **Protocol2** should be opened.
5.1.4.2.9	0x98c88bc2, 0x52c4, 0x41ac, 0xb5, 0xc2, 0x0b, 0xae, 0x7e, 0x13, 0x90, 0xe0	**BS.StartImage – StartImage()** starts an EFI boot services driver.	1. Create an EFI boot services driver that installs **Protocol1**, opens **Protocol2**, and returns **EFI_DEVICE_ERROR**. 2. Register a notification for **Protocol1**'s installation. 3. Load and Start the EFI boot services driver. The return code should be **EFI_DEVICE_ERROR**.

Number	GUID	Assertion	Test Description
5.1.4.2.10	0x9bfcca9b, 0xee53, 0x42a4, 0x98, 0x2a, 0x7b, 0x26, 0x27, 0x28, 0x46, 0xb5	**BS.StartImage – StartImage()** starts an EFI boot services driver.	1. Create an EFI boot services driver that installs **Protocol1**, opens **Protocol2**, and returns **EFI_DEVICE_ERROR**. 2. Register a notification for **Protocol1**'s installation. 3. Load and Start the EFI boot services driver. **Protocol2** should be released the open reference.
5.1.4.2.11	0x3298c357, 0xee05, 0x46c6, 0x89, 0x1f, 0xa7, 0xc9, 0xd6, 0x5e, 0x24, 0xfe	**BS.StartImage – StartImage()** starts an EFI boot services driver.	1. Create an EFI boot services driver that installs **Protocol1**, opens **Protocol2**, and returns **EFI_DEVICE_ERROR**. The driver exits with **ExitData**. 2. Register a notification for **Protocol1**'s installation. 3. Load and Start the EFI boot services driver. **ExitData** returned by **StartImage()** should be not **NULL**.
5.1.4.2.12	0x4ae6d40c, 0x53ca, 0x414b, 0xa3, 0x05, 0x9f, 0x3b, 0xb4, 0x4c, 0xf4, 0x8a	**BS.StartImage – StartImage()** starts an EFI boot services driver.	1. Create an EFI boot services driver that installs **Protocol1**, opens **Protocol2**, and returns **EFI_DEVICE_ERROR**. The driver exits with **ExitData**. 2. Register a notification for **Protocol1**'s installation. 3. Load and Start the EFI boot services driver. **ExitDataSize** returned by **StartImage()** should be unchanged.
5.1.4.2.13	0x6b0d4a31, 0x929c, 0x4911, 0xac, 0xec, 0x4a, 0x0a, 0x9a, 0x94, 0x68, 0x33	**BS.StartImage – StartImage()** starts an EFI runtime services driver.	1. Create an EFI runtime services driver that installs **Protocol1**, opens **Protocol2**, and returns **EFI_SUCCESS**. 2. Register a notification for **Protocol1**'s installation. 3. Load and Start the EFI runtime services driver. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.4.2.14	0x27cef30a, 0xf4d9, 0x434f, 0xbd, 0xf4, 0x81, 0xbf, 0x56, 0xa8, 0x1e, 0xf4	`BS.StartImage – StartImage()` starts an EFI runtime services driver.	1. Create an EFI runtime services driver that installs `Protocol1`, opens `Protocol2`, and returns `EFI_SUCCESS`. 2. Register a notification for `Protocol1`'s installation. 3. Load and Start the EFI runtime services driver. The notify function should be invoked.
5.1.4.2.15	0x989d7749, 0xba06, 0x4d68, 0x93, 0x83, 0xe3, 0xf1, 0x7b, 0x15, 0xc7, 0x47	`BS.StartImage – StartImage()` starts an EFI runtime services driver.	1. Create an EFI runtime services driver that installs `Protocol1`, opens `Protocol2`, and returns `EFI_SUCCESS`. 2. Register a notification for `Protocol1`'s installation. 3. Load and Start the EFI runtime services driver. `Protocol1` should be located.
5.1.4.2.16	0x60a9841b, 0x6b46, 0x4663, 0x92, 0xb2, 0xef, 0xa4, 0x0a, 0xaa, 0x77, 0xd2	`BS.StartImage – StartImage()` starts an EFI runtime services driver.	1. Create an EFI runtime services driver that install `Protocol1`, and open `Protocol2`. 2. Register a notification for `Protocol1`'s installation. 3. Load and Start the EFI runtime services driver. `Protocol2` should be opened.
5.1.4.2.17	0xd43b34e0, 0x2faf, 0x469a, 0xaf, 0xfc, 0xf0, 0x16, 0x0f, 0x98, 0xd6, 0xf5	`BS.StartImage – StartImage()` starts an EFI runtime services driver.	1. Create an EFI runtime services driver that installs `Protocol1`, opens `Protocol2`, and returns `EFI_NOT_FOUND`. 2. Register a notification for `Protocol1`'s installation. 3. Load and Start the EFI runtime services driver. The return code should be `EFI_NOT_FOUND`.

Number	GUID	Assertion	Test Description
5.1.4.2.18	0xb2521b21, 0x00b8, 0x47a1, 0xba, 0x65, 0x9f, 0x73, 0x73, 0xe4, 0xaf, 0xde	`BS.StartImage – StartImage()` starts an EFI runtime services driver.	1. Create an EFI runtime services driver that installs `Protocol1`, opens `Protocol2`, and returns `EFI_NOT_FOUND`. 2. Register a notification for `Protocol1`'s installation. 3. Load and Start the EFI runtime services driver. `Protocol2` should be released the open reference.
5.1.4.2.19	0x696f4976, 0x33d4, 0x4e9a, 0xb6, 0xe7, 0xd8, 0x34, 0x62, 0x90, 0xf3, 0x4f	`BS.StartImage – StartImage()` starts an EFI runtime services driver.	1. Create an EFI runtime services driver that installs `Protocol1`, opens `Protocol2`, and returns `EFI_NOT_FOUND`. The driver exits with `ExitData`. 2. Register a notification for `Protocol1`'s installation. 3. Load and Start the EFI runtime services driver. `ExitData` returned by `StartImage()` should be not `NULL`.
5.1.4.2.20	0xa1b8f0d0, 0xcb12, 0x406c, 0x8c, 0x2f, 0x08, 0x27, 0x5f, 0x71, 0x91, 0x70	`BS.StartImage – StartImage()` returns `EFI_INVALID_PARAMETER` with same image handle twice.	1. Call `StartImage()` to start an image handle. 2. Call `StartImage()` with the same image handle again. The return code should be `EFI_INVALID_PARAMETER`.

3.4.3 UnloadImage()

Number	GUID	Assertion	Test Description
5.1.4.3.1	0xe315da57, 0x5da8, 0x41dd, 0x9f, 0x0d, 0x8f, 0xf1, 0x3b, 0xa1, 0x6e, 0x1c	`BS.UnloadImage – UnloadImage()` returns `EFI_INVALID_PARAMETER` with invalid `ImageHandle`.	1. Call `UnloadImage()` with `NULL` or invalid image handle. The return code must be `EFI_INVALID_PARAMETER`.
5.1.4.3.2	0x839b440a, 0xd3bb, 0x40e8, 0x8a, 0x98, 0x3c, 0x8b, 0xbb, 0xe7, 0x7b, 0xbc	`BS.UnloadImage – UnloadImage()` unloads unstarted EFI application at `EFI_TPL_APPLICATION`.	1. Load an EFI application. 2. Call `UnloadImage()` to unload the EFI application. The return code should be `EFI_SUCCESS`.
5.1.4.3.3	0xb4b209c2, 0xddbf, 0x4b2a, 0xa3, 0xda, 0x60, 0xc5, 0x5a, 0xd9, 0x19, 0xd3	`BS.UnloadImage – UnloadImage()` unloads unstarted EFI application at `EFI_TPL_CALLBACK`.	1. Load an EFI application. 2. Call `UnloadImage()` to unload the EFI application. The return code should be `EFI_SUCCESS`.
5.1.4.3.4	0x7b343dd7, 0xc5e9, 0x42c3, 0x91, 0x29, 0x7f, 0xab, 0x0d, 0x11, 0x02, 0x3d	`BS.UnloadImage – UnloadImage()` unloads unstarted EFI boot services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI boot services driver. 2. Call `UnloadImage()` to unload the EFI boot services driver. The return code should be `EFI_SUCCESS`.
5.1.4.3.5	0xf1a04ed0, 0x40f9, 0x4b6f, 0xb8, 0x89, 0x3b, 0x49, 0x52, 0x08, 0x83, 0xe1	`BS.UnloadImage – UnloadImage()` unloads unstarted EFI boot services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI boot services driver. 2. Call `UnloadImage()` to unload the EFI boot services driver. The return code should be `EFI_SUCCESS`.
5.1.4.3.6	0x3134d2cc, 0x5ad8, 0x407e, 0x86, 0x99, 0xfd, 0x14, 0x22, 0x2e, 0x8a, 0x40	`BS.UnloadImage – UnloadImage()` unloads unstarted EFI runtime services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI runtime services driver. 2. Call `UnloadImage()` to unload the EFI runtime services driver. The return code should be `EFI_SUCCESS`.
5.1.4.3.7	0x6843ffe5, 0x6ebe, 0x4164, 0xbb, 0xaf, 0x7e, 0x82, 0xa1, 0x11, 0xcf, 0x6d	`BS.UnloadImage – UnloadImage()` unloads unstarted EFI runtime services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI runtime services driver. 2. Call `UnloadImage()` to unload the EFI runtime services driver. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.4.3.8	0xa78edb49, 0xe488, 0x415d, 0x83, 0x1d, 0xda, 0x9c, 0x25, 0x06, 0xec, 0x89	`BS.UnloadImage – UnloadImage()` unloads started EFI application at `EFI_TPL_APPLICATION`.	1. Load an EFI application. 2. Start the EFI application. 3. Call `UnloadImage()` to unload the EFI application. The return code should be `EFI_INVALID_PARAMETER`.
5.1.4.3.9	0x77bfbb63, 0x10c4, 0x4cdf, 0x95, 0x26, 0x1a, 0x69, 0x3b, 0xb8, 0x60, 0x39	`BS.UnloadImage – UnloadImage()` unloads started EFI application at `EFI_TPL_CALLBACK`.	1. Load an EFI application. 2. Start the EFI application. 3. Call `UnloadImage()` to unload the EFI application. The return code should be `EFI_INVALID_PARAMETER`.
5.1.4.3.10	0xf50493b0, 0x9653, 0x409b, 0x83, 0xa9, 0xc0, 0x13, 0x3a, 0x34, 0xa4, 0x20	`BS.UnloadImage – UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI boot services driver. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. The return code should be `EFI_SUCCESS`.
5.1.4.3.11	0x5a612e62, 0x9982, 0x4f87, 0xa3, 0xa1, 0x16, 0xaf, 0x5f, 0x8d, 0xbd, 0x87	`BS.UnloadImage – UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI boot services driver. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. The return code should be `EFI_SUCCESS`.
5.1.4.3.12	0xec5c4ee0, 0x9a37, 0x488e, 0x8e, 0xee, 0xb0, 0x61, 0xa7, 0x3c, 0xc5, 0x03	`BS.UnloadImage – UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI boot services driver that uninstalls `Protocol1` in `Unload()` function. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. `Protocol1` should not be located.
5.1.4.3.13	0x51ab01a4, 0x6a66, 0x468f, 0xae, 0xe4, 0x4d, 0x5e, 0xb5, 0x88, 0x00, 0x76	`BS.UnloadImage – UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI boot services driver that uninstalls `Protocol1` in `Unload()` function. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. `Protocol1` should not be located.

Number	GUID	Assertion	Test Description
5.1.4.3.14	0xe7dd55e2, 0x2461, 0x40e6, 0x8d, 0x97, 0x6d, 0x9e, 0x2a, 0xf1, 0xe1, 0x67	`BS.UnloadImage – UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI boot services driver that does not close `Protocol2` in `Unload()` function. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. `Protocol2` could still be located.
5.1.4.3.15	0x8c83ad3d, 0xb796, 0x45b6, 0xa8, 0x0c, 0xe4, 0x89, 0xed, 0xa5, 0x34, 0x7f	`BS.UnloadImage – UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI boot services driver which does not close `Protocol2` in `Unload()` function. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. `Protocol1` could still be located.
5.1.4.3.16	0x86de7316, 0xc7a1, 0x4553, 0xa0, 0xf6, 0x52, 0x41, 0x98, 0x51, 0xfb, 0x3f	`BS.UnloadImage – UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI boot services driver which returns `EFI_DEVICE_ERROR` in `Unload()` function. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. The return code should be `EFI_DEVICE_ERROR`.
5.1.4.3.17	0xf9d2a7c4, 0x5f7f, 0x4e7e, 0x98, 0x27, 0x39, 0xf5, 0x78, 0x07, 0x6b, 0x83	`BS.UnloadImage – UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI boot services driver which returns `EFI_DEVICE_ERROR` in `Unload()` function. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. The return code should be `EFI_DEVICE_ERROR`.

Number	GUID	Assertion	Test Description
5.1.4.3.18	0x7069cedb, 0xc81c, 0x4d24, 0xac, 0xa4, 0x0f, 0xd2, 0x0d, 0x81, 0x5d, 0x13	`BS.UnloadImage –` `UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI boot services driver which uninstalls `Protocol1` and returns `EFI_DEVICE_ERROR` in `Unload()` function. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. `Protocol1` should not be located.
5.1.4.3.19	0x6b493911, 0x11b7, 0x4468, 0xb2, 0x56, 0xe5, 0xb8, 0xcb, 0xdf, 0xbf, 0x4d	`BS.UnloadImage –` `UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI boot services driver which uninstalls `Protocol1` and returns `EFI_DEVICE_ERROR` in `Unload()` function. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. `Protocol1` should not be located.
5.1.4.3.20	0x1bb5bf2c, 0x98e2, 0x4bef, 0xbe, 0x43, 0x9b, 0xb8, 0x92, 0x99, 0xd5, 0xf0	`BS.UnloadImage –` `UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI boot services driver which does not close `Protocol2` and returns `EFI_DEVICE_ERROR` in `Unload()` function. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. `Protocol2` should still be opened.
5.1.4.3.21	0xb55e7fa8, 0x39b0, 0x4eab, 0x84, 0xdd, 0xcd, 0x5f, 0xac, 0x63, 0x65, 0xa9	`BS.UnloadImage –` `UnloadImage()` unloads started EFI boot services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI boot services driver which does not close `Protocol2` and returns `EFI_DEVICE_ERROR` in `Unload()` function. 2. Start the EFI boot services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. `Protocol2` should still be opened.

Number	GUID	Assertion	Test Description
5.1.4.3.22	0xbe80ffe7, 0xcd56, 0x4e7a, 0xae, 0xb1, 0xd5, 0x05, 0x2d, 0xe7, 0x3a, 0x66	**BS.UnloadImage –** **UnloadImage()** unloads started EFI boot services driver at **EFI_TPL_APPLICATION**.	1. Load an EFI boot services driver which sets up the **Unload()** function in **DriverBinding.Start()** function. 2. Call **UnloadImage()** to unload the EFI boot services driver. The return code should be **EFI_UNSUPPORTED**.
5.1.4.3.23	0x25611b63, 0x6439, 0x4bcb, 0xb4, 0xd8, 0xb5, 0x0a, 0x34, 0xf9, 0x0e, 0x45	**BS.UnloadImage –** **UnloadImage()** unloads started EFI boot services driver at **EFI_TPL_CALLBACK**.	1. Load an EFI boot services driver which sets up the **Unload()** function in **DriverBinding.Start()** function. 2. Call **UnloadImage()** to unload the EFI boot services driver. The return code should be **EFI_UNSUPPORTED**.
5.1.4.3.24	0x5a21983a, 0xc872, 0x4e12, 0x97, 0x36, 0xe5, 0x33, 0xe7, 0x8d, 0xad, 0xfe	**BS.UnloadImage –** **UnloadImage()** unloads started EFI boot services driver at **EFI_TPL_APPLICATION**.	1. Load an EFI boot services driver which installs **Protocol1** in the entry point, and sets up the **Unload()** function in **DriverBinding.Start()** function. 2. Call **UnloadImage()** to unload the EFI boot services driver. **Protocol1** should still be located.
5.1.4.3.25	0xe29713dc, 0xcb25, 0x4abc, 0xb7, 0xec, 0x3c, 0xbb, 0xfc, 0xe6, 0xf3, 0xcf	**BS.UnloadImage –** **UnloadImage()** unloads started EFI boot services driver at **EFI_TPL_CALLBACK**.	1. Load an EFI boot services driver which installs **Protocol1** in the entry point, and sets up the **Unload()** function in **DriverBinding.Start()** function. 2. Call **UnloadImage()** to unload the EFI boot services driver. **Protocol1** should still be located.

Number	GUID	Assertion	Test Description
5.1.4.3.26	0x7a648f75, 0x6bb8, 0x4b57, 0xa5, 0xe3, 0x82, 0x1a, 0xe9, 0xa3, 0x2a, 0xd8	**BS.UnloadImage – UnloadImage()** unloads started EFI boot services driver at **EFI_TPL_APPLICATION**.	1. Load an EFI boot services driver which opens **Protocol2** in the entry point, and sets up the **Unload()** function in **DriverBinding.Start()** function. 2. Call **UnloadImage()** to unload the EFI boot services driver. **Protocol2** should still be opened.
5.1.4.3.27	0xa05b3b2b, 0x0d6c, 0x469c, 0xa3, 0x25, 0x97, 0x4f, 0xa4, 0xc2, 0x59, 0x2d	**BS.UnloadImage – UnloadImage()** unloads started EFI boot services driver at **EFI_TPL_CALLBACK**.	1. Load an EFI boot services driver which opens **Protocol2** in the entry point, and sets up the **Unload()** function in **DriverBinding.Start()** function. 2. Call **UnloadImage()** to unload the EFI boot services driver. **Protocol2** should still be opened.
5.1.4.3.28	0x81866024, 0x8bfb, 0x4489, 0x83, 0x58, 0xc8, 0xcc, 0x4c, 0x4a, 0xd1, 0x79	**BS.UnloadImage – UnloadImage()** unloads started EFI runtime services driver at **EFI_TPL_APPLICATION**.	1. Load an EFI runtime services driver. 2. Start the EFI runtime services driver. 3. Call **UnloadImage()** to unload the EFI runtime services driver. The return code should be **EFI_SUCCESS**.
5.1.4.3.29	0x4fe0c243, 0x1691, 0x4c99, 0x90, 0xf9, 0xaa, 0xb0, 0x19, 0xd2, 0xb5, 0xa9	**BS.UnloadImage – UnloadImage()** unloads started EFI runtime services driver at **EFI_TPL_CALLBACK**.	1. Load an EFI runtime services driver. 2. Start the EFI runtime services driver. 3. Call **UnloadImage()** to unload the EFI runtime services driver. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.4.3.30	0x07331a90, 0xfb7b, 0x45f9, 0x82, 0x9d, 0x4e, 0x95, 0x0a, 0x3b, 0x5b, 0x0c	BS.UnloadImage – UnloadImage() unloads started EFI runtime services driver at EFI_TPL_APPLICATION.	1. Load an EFI runtime services driver which uninstalls Protocol1 in Unload() function. 2. Start the EFI runtime services driver. 3. Call UnloadImage() to unload the EFI runtime services driver. Protocol1 should not be located.
5.1.4.3.31	0x6ff0ddac, 0xd358, 0x4e0d, 0xb7, 0x07, 0x84, 0xc6, 0xa9, 0xf6, 0x13, 0x2f	BS.UnloadImage – UnloadImage() unloads started EFI runtime services driver at EFI_TPL_CALLBACK.	1. Load an EFI runtime services driver which uninstalls Protocol1 in Unload() function. 2. Start the EFI runtime services driver. 3. Call UnloadImage() to unload the EFI runtime services driver. Protocol1 should not be located.
5.1.4.3.32	0x7ea89cd8, 0x1dfb, 0x4949, 0xac, 0xe0, 0x0a, 0x2c, 0x19, 0x8c, 0x51, 0x3d	BS.UnloadImage – UnloadImage() unloads started EFI runtime services driver at EFI_TPL_APPLICATION.	1. Load an EFI runtime services driver which does not close Protocol2 in Unload() function. 2. Start the EFI runtime services driver. 3. Call UnloadImage() to unload the EFI runtime services driver. Protocol2 should still be opened.
5.1.4.3.33	0x40a4f27e, 0x4854, 0x4e52, 0x8a, 0x4f, 0x72, 0xb3, 0xb4, 0x0e, 0xaf, 0xdb	BS.UnloadImage – UnloadImage() unloads started EFI runtime services driver at EFI_TPL_CALLBACK.	1. Load an EFI runtime services driver which does not close Protocol2 in Unload() function. 2. Start the EFI runtime services driver. 3. Call UnloadImage() to unload the EFI runtime services driver. Protocol2 should still be opened.

Number	GUID	Assertion	Test Description
5.1.4.3.34	0xea461fd1, 0xa5de, 0x4f17, 0xbc, 0xa3, 0x6c, 0x5c, 0xa9, 0xaf, 0x2f, 0xf7	`BS.UnloadImage – UnloadImage()` unloads started EFI runtime services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI runtime services driver which returns `EFI_DEVICE_ERROR` in `Unload()` function. 2. Start the EFI runtime services driver. 3. Call `UnloadImage()` to unload the EFI boot services driver. The return code should be `EFI_DEVICE_ERROR`.
5.1.4.3.35	0x221ab8d1, 0xd19c, 0x4877, 0xaa, 0x13, 0x36, 0xb9, 0x93, 0xfd, 0x8b, 0x3c	`BS.UnloadImage – UnloadImage()` unloads started EFI runtime services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI runtime services driver which returns `EFI_DEVICE_ERROR` in `Unload()` function. 2. Start the EFI runtime services driver. 3. Call `UnloadImage()` to unload the EFI runtime services driver. The return code should be `EFI_DEVICE_ERROR`.
5.1.4.3.36	0x657d6565, 0xf26b, 0x468a, 0xb7, 0x37, 0x68, 0xd1, 0x09, 0xd9, 0xfa, 0xc3	`BS.UnloadImage – UnloadImage()` unloads started EFI runtime services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI runtime services driver which uninstalls `Protocol1` and returns `EFI_DEVICE_ERROR` in `Unload()` function. 2. Start the EFI runtime services driver. 3. Call `UnloadImage()` to unload the EFI runtime services driver. `Protocol1` should not be located.
5.1.4.3.37	0xb792ec09, 0x49c5, 0x42f6, 0xba, 0xe3, 0x71, 0x76, 0xe6, 0x4c, 0xe8, 0xad	`BS.UnloadImage – UnloadImage()` unloads started EFI runtime services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI runtime services driverthat uninstalls `Protocol1` and returns `EFI_DEVICE_ERROR` in `Unload()` function. 2. Start the EFI runtime services driver. 3. Call `UnloadImage()` to unload the EFI runtime services driver. `Protocol1` should not be located.

Number	GUID	Assertion	Test Description
5.1.4.3.38	0xca0fd0c5, 0x37a4, 0x4483, 0xbb, 0xb3, 0xca, 0x5a, 0x50, 0x4d, 0xbc, 0x1d	BS.UnloadImage – UnloadImage() unloads started EFI runtime services driver at EFI_TPL_APPLICATION.	1. Load an EFI runtime services driver which does not close Protocol2 and returns EFI_DEVICE_ERROR in Unload() function. 2. Start the EFI runtime services driver. 3. Call UnloadImage() to unload the EFI runtime services driver. Protocol2 should still be opened.
5.1.4.3.39	0x121c720e, 0x8d87, 0x49bd, 0xac, 0x98, 0x87, 0x39, 0x51, 0xea, 0xd4, 0x5e	BS.UnloadImage – UnloadImage() unloads started EFI runtime services driver at EFI_TPL_CALLBACK.	1. Load an EFI runtime services driver which does not close Protocol2 and returns EFI_DEVICE_ERROR in Unload() function. 2. Start the EFI runtime services driver. 3. Call UnloadImage() to unload the EFI runtime services driver. Protocol2 should still be opened.
5.1.4.3.40	0xbf69d01d, 0x2bcf, 0x4a9b, 0xb5, 0x51, 0xf7, 0xa4, 0x6d, 0x13, 0x6c, 0xba	BS.UnloadImage – UnloadImage() unloads started EFI runtime services driver at EFI_TPL_APPLICATION.	1. Load an EFI runtime services driver which sets up the Unload() function in DriverBinding.Start() function. 2. Call UnloadImage() to unload the EFI runtime services driver. The return code should be EFI_UNSUPPORTED.
5.1.4.3.41	0xf5f305cb, 0x4828, 0x476b, 0xa2, 0x18, 0x77, 0x9c, 0xe8, 0x04, 0x04, 0x4f	BS.UnloadImage – UnloadImage() unloads started EFI runtime services driver at EFI_TPL_CALLBACK.	1. Load an EFI runtime services driver which sets up the Unload() function in DriverBinding.Start() function. 2. Call UnloadImage() to unload the EFI runtime services driver. The return code should be EFI_UNSUPPORTED.

Number	GUID	Assertion	Test Description
5.1.4.3.42	0xe6c5f338, 0x8654, 0x452a, 0xb7, 0x69, 0xa9, 0xb3, 0x2f, 0x0a, 0x37, 0x6b	`BS.UnloadImage – UnloadImage()` unloads started EFI runtime services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI runtime services driver which installs `Protocol1` in the entry point, and sets up the `Unload()` function in `DriverBinding.Start()` function. 2. Call `UnloadImage()` to unload the EFI runtime services driver. `Protocol1` should still be located.
5.1.4.3.43	0xa390f3e7, 0x90d9, 0x439b, 0xa8, 0x39, 0x66, 0x5c, 0xc9, 0x12, 0x2d, 0x4f	`BS.UnloadImage – UnloadImage()` unloads started EFI runtime services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI runtime services driver which installs `Protocol1` in the entry point, and sets up the `Unload()` function in `DriverBinding.Start()` function. 2. Call `UnloadImage()` to unload the EFI runtime services driver. `Protocol1` should still be located.
5.1.4.3.44	0x026166c4, 0x14df, 0x4b40, 0x82, 0xd0, 0x4f, 0x0a, 0x9d, 0x4f, 0x97, 0xd3	`BS.UnloadImage – UnloadImage()` unloads started EFI runtime services driver at `EFI_TPL_APPLICATION`.	1. Load an EFI runtime services driver which opens `Protocol2` in the entry point, and sets up the `Unload()` function in `DriverBinding.Start()` function. 2. Call `UnloadImage()` to unload the EFI runtime services driver. `Protocol2` should still be opened.
5.1.4.3.45	0x8cbea92b, 0x2cbf, 0x4660, 0x97, 0x0f, 0x95, 0x0a, 0x3c, 0x46, 0xd1, 0x67	`BS.UnloadImage – UnloadImage()` unloads started EFI runtime services driver at `EFI_TPL_CALLBACK`.	1. Load an EFI runtime services driverthat opens `Protocol2` in the entry point, and sets up the `Unload()` function in `DriverBinding.Start()` function. 2. Call `UnloadImage()` to unload the EFI runtime services driver. `Protocol2` should still be opened.

3.5 EFI_IMAGE_ENTRY_POINT

This is the entry point of EFI image. No test case is designed to verify it.

3.5.1 Exit()

Number	GUID	Assertion	Test Description
5.1.4.5.1	0xe2a045da, 0xec4f, 0x4b61, 0xbb, 0x44, 0x18, 0xab, 0xce, 0x47, 0x80, 0xff	**BS.Exit – Exit()** returns **EFI_INVALID_PARAMETER** with invalid **ImageHandle**.	1. Call **Exit()** with **NULL** or invalid image handle. The return code should be **EFI_INVALID_PARAMETER**.
5.1.4.5.2	0x8300df83, 0xdfdc, 0x4933, 0xa1, 0xc1, 0x19, 0x32, 0x1f, 0x24, 0xd5, 0xf5	**BS.Exit – Exit()** exits an unstarted EFI application at **EFI_TPL_APPLICATION**.	1. Call **LoadImage()** to load an EFI application. 2. Call **Exit()** to unload the unstarted image. The return code should be **EFI_SUCCESS**.
5.1.4.5.3	0xfea31754, 0x871d, 0x45e2, 0xb5, 0xdc, 0xbc, 0xbb, 0x7f, 0x99, 0x1d, 0xa9	**BS.Exit – Exit()** exits an unstarted EFI application at **EFI_TPL_CALLBACK**.	1. Call **LoadImage()** to load an EFI application. 2. Call **Exit()** to unload the unstarted image. The return code should be **EFI_SUCCESS**.
5.1.4.5.4	0x8dd098c6, 0x9755, 0x4b7c, 0xbe, 0x51, 0xbc, 0xfa, 0x15, 0xfb, 0x34, 0x13	**BS.Exit – Exit()** exits an unstarted EFI boot services driver at **EFI_TPL_APPLICATION**.	1. Call **LoadImage()** to load an EFI boot services driver. 2. Call **Exit()** to unload the unstarted image. The return code should be **EFI_SUCCESS**.
5.1.4.5.5	0xa557943e, 0x7aa0, 0x42c0, 0x9a, 0x87, 0x2f, 0xde, 0x4e, 0x32, 0x1d, 0xa9	**BS.Exit – Exit()** exits an unstarted EFI boot services driver at **EFI_TPL_CALLBACK**.	1. Call **LoadImage()** to load an EFI boot services driver. 2. Call **Exit()** to unload the unstarted image. The return code should be **EFI_SUCCESS**.
5.1.4.5.6	0x7446e86b, 0xcb74, 0x47b1, 0xab, 0x9a, 0x58, 0x37, 0x6a, 0xa7, 0x7a, 0xbd	**BS.Exit – Exit()** exits an unstarted EFI runtime services driver at **EFI_TPL_APPLICATION**.	1. Call **LoadImage()** to load an EFI runtime services driver. 2. Call **Exit()** to unload the unstarted image. The return code should be **EFI_SUCCESS**.
5.1.4.5.7	0x0b8c9ac6, 0xc469, 0x465e, 0xa8,0xc6, 0x50, 0xfa, 0xab, 0xeb, 0x86, 0x2b	**BS.Exit – Exit()** exits an unstarted EFI runtime services driver at **EFI_TPL_CALLBACK**.	1. Call **LoadImage()** to load an EFI runtime services driver. 2. Call **Exit()** to unload the unstarted image. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.4.5.8	0xbcfbdc03, 0x1b40, 0x4637, 0xb2, 0x9f, 0xbb, 0x4b, 0x1c, 0x98, 0xf4, 0xc7	`BS.Exit – Exit()` returns `EFI_INVALID_PARAMETER` with started image at `EFI_TPL_CALLBACK`.	1. Call `LoadImage()` to load an EFI application, an EFI boot services driver, and an EFI runtime services driver. 2. Call `StartImage()` to start them. 3. Call `Exit()` to unload the started images. The return code should be `EFI_INVALID_PARAMETER`.
5.1.4.5.9	0x245f4a63, 0x30bb, 0x4feb, 0xa2, 0x80, 0x80, 0x66, 0xa7, 0x00, 0x9d, 0xb8	`BS.Exit – Exit()` returns `EFI_INVALID_PARAMETER` with started image at `EFI_TPL_APPLICATION`.	1. Call `LoadImage()` to load an EFI application, an EFI boot services driver, and an EFI runtime services driver. 2. Call `StartImage()` to start them. 3. Call `Exit()` to unload the started images. The return code should be `EFI_INVALID_PARAMETER`.
5.1.4.5.10	0x9ee96cf8, 0xaefd, 0x4eb4, 0xab, 0x62, 0x0b, 0x57, 0x3d, 0x9f, 0x7f, 0x67	`BS.Exit – Exit()` exits an EFI application in its entry point.	1. Call `LoadImage()` to load an EFI application in which `Exit()` is invoked with a successful exit code in its entry point. 2. Call `StartImage()` to start it. The return code should be `EFI_SUCCESS`.
5.1.4.5.11	0xb8a2b65d, 0xfe9c, 0x4eee, 0xab, 0x58, 0xd6, 0xf5, 0x4d, 0x38, 0x74, 0x29	`BS.Exit – Exit()` exits an EFI application in its entry point.	1. Call `LoadImage()` to load an EFI application in which `Protocol3` is installed and uninstalled, and `Exit()` is invoked with a successful exit code in its entry point. 2. Register a notify function to `Protocol3`'s installation. 3. Call `StartImage()` to start it. The notify function should be invoked.
5.1.4.5.12	0x6ad85f56, 0xcf1d, 0x468c, 0xa9, 0x35, 0x10, 0xc4, 0x72, 0x72, 0xbf, 0x19	`BS.Exit – Exit()` exits an EFI application in its entry point.	1. Call `LoadImage()` to load an EFI application in which `Protocol4` is opened, and `Exit()` is invoked with a successful exit code in its entry point. 2. Call `StartImage()` to start it. `Protocol4` should not be opened.

Number	GUID	Assertion	Test Description
5.1.4.5.13	0x73d43440, 0x619a, 0x45d7, 0x9d, 0x37, 0xaa, 0xb7, 0xca, 0x34, 0x4f, 0x4d	BS.Exit – Exit() exits an EFI application in its entry point.	1. Call LoadImage() to load an EFI application in which Exit() is invoked with a successful exit code, and after Exit a variable is set in its entry point. 2. Call StartImage() to start it. The variable should not be set.
5.1.4.5.14	0xbd9dae62, 0xab61, 0x40b0, 0x8f, 0xbc, 0xdd, 0xc8, 0x39, 0xcc, 0x18, 0x62	BS.Exit – Exit() exits an EFI application in its entry point with error code.	1. Call LoadImage() to load an EFI application in which Exit() is invoked with exit code EFI_DEVICE_ERROR in its entry point. 2. Call StartImage() to start it. The return code should be EFI_DEVICE_ERROR.
5.1.4.5.15	0x6059ace5, 0xb01c, 0x4886, 0xb9, 0xf3, 0xd0, 0x72, 0x61, 0x2c, 0xfc, 0x44	BS.Exit – Exit() exits an EFI application in its entry point with error code.	1. Call LoadImage() to load an EFI application in which Protocol3 is installed and uninstalled, and Exit() is invoked with exit code EFI_DEVICE_ERROR in its entry point. 2. Register a notify function to Protocol3's installation. 3. Call StartImage() to start it. The notify function should be invoked.
5.1.4.5.16	0xfae6a2d2, 0x0b34, 0x48af, 0x97, 0x0c, 0xe6, 0x84, 0xa5, 0x05, 0x9b, 0x0d	BS.Exit – Exit() exits an EFI application in its entry point with error code.	1. Call LoadImage() to load an EFI application in which Protocol4 is opened, and Exit() is invoked with exit code EFI_DEVICE_ERROR in its entry point. 2. Call StartImage() to start it. Protocol4 should not be opened.
5.1.4.5.17	0x7ef5b4f4, 0xd07a, 0x4610, 0x91, 0xc9, 0x4f, 0x2b, 0x6a, 0x2e, 0xd0, 0x68	BS.Exit – Exit() exits an EFI application in its entry point with error code.	1. Call LoadImage() to load an EFI application in which Exit() is invoked with exit code EFI_DEVICE_ERROR in its entry point. 2. Call StartImage() to start it. The return ExitData should be the same as in EFI application.

Number	GUID	Assertion	Test Description
5.1.4.5.18	0x4e3985c7, 0x65ac, 0x4cd2, 0x89, 0xba, 0x57, 0x81, 0xad, 0xd5, 0xd1, 0x47	BS.Exit – Exit() exits an EFI application in its entry point with error code.	1. Call LoadImage() to load an EFI application in which Exit() is invoked with a successful exit code, and after Exit a variable is set in its entry point. 2. Call StartImage() to start it. The variable should not be set.
5.1.4.5.19	0xb35676e3, 0xcd57, 0x4df0, 0xba, 0x3a, 0xd3, 0x24, 0x77, 0x44, 0xca, 0x4f	BS.Exit – Exit() exits an EFI boot services driver in its entry point.	1. Call LoadImage() to load an EFI boot services driver in which Exit() is invoked with a successful exit code in its entry point. 2. Call StartImage() to start it. The return code should be EFI_SUCCESS.
5.1.4.5.20	0x66e31a54, 0xb900, 0x410f, 0xbe, 0xa2, 0x25, 0x8e, 0x6b, 0x98, 0x3e, 0xf8	BS.Exit – Exit() exits an EFI boot services driver in its entry point.	1. Call LoadImage() to load an EFI boot services driver in which Protocol3 is installed, and Exit() is invoked with a successful exit code in its entry point. 2. Register a notify function to Protocol3's installation. 3. Call StartImage() to start it. The notify function should be invoked.
5.1.4.5.21	0x8a01c7fb, 0xee3c, 0x4e7f, 0x8b, 0xc9, 0xfb, 0xe0, 0x3d, 0x69, 0xaf, 0x3f	BS.Exit – Exit() exits an EFI boot services driver in its entry point.	1. Call LoadImage() to load an EFI boot services driver in which Protocol3 is installed, and Exit() is invoked with a successful exit code in its entry point. 2. Register a notify function to Protocol3's installation. 3. Call StartImage() to start it. Protocol3 should be located.
5.1.4.5.22	0xec2e0e5a, 0xac2e, 0x4f31, 0x9f, 0x39, 0xc7, 0x0a, 0xb1, 0x76, 0x0e, 0x82	BS.Exit – Exit() exits an EFI boot services driver in its entry point.	1. Call LoadImage() to load an EFI boot services driver in which Protocol4 is opened, and Exit() is invoked with success exit code in its entry point. 2. Call StartImage() to start it. Protocol4 should be opened.

Number	GUID	Assertion	Test Description
5.1.4.5.23	0xea28a835, 0xcfaa, 0x4d4a, 0x8f, 0xf3, 0x13, 0xea, 0x84, 0x7e, 0x8f, 0xf2	`BS.Exit – Exit()` exits an EFI boot services driver in its entry point.	1. Call `LoadImage()` to load an EFI boot services driver in which `Exit()` is invoked with a successful exit code, and after Exit an variable is set in its entry point. 2. Call `StartImage()` to start it. The variable should not be set.
5.1.4.5.24	0x17a5a71f, 0xc831, 0x469a, 0xbf, 0x84, 0x72, 0xc6, 0xc3, 0xd5, 0xd5, 0xac	`BS.Exit – Exit()` exits an EFI boot services driver in its entry point with error code.	1. Call `LoadImage()` to load an EFI boot services driver in which `Exit()` is invoked with exit code `EFI_DEVICE_ERROR` in its entry point. 2. Call `StartImage()` to start it. The return code should be `EFI_DEVICE_ERROR`.
5.1.4.5.25	0xd9143e4b, 0xab3d, 0x4a80, 0xa6, 0xee, 0xe3, 0xd8, 0x92, 0x50, 0x8b, 0x47	`BS.Exit – Exit()` exits an EFI boot services driver in its entry point with error code.	1. Call `LoadImage()` to load an EFI boot services driver in which `Protocol3` is installed, and `Exit()` is invoked with exit code `EFI_DEVICE_ERROR` in its entry point. 2. Register a notify function to `Protocol3`'s installation. 3. Call `StartImage()` to start it. The notify function should be invoked.
5.1.4.5.26	0xce9000ba, 0xb4a8, 0x4f89, 0xaf, 0x2a, 0x99, 0x4a, 0x8c, 0xf8, 0x7b, 0xcd	`BS.Exit – Exit()` exits an EFI boot services driver in its entry point with error code.	1. Call `LoadImage()` to load an EFI boot services driver in which `Protocol4` is opened, and `Exit()` is invoked with exit code `EFI_DEVICE_ERROR` in its entry point. 2. Call `StartImage()` to start it. `Protocol4` should be opened.

Number	GUID	Assertion	Test Description
5.1.4.5.27	0xb9868240, 0x9b8d, 0x4e5d, 0x8b, 0x22, 0x21, 0xce, 0x0a, 0xee, 0x0a, 0x91	`BS.Exit – Exit()` exits an EFI boot services driver in its entry point with error code.	1. Call `LoadImage()` to load an EFI boot services driver in which `Exit()` is invoked with exit code `EFI_DEVICE_ERROR` in its entry point. 2. Call `StartImage()` to start it. The return `ExitData` should be the same as in EFI application.
5.1.4.5.28	0x5a639776, 0x7d9c, 0x4775, 0xaa, 0x37, 0x2d, 0xb9, 0x55, 0x28, 0x64, 0xea	`BS.Exit – Exit()` exits an EFI runtime services driver in its entry point.	1. Call `LoadImage()` to load an EFI runtime services driver in which `Exit()` is invoked with a successful exit code in its entry point. 2. Call `StartImage()` to start it. The return code should be `EFI_SUCCESS`.
5.1.4.5.29	0x85aedeeb, 0x351b, 0x4359, 0x8d, 0xb6, 0xbc, 0x4d, 0x58, 0x87, 0x64, 0x31	`BS.Exit – Exit()` exits an EFI runtime services driver in its entry point.	1. Call `LoadImage()` to load an EFI runtime services driver in which `Protocol3` is installed, and `Exit()` is invoked with a successful exit code in its entry point. 2. Register a notify function to `Protocol3`'s installation. 3. Call `StartImage()` to start it. The notify function should be invoked.
5.1.4.5.30	0x89f38a82, 0x295a, 0x4388, 0x8a, 0x25, 0x3e, 0x23, 0xe1, 0xeb, 0x96, 0xef	`BS.Exit – Exit()` exits an EFI runtime services driver in its entry point.	1. Call `LoadImage()` to load an EFI runtime services driver in which `Protocol3` is installed, and `Exit()` is invoked with a successful exit code in its entry point. 2. Register a notify function to `Protocol3`'s installation. 3. Call `StartImage()` to start it. `Protocol3` should be located.

Number	GUID	Assertion	Test Description
5.1.4.5.31	0x957ab7aa, 0x0eef, 0x48cc, 0xb2, 0x25, 0xa0, 0x11, 0xd8, 0x81, 0xe6, 0x81	`BS.Exit – Exit()` exits an EFI runtime services driver in its entry point.	1. Call `LoadImage()` to load an EFI runtime services driver in which `Protocol4` is opened, and `Exit()` is invoked with a successful exit code in its entry point. 2. Call `StartImage()` to start it. `Protocol4` should be opened.
5.1.4.5.32	0x04fb22ab, 0x6cf6, 0x411f, 0x85, 0x90, 0x28, 0x9c, 0x02, 0x03, 0xcc, 0x36	`BS.Exit – Exit()` exits an EFI runtime services driver in its entry point.	1. Call `LoadImage()` to load an EFI runtime services driver in which `Exit()` is invoked with a successful exit code, and after Exit an variable is set in its entry point. 2. Call `StartImage()` to start it. The variable should not be set.
5.1.4.5.33	0x683163f8, 0x1e56, 0x49e3, 0xa7, 0x9e, 0x9f, 0xea, 0x90, 0x46, 0x4a, 0x18	`BS.Exit – Exit()` exits an EFI runtime services driver in its entry point with error code.	1. Call `LoadImage()` to load an EFI runtime services driver in which `Exit()` is invoked with exit code `EFI_DEVICE_ERROR` in its entry point. 2. Call `StartImage()` to start it. The return code should be `EFI_DEVICE_ERROR`.
5.1.4.5.34	0x047da922, 0xfdcc, 0x4be2, 0xbb, 0x14, 0x29, 0x79, 0x18, 0xf8, 0x03, 0x1c	`BS.Exit – Exit()` exits an EFI runtime services driver in its entry point with error code.	1. Call `LoadImage()` to load an EFI runtime services driver in which `Protocol3` is installed, and `Exit()` is invoked with exit code `EFI_DEVICE_ERROR` in its entry point. 2. Register a notify function to `Protocol3`'s installation. 3. Call `StartImage()` to start it. The notify function should be invoked.

Number	GUID	Assertion	Test Description
5.1.4.5.35	0x1a133e13, 0xcb01, 0x4297, 0xaf, 0x19, 0x03, 0xd7, 0x46, 0x06, 0x8b, 0xaa	**BS.Exit – Exit()** exits an EFI runtime services driver in its entry point with error code.	1. Call **LoadImage()** to load an EFI runtime services driver in which **Protocol4** is opened, and **Exit()** is invoked with exit code **EFI_DEVICE_ERROR** in its entry point. 2. Call **StartImage()** to start it. **Protocol4** should be opened.
5.1.4.5.36	0x85c85f4d, 0x519b, 0x4b98, 0xbc, 0x7a, 0x94, 0x47, 0xcc, 0x27, 0xf6, 0x1e	**BS.Exit – Exit()** exits an EFI runtime services driver in its entry point with error code.	1. Call **LoadImage()** to load an EFI runtime services driver in which **Exit()** is invoked with exit code **EFI_DEVICE_ERROR** in its entry point. 2. Call **StartImage()** to start it. The return **ExitData** should be same as in EFI application.

3.5.2 ExitBootServices()

Number	GUID	Assertion	Test Description
5.1.4.6.1	0xa5bb81fa, 0x1063, 0x4358, 0x97, 0xaf, 0xad, 0x57, 0xd4, 0x2b, 0xf0, 0x55	**BS.ExitBootServices – ExitBootServices()** returns **EFI_INVALID_PARAMETER** with invalid **MapKey**	1. Call **ExitBootServices()** with invalid **MapKey**, The return code should be **EFI_INVALID_PARAMETER**.

3.6 Misc Boot Services Test

Reference Document:

UEFI Specification, Miscellaneous Boot Services Section.

Table 5. Miscellaneous Boot Services Functions

Name	Type	Description
SetWatchdogTimer()	Boot	Resets and sets a watchdog timer used during boot services time.
Stall()	Boot	Stalls the processor.
CopyMem()	Boot	Copies the contents of one buffer to another buffer.
SetMem()	Boot	Fills a buffer with a specified value.
GetNextMonotonicCount()	Boot	Returns a monotonically increasing count for the platform.

Name	Type	Description
InstallConfigurationTable()	Boot	Adds, updates, or removes a configuration table from the EFI System Table.
CalculateCrc32()	Boot	Computes and returns a 32-bit CRC for a data buffer.

3.6.1 SetWatchdogTimer()

Number	GUID	Assertion	Test Description
5.1.5.1.1	0x9f677836, 0x5175, 0x4fdf, 0x85, 0x2e, 0xe8, 0xfd, 0x46, 0x53, 0xb2, 0x1c	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer at **EFI_TPL_APPLICATION**	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. The return code should be **EFI_SUCCESS**.
5.1.5.1.2	0xea8d88ac, 0x05b1, 0x4d69, 0xbb, 0xc1, 0xa0, 0x72, 0x04, 0x2f, 0xb8, 0x98	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer at **EFI_TPL_CALLBACK**	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. The return code should be **EFI_SUCCESS**.
5.1.5.1.3	0xa6d41372, 0x4cce, 0x4e11, 0x8d, 0x84, 0xc3, 0x35, 0x46, 0x0a, 0xe1, 0xaf	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer at **EFI_TPL_NOTIFY**	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. The return code should be **EFI_SUCCESS**.
5.1.5.1.4	0x4cd2a140, 0x94e1, 0x448c, 0x99, 0xe7, 0xd4, 0xf5, 0x3b, 0xd8, 0x45, 0x44	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer at **EFI_TPL_APPLICATION**	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. 2. Call **Stall()** with 3.5 seconds. 3. Call **SetWatchdogTimer()** to disable the watchdog timer. The system should not be reset.
5.1.5.1.5	0x3d3bee76, 0x3be8, 0x40dd, 0xbd, 0x34, 0xc3, 0x8a, 0xfe, 0x2b, 0xbd, 0xeb	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer at **EFI_TPL_CALLBACK**	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. 2. Call **Stall()** with 3.5 seconds. 3. Call **SetWatchdogTimer()** to disable the watchdog timer. The system should not be reset.

Number	GUID	Assertion	Test Description
5.1.5.1.6	0x79bcdd1e, 0x1ce2, 0x4a08, 0xaf, 0x85, 0xe8, 0xe8, 0xc1, 0xda, 0x88, 0xbe	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer at **EFI_TPL_NOTIFY**	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. 2. Call **Stall()** with 3.5 seconds. 3. Call **SetWatchdogTimer()** to disable the watchdog timer. The system should not be reset.
5.1.5.1.7	0x021fae0d, 0xcca8, 0x4658, 0x92, 0xab, 0x40, 0x37, 0xc2, 0x23, 0xe8, 0x0f	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer at **EFI_TPL_APPLICATION**	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. 2. Call **Stall()** with 6.5 seconds. The system should be reset in stall.
5.1.5.1.8	0x13dcf833, 0x8209, 0x43d3, 0xb6, 0x70, 0x30, 0x8c, 0x35, 0x2b, 0x51, 0x1f	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer at **EFI_TPL_CALLBACK**	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. 2. Call **Stall()** with 6.5 seconds. The system should be reset in stall.
5.1.5.1.9	0xa2e5497c, 0xac0a, 0x428a, 0xbc, 0x6d, 0xf5, 0x12, 0xfb, 0xc0, 0x70, 0x70	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer at **EFI_TPL_NOTIFY**	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. 2. Call **Stall()** with 6.5 seconds. The system should be reset in stall.
5.1.5.1.10	0x6cf828d1, 0x1871, 0x4bfe, 0x8c, 0x07, 0x71, 0x14, 0x03, 0x7a, 0x0d, 0x7f	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer twice at **EFI_TPL_APPLICATION**.	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. The return code must be **EFI_SUCCESS**.
5.1.5.1.11	0x0af6cd64, 0x1ad9, 0x4e60, 0x97, 0x38, 0x41, 0x4a, 0xe4, 0x73, 0x77, 0x10	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer twice at **EFI_TPL_CALLBACK**.	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. The return code must be **EFI_SUCCESS**.
5.1.5.1.12	0xd6c8200e, 0xf3e0, 0x46ed, 0xb0, 0x14, 0xfe, 0x35, 0x7d, 0xe4, 0xa1, 0xa7	**BS.SetWatchdogTimer – SetWatchdogTimer()** enables the watchdog timer twice at **EFI_TPL_NOTIFY**.	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. The return code must be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.5.1.13	0xf2eb72b7, 0x07ec, 0x4d8e, 0xb6, 0x0f, 0x2c, 0x60, 0xf8, 0x53, 0xbb, 0x62	BS.SetWatchdogTimer – SetWatchdogTimer() enables the watchdog timer twice at EFI_TPL_APPLICATION.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. 2. Call SetWatchdogTimer() again with Timeout is 10 seconds. The return code must be EFI_SUCCESS.
5.1.5.1.14	0xf0e7c390, 0x9d0f, 0x42ca, 0x91, 0x15, 0x42, 0x31, 0x30, 0x1a, 0x54, 0x50	BS.SetWatchdogTimer – SetWatchdogTimer() enables the watchdog timer twice at EFI_TPL_CALLBACK.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. 2. Call SetWatchdogTimer() again with Timeout is 10 seconds. The return code must be EFI_SUCCESS.
5.1.5.1.15	0xf60fc2cb, 0x12df, 0x4147, 0xb0, 0x87, 0x77, 0x8e, 0x9e, 0xdb, 0xa3, 0xb9	BS.SetWatchdogTimer – SetWatchdogTimer() enables the watchdog timer twice at EFI_TPL_NOTIFY.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. 2. Call SetWatchdogTimer() again with Timeout is 10 seconds. The return code must be EFI_SUCCESS.
5.1.5.1.16	0x6c75d979, 0x2e6f, 0x4185, 0x84, 0xa3, 0x6b, 0xd0, 0x90, 0x36, 0x15, 0x4a	BS.SetWatchdogTimer – SetWatchdogTimer() enables the watchdog timer twice at EFI_TPL_APPLICATION.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. 2. Call SetWatchdogTimer() again with Timeout is 10 seconds. 3. Call Stall() with 8.5 seconds. 4. Call SetWatchdogTimer() to disable the watchdog timer. The system should not be reset.

Number	GUID	Assertion	Test Description
5.1.5.1.17	0xe728070e, 0x3393, 0x4798, 0xa2, 0x1a, 0x8e, 0x53, 0x40, 0xb3, 0xfc, 0x61	`BS.SetWatchdogTimer – SetWatchdogTimer()` enables the watchdog timer twice at `EFI_TPL_CALLBACK`.	1. Call `SetWatchdogTimer()` with `Timeout` is 5 seconds. 2. Call `SetWatchdogTimer()` again with `Timeout` is 10 seconds. 3. Call `Stall()` with 8.5 seconds. 4. Call `SetWatchdogTimer()` to disable the watchdog timer. The system should not be reset.
5.1.5.1.18	0xe70ae9bb, 0x403b, 0x42ff, 0x8f, 0x64, 0xa4, 0xdf, 0xf9, 0x24, 0x29, 0xed	`BS.SetWatchdogTimer – SetWatchdogTimer()` enables the watchdog timer twice at `EFI_TPL_NOTIFY`.	1. Call `SetWatchdogTimer()` with `Timeout` is 5 seconds. 2. Call `SetWatchdogTimer()` again with `Timeout` is 10 seconds. 3. Call `Stall()` with 8.5 seconds. 4. Call `SetWatchdogTimer()` to disable the watchdog timer. The system should not be reset.
5.1.5.1.19	0xf799cc16, 0xaccb, 0x4d6d, 0xa8, 0x61, 0x90, 0x6c, 0x6a, 0xea, 0x65, 0x09	`BS.SetWatchdogTimer – SetWatchdogTimer()` enables the watchdog timer twice at `EFI_TPL_APPLICATION`.	1. Call `SetWatchdogTimer()` with `Timeout` is 5 seconds. 2. Call `SetWatchdogTimer()` again with `Timeout` is 10 seconds. 3. Call `Stall()` with 11.5 seconds. The system should be reset in stall.
5.1.5.1.20	0xbb913ccf, 0x026f, 0x4e83, 0xa3, 0x86, 0x24, 0x81, 0xa1, 0xe5, 0x87, 0x6a	`BS.SetWatchdogTimer – SetWatchdogTimer()` enables the watchdog timer twice at `EFI_TPL_CALLBACK`.	1. Call `SetWatchdogTimer()` with `Timeout` is 5 seconds. 2. Call `SetWatchdogTimer()` again with `Timeout` is 10 seconds. 3. Call `Stall()` with 11.5 seconds. The system should be reset in stall.

Number	GUID	Assertion	Test Description
5.1.5.1.21	0x135894cb, 0xc6e3, 0x4345, 0xb0, 0x3b, 0xfd, 0x36, 0x97, 0x10, 0x3f, 0x03	BS.SetWatchdogTimer – SetWatchdogTimer() enables the watchdog timer twice at EFI_TPL_NOTIFY.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. 2. Call SetWatchdogTimer() again with Timeout is 10 seconds. 3. Call Stall() with 11.5 seconds. The system should be reset in stall.
5.1.5.1.22	0x0143203e, 0x56b4, 0x40a3, 0x9e, 0x82, 0xfe, 0xb9, 0x38, 0xb2, 0x68, 0xa0	BS.SetWatchdogTimer – SetWatchdogTimer() disables the watchdog timer at EFI_TPL_APPLICATION.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. The return code should be EFI_SUCCESS.
5.1.5.1.23	0x3cb96e47, 0xec97, 0x4bd1, 0x85, 0x03, 0xa6, 0xcf, 0x2f, 0xd6, 0x15, 0x15	BS.SetWatchdogTimer – SetWatchdogTimer() disables the watchdog timer at EFI_TPL_CALLBACK.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. The return code should be EFI_SUCCESS.
5.1.5.1.24	0xb7d32717, 0xc4af, 0x41ca, 0xab, 0xf7, 0xc3, 0xd2, 0xf8, 0xd2, 0xa9, 0xb1	BS.SetWatchdogTimer – SetWatchdogTimer() disables the watchdog timer at EFI_TPL_NOTIFY.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. The return code should be EFI_SUCCESS.
5.1.5.1.25	0x2d2ef875, 0x4ca4, 0x49c1, 0xb4, 0xb3, 0x42, 0x30, 0x4c, 0xdb, 0x4d, 0x01	BS.SetWatchdogTimer – SetWatchdogTimer() disables the watchdog timer at EFI_TPL_APPLICATION.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. 2. Call SetWatchdogTimer() with Timeout is 0 seconds to disable it. The return code should be EFI_SUCCESS.
5.1.5.1.26	0xae9638a4, 0xad2e, 0x426b, 0xb3, 0x2f, 0x25, 0x1d, 0x02, 0x09, 0xf6, 0x1b	BS.SetWatchdogTimer – SetWatchdogTimer() disables the watchdog timer at EFI_TPL_CALLBACK.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. 2. Call SetWatchdogTimer() with Timeout is 0 seconds to disable it. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.5.1.27	0x6d1ada77, 0x43fa, 0x4502, 0x87, 0x71, 0xea, 0xbf, 0x48, 0xff, 0x9b, 0x90	BS.SetWatchdogTimer – SetWatchdogTimer() disables the watchdog timer at EFI_TPL_NOTIFY.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. 2. Call SetWatchdogTimer() with Timeout is 0 seconds to disable it. The return code should be EFI_SUCCESS.
5.1.5.1.28	0x2fdd96ef, 0x8b9f, 0x4a4e, 0xa3, 0xb7, 0xae, 0x13, 0xf8, 0x17, 0xbd, 0x2b	BS.SetWatchdogTimer – SetWatchdogTimer() disables the watchdog timer twice at EFI_TPL_APPLICATION.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. 2. Call SetWatchdogTimer() with Timeout is 0 seconds to disable it. 3. Call SetWatchdogTimer() with Timeout is 0 seconds again. The return code should be EFI_SUCCESS.
5.1.5.1.29	0x55b55a8a, 0x0adb, 0x4ad0, 0xac, 0x45, 0x83, 0xf4, 0xf9, 0x55, 0x6d, 0x61	BS.SetWatchdogTimer – SetWatchdogTimer() disables the watchdog timer twice at EFI_TPL_CALLBACK.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. 2. Call SetWatchdogTimer() with Timeout is 0 seconds to disable it. 3. Call SetWatchdogTimer() with Timeout is 0 seconds again. The return code should be EFI_SUCCESS.
5.1.5.1.30	0x67f3f8fc, 0x56dd, 0x49b9, 0xad, 0x13, 0x17, 0x84, 0x4e, 0xf6, 0x54, 0xeb	BS.SetWatchdogTimer – SetWatchdogTimer() disables the watchdog timer twice at EFI_TPL_NOTIFY.	1. Call SetWatchdogTimer() with Timeout is 5 seconds. 2. Call SetWatchdogTimer() with Timeout is 0 seconds to disable it. 3. Call SetWatchdogTimer() with Timeout is 0 seconds again. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.5.1.31	0x745345a0, 0x216b, 0x42c0, 0xb2, 0xf5, 0xa7, 0xae, 0x0d, 0x27, 0x75, 0x46	**BS.SetWatchdogTimer – SetWatchdogTimer()** disables the watchdog timer at **EFI_TPL_APPLICATION**.	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. 2. Call **SetWatchdogTimer()** with **Timeout** is 0 seconds to disable it. 3. Call **SetWatchdogTimer()** with **Timeout** is 0 seconds again. 4. Call **Stall()** with 6 seconds. The system should not be reset.
5.1.5.1.32	0x52279d8d, 0x1a05, 0x4c97, 0x8e, 0x09, 0x16, 0xf7, 0x15, 0x3c, 0xac, 0x3f	**BS.SetWatchdogTimer – SetWatchdogTimer()** disables the watchdog timer at **EFI_TPL_CALLBACK**.	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. 2. Call **SetWatchdogTimer()** with **Timeout** is 0 seconds to disable it. 3. Call **SetWatchdogTimer()** with **Timeout** is 0 seconds again. 4. Call **Stall()** with 6 seconds. The system should not be reset.
5.1.5.1.33	0x6d2dfb29, 0x4989, 0x4b89, 0xb7, 0x0a, 0x77, 0xfe, 0x56, 0x2a, 0x0a, 0x79	**BS.SetWatchdogTimer – SetWatchdogTimer()** disables the watchdog timer at **EFI_TPL_NOTIFY**.	1. Call **SetWatchdogTimer()** with **Timeout** is 5 seconds. 2. Call **SetWatchdogTimer()** with **Timeout** is 0 seconds to disable it. 3. Call **SetWatchdogTimer()** with **Timeout** is 0 seconds again. 4. Call **Stall()** with 6 seconds. The system should not be reset.

3.6.2 Stall()

Number	GUID	Assertion	Test Description
5.1.5.2.1	0x9c41568f, 0xa409, 0x4951, 0x9a, 0xc8, 0xd2, 0x70, 0xfa, 0x62, 0xf8, 0xfa	`BS.Stall – Stall()` returns `EFI_SUCCESS` with 10 seconds at `EFI_TPL_APPLICATION`.	1. Call `Stall()` with `Microseconds` is 10000000. The return code should be `EFI_SUCCESS`.
5.1.5.2.2	0x10c23746, 0xd001, 0x400a, 0xbe, 0xf8, 0x57, 0x7f, 0x48, 0x59, 0x0d, 0x7a	`BS.Stall – Stall()` returns `EFI_SUCCESS` with 10 seconds at `EFI_TPL_CALLBACK`.	1. Call `Stall()` with `Microseconds` is 10000000. The return code should be `EFI_SUCCESS`.
5.1.5.2.3	0x4d35fc36, 0xca2d, 0x45db, 0xb9, 0x24, 0x16, 0x77, 0x10, 0xc3, 0x2c, 0xe1	`BS.Stall – Stall()` returns `EFI_SUCCESS` with 10 seconds at `EFI_TPL_NOTIFY`.	1. Call `Stall()` with `Microseconds` is 10000000. The return code should be `EFI_SUCCESS`.
5.1.5.2.4	0x93313097, 0x5d74, 0x4b92, 0x85, 0x9a, 0xab, 0x54, 0xe1, 0x10, 0xdc, 0xdc	`BS.Stall – Stall()` stalls the specified duration with 10 seconds at `EFI_TPL_APPLICATION`.	1. Call `Stall()` with `Microseconds` is 10000000. The duration should be about 10 seconds.
5.1.5.2.5	0xe169d151, 0x3067, 0x424d, 0x9e, 0x5e, 0x0d, 0xd7, 0x41, 0xc8, 0xab, 0x30	`BS.Stall – Stall()` stalls the specified duration with 10 seconds at `EFI_TPL_CALLBACK`.	1. Call `Stall()` with `Microseconds` is 10000000. The duration should be about 10 seconds.
5.1.5.2.6	0x8bcca221, 0x796d, 0x4954, 0x97, 0xd8, 0xbd, 0x13, 0x3b, 0x50, 0xd6, 0x46	`BS.Stall – Stall()` stalls the specified duration with 10 seconds at `EFI_TPL_NOTIFY`.	1. Call `Stall()` with `Microseconds` is 10000000. The duration should be about 10 seconds.

3.6.3 CopyMem()

Number	GUID	Assertion	Test Description
5.1.5.3.1	0xa26a435c, 0x2e00, 0x4b1a, 0xa7, 0xe1, 0xaa, 0x2a, 0x44, 0xb8, 0x9a, 0xc7	`BS.CopyMem – CopyMem()` copies non overlapped memory at `EFI_TPL_APPLICATION`.	1. Call `CopyMem()` with the `Source` and `Destination` not overlapped. The source and destination should have the same contents.

Number	GUID	Assertion	Test Description
5.1.5.3.2	0xf0629f29, 0x244c, 0x4360, 0x8f, 0x33, 0xf8, 0x19, 0xbb, 0x73, 0xad, 0x9d	**BS.CopyMem – CopyMem()** copies non overlapped memory at **EFI_TPL_CALLBACK**.	1. Call **CopyMem()** with the **Source** and **Destination** not overlapped. The source and destination should have the same contents.
5.1.5.3.3	0x4cff47d5, 0x21e5, 0x4e5c, 0xba, 0x2e, 0xba, 0xee, 0xec, 0x3c, 0xc8, 0x1f	**BS.CopyMem – CopyMem()** copies non overlapped memory at **EFI_TPL_NOTIFY**.	1. Call **CopyMem()** with the **Source** and **Destination** not overlapped. The source and destination should have the same contents.
5.1.5.3.4	0xba9e7483, 0xdaaa, 0x455b, 0xa8, 0x1e, 0x4a, 0x9a, 0x39, 0xb2, 0x0d, 0xba	**BS.CopyMem – CopyMem()** copies fully overlapped memory at **EFI_TPL_APPLICATION**.	1. Call **CopyMem()** with the **Source** and **Destination** fully overlapped. The source contents should not be changed.
5.1.5.3.5	0x8bed91fa, 0x816b, 0x4024, 0x83, 0xeb, 0xb1, 0x67, 0x81, 0xeb, 0x43, 0xa0	**BS.CopyMem – CopyMem()** copies fully overlapped memory at **EFI_TPL_CALLBACK**.	1. Call **CopyMem()** with the **Source** and **Destination** fully overlapped. The source contents should not be changed.
5.1.5.3.6	0x45f085aa, 0xaf0e, 0x4fa3, 0xb1, 0xfc, 0x62, 0xef, 0x34, 0xc9, 0x7f, 0x8e	**BS.CopyMem – CopyMem()** copies fully overlapped memory at **EFI_TPL_NOTIFY**.	1. Call **CopyMem()** with the **Source** and **Destination** fully overlapped. The source contents should not be changed.
5.1.5.3.7	0x319cc445, 0xae39, 0x42bb, 0x99, 0x67, 0x15, 0x0a, 0xc1, 0x62, 0x45, 0xfb	**BS.CopyMem – CopyMem()** copies top source overlapped memory at **EFI_TPL_APPLICATION**.	1. Call **CopyMem()** with the **Source** and **Destination** in which the top half of source and the bottom half of destination are overlapped. The destination contents should be the same as the source before **CopyMem()**.
5.1.5.3.8	0x46180798, 0x50af, 0x4ac0, 0xa1, 0xe5, 0x74, 0x50, 0x61, 0xf3, 0x17, 0x3f	**BS.CopyMem – CopyMem()** copies top source overlapped memory at **EFI_TPL_CALLBACK**.	1. Call **CopyMem()** with the **Source** and **Destination** in which the top half of source and the bottom half of destination are overlapped. The destination contents should be the same as the source before **CopyMem()**.

Number	GUID	Assertion	Test Description
5.1.5.3.9	0xcf0ea49d, 0xb03f, 0x41c8, 0xae, 0xd6, 0x0e, 0x37, 0x6f, 0x80, 0x30, 0x7c	`BS.CopyMem – CopyMem()` copies top source overlapped memory at `EFI_TPL_NOTIFY`.	1. Call `CopyMem()` with the `Source` and `Destination` in which the top half of source and the bottom half of destination are overlapped. The destination contents should be the same as the source before `CopyMem()`.
5.1.5.3.10	0x1ac0daf5, 0x5dc0, 0x4315, 0xa2, 0xe5, 0x7f, 0x80, 0x18, 0x5e, 0x1d, 0x2c	`BS.CopyMem – CopyMem()` copies top source overlapped memory at `EFI_TPL_APPLICATION`.	1. Call `CopyMem()` with the `Source` and `Destination` in which the top of source and the bottom of destination are overlapped. Only 1 byte is not overlapped. The destination contents should be the same as the source before `CopyMem()`.
5.1.5.3.11	0x0e16a1dd, 0x0aff, 0x451d, 0x80, 0xd6, 0xe3, 0x9c, 0x43, 0x4f, 0xe6, 0xa3	`BS.CopyMem – CopyMem()` copies top source overlapped memory at `EFI_TPL_CALLBACK`.	1. Call `CopyMem()` with the `Source` and `Destination` in which the top of source and the bottom of destination are overlapped. Only 1 byte is not overlapped. The destination contents should be the same as the source before `CopyMem()`.
5.1.5.3.12	0x268e92a3, 0x7073, 0x428f, 0xbc, 0xfe, 0x32, 0x29, 0xe9, 0x10, 0x66, 0x61	`BS.CopyMem – CopyMem()` copies top source overlapped memory at `EFI_TPL_NOTIFY`.	1. Call `CopyMem()` with the `Source` and `Destination` in which the top of source and the bottom of destination are overlapped. Only 1 byte is not overlapped. The destination contents should be the same as the source before `CopyMem()`.
5.1.5.3.13	0x951403c5, 0x8252, 0x4013, 0x83, 0xd8, 0x51, 0xd0, 0x7e, 0x1d, 0x27, 0x66	`BS.CopyMem – CopyMem()` copies bottom source overlapped memory at `EFI_TPL_APPLICATION`.	1. Call `CopyMem()` with the `Source` and `Destination` in which the bottom half of source and the top half of destination are overlapped. The destination contents should be the same as the source before `CopyMem()`.

Number	GUID	Assertion	Test Description
5.1.5.3.14	0xc855adf4, 0x3b1f, 0x4317, 0x92, 0xd8, 0x72, 0x56, 0x7b, 0x00, 0xa8, 0xe2	BS.CopyMem – CopyMem() copies bottom source overlapped memory at EFI_TPL_CALLBACK.	1. Call CopyMem() with the Source and Destination in which the bottom half of source and the top half of destination are overlapped. The destination contents should be the same as the source before CopyMem().
5.1.5.3.15	0x34ac7d4a, 0x00ae, 0x4a95, 0xa3, 0x18, 0xea, 0x5a, 0x47, 0x1f, 0xde, 0xf2	BS.CopyMem – CopyMem() copies bottom source overlapped memory at EFI_TPL_NOTIFY.	1. Call CopyMem() with the Source and Destination in which the bottom half of source and the top half of destination are overlapped. The destination contents should be the same as the source before CopyMem().
5.1.5.3.16	0xafb876cf, 0xe9c3, 0x4980, 0xb7, 0x40, 0xe4, 0x6d, 0x03, 0x9b, 0xfd, 0xf7	BS.CopyMem – CopyMem() copies bottom source overlapped memory at EFI_TPL_APPLICATION.	1. Call CopyMem() with the Source and Destination in which the bottom half of source and the top half of destination are overlapped. Only 1 byte is not overlapped. The destination contents should be the same as the source before CopyMem().
5.1.5.3.17	0x88d469f3, 0x5538, 0x426f, 0x9e, 0x4f, 0x28, 0x3f, 0xe2, 0x7c, 0x25, 0x8b	BS.CopyMem – CopyMem() copies bottom source overlapped memory at EFI_TPL_CALLBACK.	1. Call CopyMem() with the Source and Destination in which the bottom half of source and the top half of destination are overlapped. Only 1 byte is not overlapped. The destination contents should be the same as the source before CopyMem().
5.1.5.3.18	0x939a7d40, 0x21c1, 0x4472, 0xa7, 0x2e, 0xdd, 0x3f, 0xe2, 0x43, 0x33, 0xe0	BS.CopyMem – CopyMem() copies bottom source overlapped memory at EFI_TPL_NOTIFY.	1. Call CopyMem() with the Source and Destination in which the bottom half of source and the top half of destination are overlapped. Only 1 byte is not overlapped. The destination contents should be the same as the source before CopyMem().

Number	GUID	Assertion	Test Description
5.1.5.3.19	0xb3c59c5b, 0x3e34, 0x466e, 0xb4, 0x30, 0x1c, 0x24, 0x8b, 0x1b, 0x41, 0x24	`BS.CopyMem – CopyMem()` does not copy memory with `Length` is 0 at `EFI_TPL_APPLICATION`.	1. Call `CopyMem()` with the `Length` is 0. The contents in the `Destination` should not be changed.
5.1.5.3.20	0x86b68d03, 0x1543, 0x48aa, 0x82, 0xdb, 0xf9, 0x85, 0x6e, 0xcc, 0x71, 0xa6	`BS.CopyMem – CopyMem()` does not copy memory with `Length` is 0 at `EFI_TPL_CALLBACK`.	1. Call `CopyMem()` with the `Length` is 0. The contents in the `Destination` should not be changed.
5.1.5.3.21	0x040d9af9, 0x6e5a, 0x4ddb, 0xa9, 0x93, 0x36, 0xfc, 0x8a, 0xe6, 0x2f, 0xaa	`BS.CopyMem – CopyMem()` does not copy memory with `Length` is 0 at `EFI_TPL_NOTIFY`.	1. Call `CopyMem()` with the `Length` is 0. The contents in the `Destination` should not be changed.

3.6.4 SetMem()

Number	GUID	Assertion	Test Description
5.1.5.4.1	0x9130e120, 0xa8ad, 0x499d, 0x97, 0xb8, 0xed, 0xbe, 0x59, 0x02, 0x64, 0x3a	`BS.SetMem – SetMem()` sets the specified value at `EFI_TPL_APPLICATION`.	1. Call `SetMem()` to set a buffer to a predefined value. The buffer should be filled with the predefined value.
5.1.5.4.2	0xc03d5d65, 0xb103, 0x4c35, 0xb3, 0xff, 0xe5, 0x2a, 0xf3, 0xc6, 0x06, 0x3d	`BS.SetMem – SetMem()` sets the specified value at `EFI_TPL_CALLBACK`.	1. Call `SetMem()` to set a buffer to a predefined value. The buffer should be filled with the predefined value.
5.1.5.4.3	0xabb87276, 0x13bc, 0x47fa, 0xa5, 0x22, 0xe3, 0xa1, 0x5b, 0x1a, 0x9d, 0xdb	`BS.SetMem – SetMem()` sets the specified value at `EFI_TPL_NOTIFY`.	1. Call `SetMem()` to set a buffer to a predefined value. The buffer should be filled with the predefined value.
5.1.5.4.4	0x0db11970, 0xcd34, 0x4a38, 0xaa, 0x89, 0xb4, 0xb8, 0xd5, 0xc2, 0x19, 0x78	`BS.SetMem – SetMem()` does not set memory with `Size` is 0 at `EFI_TPL_APPLICATION`.	1. Call `SetMem()` with `Size` is 0. The contents in the buffer should not be changed.
5.1.5.4.5	0x37833e1b, 0xd882, 0x4614, 0xa8, 0x58, 0xfb, 0x96, 0x88, 0xf9, 0x9b, 0x1d	`BS.SetMem – SetMem()` does not set memory with `Size` is 0 at `EFI_TPL_CALLBACK`.	1. Call `SetMem()` with `Size` is 0. The contents in the buffer should not be changed.
5.1.5.4.6	0x198b78c3, 0xaf1e, 0x4d41, 0xa4, 0x41, 0xd1, 0xaf, 0x67, 0x0b, 0xa7, 0xbf	`BS.SetMem – SetMem()` does not set memory with `Size` is 0 at `EFI_TPL_NOTIFY`.	1. Call `SetMem()` with `Size` is 0. The contents in the buffer should not be changed.
5.1.5.4.7	0xfb7fb608, 0x6d80, 0x47bd, 0x89, 0x7c, 0x17, 0xbf, 0x76, 0xde, 0x8f, 0x1c	`BS.SetMem – SetMem()` sets not 4-byte aligned memory at `EFI_TPL_APPLICATION`.	1. Call `SetMem()` to set a buffer to a predefined value. The `Buffer` is not 4-byte aligned. The buffer should be filled with the predefined value.
5.1.5.4.8	0x54927bc1, 0xbf3c, 0x4711, 0xa9, 0x1e, 0xb1, 0xe0, 0x1a, 0xa3, 0xcd, 0x64	`BS.SetMem – SetMem()` sets not 4-byte aligned memory at `EFI_TPL_CALLBACK`.	1. Call `SetMem()` to set a buffer to a predefined value. The `Buffer` is not 4-byte aligned. The buffer should be filled with the predefined value.

Number	GUID	Assertion	Test Description
5.1.5.4.9	0x78c81526, 0xe99c, 0x4596, 0xbe, 0x1e, 0x5f, 0x34, 0x3f, 0x2b, 0x2a, 0x03	`BS.SetMem – SetMem()` sets not 4-byte aligned memory at `EFI_TPL_NOTIFY`.	1. Call `SetMem()` to set a buffer to a predefined value. The `Buffer` is not 4-byte aligned. The buffer should be filled with the predefined value.

3.6.5 GetNextMonotonicCount()

Number	GUID	Assertion	Test Description
5.1.5.5.1	0x0b749aae, 0xb782, 0x4cf3, 0xaf, 0x4e, 0xa4, 0x3a, 0xc7, 0x34, 0x5e, 0x79	BS.GetNextMonotonicCount – GetNextMonotonicCount() returns EFI_INVALID_PARAMETER with Count is NULL.	1. Call GetNextMonotonicCount() with Count is NULL. The return code must be EFI_INVALID_PARAMETER.
5.1.5.5.2	0xfdb43f9c, 0x91aa, 0x4628, 0xb9, 0xf7, 0xab, 0xaa, 0xf6, 0x9c, 0xc2, 0x99	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the current count at EFI_TPL_APPLICATION.	1. Call GetNextMonotonicCount() to get the current count. The return codes should be EFI_SUCCESS.
5.1.5.5.3	0xd2f8b66f, 0x0b7f, 0x437e, 0x9c, 0x98, 0xea, 0x72, 0x67, 0xe1, 0xbc, 0xa9	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the current count at EFI_TPL_CALLBACK.	1. Call GetNextMonotonicCount() to get the current count. The return codes should be EFI_SUCCESS.
5.1.5.5.4	0x31ee957c, 0x2ac5, 0x4e81, 0xaa, 0x21, 0x48, 0xd3, 0xff, 0x9c, 0x26, 0xcb	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the current count at EFI_TPL_NOTIFY.	1. Call GetNextMonotonicCount() to get the current count. The return codes should be EFI_SUCCESS.
5.1.5.5.5	0x730b532e, 0xb45f, 0x4a33, 0xab, 0x22, 0x50, 0x97, 0xe9, 0x9f, 0x1d, 0xc4	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the increasing count at EFI_TPL_APPLICATION.	1. Call GetNextMonotonicCount() to get the current count. 2. GetNextMonotonicCount() to get the count again. The return code should be EFI_SUCCESS.
5.1.5.5.6	0x60f6eb2f, 0x8445, 0x4c51, 0xa3, 0xaf, 0xcf, 0xc9, 0x3f, 0xb4, 0x4e, 0x5e	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the increasing count at EFI_TPL_CALLBACK.	1. Call GetNextMonotonicCount() to get the current count. 2. GetNextMonotonicCount() to get the count again. The return code should be EFI_SUCCESS.
5.1.5.5.7	0x07e69104, 0xda46, 0x47b1, 0xb5, 0x8f, 0xa7, 0x41, 0xf7, 0x9a, 0x6b, 0x78	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the increasing count at EFI_TPL_NOTIFY.	1. Call GetNextMonotonicCount() to get the current count. 2. GetNextMonotonicCount() to get the count again. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.5.5.8	0xca4ef318, 0xd9a1, 0x4868, 0xb6, 0xd7, 0xf9, 0x96, 0x41, 0xa1, 0xe2, 0xe8	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the increasing count at EFI_TPL_APPLICATION.	1. Call GetNextMonotonicCount() to get the current count. 2. GetNextMonotonicCount() to get the count again. The return Count should be the previous Count + 1.
5.1.5.5.9	0x6ba5a056, 0xb175, 0x452a, 0x9b, 0x2a, 0x28, 0x3b, 0x1c, 0xc3, 0x28, 0xfb	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the increasing count at EFI_TPL_CALLBACK.	1. Call GetNextMonotonicCount() to get the current count. 2. GetNextMonotonicCount() to get the count again. The return Count should be the previous Count + 1.
5.1.5.5.10	0xe0f339b3, 0xa5ce, 0x42d3, 0xbe, 0x07, 0x67, 0x7b, 0xfa, 0x65, 0x45, 0xd9	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the increasing count at EFI_TPL_NOTIFY.	1. Call GetNextMonotonicCount() to get the current count. 2. GetNextMonotonicCount() to get the count again. The return Count should be the previous Count + 1.
5.1.5.5.11	0x1e49030e, 0x9c2e, 0x4df5, 0xb1, 0x52, 0x46, 0xb3, 0x57, 0xa4, 0xe5, 0x06	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the high 32-bit after reset at EFI_TPL_APPLICATION.	1. Call GetNextMonotonicCount() to get the count. The return codes should be EFI_SUCCESS.
5.1.5.5.12	0x2e10dcf6, 0xe693, 0x492e, 0x9e, 0x34, 0xe6, 0x94, 0x58, 0x31, 0x46, 0xde	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the high 32-bit after reset at EFI_TPL_CALLBACK.	1. Call GetNextMonotonicCount() to get the count. The return codes should be EFI_SUCCESS.
5.1.5.5.13	0x7eaae4e3, 0x50b5, 0x4031, 0xa2, 0xab, 0xcf, 0x9c, 0x76, 0xb1, 0x9b, 0xde	BS.GetNextMonotonicCount – GetNextMonotonicCount() gets the high 32-bit after reset at EFI_TPL_NOTIFY.	1. Call GetNextMonotonicCount() to get the count. The return codes should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.1.5.5.14	0x0878d690, 0x406e, 0x4167, 0xab, 0x44, 0x67, 0x65, 0xec, 0xe0, 0xcc, 0x95	`BS.GetNextMonotonicCount – GetNextMonotonicCount()` gets the high 32-bit after reset at `EFI_TPL_APPLICATION`.	1. Call `GetNextMonotonicCount()` to get the count. Record the high 32-bit value of return count. 2. Reset the system. 3. Call `GetNextMonotonicCount()` to get the count. The return code should be `EFI_SUCCESS`.
5.1.5.5.15	0x958d838a, 0x21a7, 0x4e5b, 0xa0, 0xe6, 0x75, 0x57, 0x74, 0x55, 0xeb, 0xed	`BS.GetNextMonotonicCount – GetNextMonotonicCount()` gets the high 32-bit after reset at `EFI_TPL_CALLBACK`.	1. Call `GetNextMonotonicCount()` to get the count. Record the high 32-bit value of return count. 2. Reset the system. 3. Call `GetNextMonotonicCount()` to get the count. The return code should be `EFI_SUCCESS`.
5.1.5.5.16	0x9611aa6e, 0x85bc, 0x4e20, 0xac, 0x54, 0x68, 0x78, 0xd4, 0xbd, 0xa7, 0x54	`BS.GetNextMonotonicCount – GetNextMonotonicCount()` gets the high 32-bit after reset at `EFI_TPL_NOTIFY`.	1. Call `GetNextMonotonicCount()` to get the count. Record the high 32-bit value of return count. 2. Reset the system. 3. Call `GetNextMonotonicCount()` to get the count. The return code should be `EFI_SUCCESS`.
5.1.5.5.17	0xf48d1c2d, 0x1eba, 0x4e4c, 0xa1, 0x6d, 0x74, 0x8a, 0x01, 0xab, 0xe6, 0xc1	`BS.GetNextMonotonicCount – GetNextMonotonicCount()` gets the high 32-bit after reset at `EFI_TPL_APPLICATION`.	1. Call `GetNextMonotonicCount()` to get the count. Record the high 32-bit value of return count. 2. Reset the system. 3. Call `GetNextMonotonicCount()` to get the count. The high 32-bit of return count should be the previous 32-bit value + 1.

Number	GUID	Assertion	Test Description
5.1.5.5.18	0xe8b96ea0, 0x6413, 0x4947, 0xad, 0x1a, 0x31, 0xee, 0xf8, 0x68, 0xa3, 0x72	`BS.GetNextMonotonicCount – GetNextMonotonicCount()` gets the high 32-bit after reset at `EFI_TPL_CALLBACK`.	1. Call `GetNextMonotonicCount()` to get the count. Record the high 32-bit value of return count. 2. Reset the system. 3. Call `GetNextMonotonicCount()` to get the count. The high 32-bit of return count should be the previous 32-bit value + 1.
5.1.5.5.19	0x0ec16c83, 0x177d, 0x461a, 0x96, 0x22, 0x42, 0x50, 0x8c, 0x99, 0xd9, 0x66	`BS.GetNextMonotonicCount – GetNextMonotonicCount()` gets the high 32-bit after reset at `EFI_TPL_NOTIFY`.	1. Call `GetNextMonotonicCount()` to get the count. Record the high 32-bit value of return count. 2. Reset the system. 3. Call `GetNextMonotonicCount()` to get the count. The high 32-bit of return count should be the previous 32-bit value + 1.

3.6.6 InstallConfigurationTable()

Number	GUID	Assertion	Test Description
5.1.5.6.1	0x12855ef2, 0x5ec3, 0x46ee, 0x84, 0x3a, 0xe5, 0xa8, 0xf3, 0xd5, 0x7b, 0xa4	**BS.InstallConfigurationTable – InstallConfigurationTable()** returns **EFI_INVALID_PARAMETER** with **Guid** is **NULL**.	1. Call **InstallConfigurationTable()** with the **Guid** is **NULL**. The return code must be **EFI_INVALID_PARAMETER**.
5.1.5.6.2	0x7a96cefe, 0x452c, 0x4ea1, 0x8c, 0x75, 0xd9, 0x03, 0x4e, 0x92, 0xed, 0x84	**BS.InstallConfigurationTable – InstallConfigurationTable()** returns **EFI_NOT_FOUND** with **Guid** is not present.	1. Call **InstallConfigurationTable()** with the **Guid** is not present in the System Table and **Table** is **NULL**. The return code must be **EFI_NOT_FOUND**.
5.1.5.6.3	0x31f1c3b2, 0x08ca, 0x404f, 0x8f, 0x4a, 0xbe, 0x94, 0x2c, 0xab, 0x1c, 0x49	**BS.InstallConfigurationTable – InstallConfigurationTable()** adds a new table at **EFI_TPL_APPLICATION**.	1. Call **InstallConfigurationTable()** to add a new configuration table. The return codes should be **EFI_SUCCESS**.
5.1.5.6.4	0xb4d87dcf, 0xa731, 0x4fa7, 0xa9, 0xf1, 0xd8, 0xcf, 0xf2, 0x31, 0x76, 0xff	**BS.InstallConfigurationTable – InstallConfigurationTable()** adds a new table at **EFI_TPL_CALLBACK**.	1. Call **InstallConfigurationTable()** to add a new configuration table. The return codes should be **EFI_SUCCESS**.
5.1.5.6.5	0xce67f821, 0x1add, 0x44b9, 0xa2, 0x9d, 0x9d, 0x25, 0x4c, 0x08, 0x83, 0x78	**BS.InstallConfigurationTable – InstallConfigurationTable()** adds a new table at **EFI_TPL_NOTIFY**.	1. Call **InstallConfigurationTable()** to add a new configuration table. The return codes should be **EFI_SUCCESS**.
5.1.5.6.6	0xd7580a1c, 0xd410, 0x4fe8, 0x93, 0xfc, 0x0b, 0xfe, 0x0b, 0xe8, 0x0d, 0xee	**BS.InstallConfigurationTable – InstallConfigurationTable()** gets an existing table at **EFI_TPL_APPLICATION**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** to get the configuration table. The return codes should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.5.6.7	0x3dc7344c, 0x55aa, 0x4b75, 0xbe, 0x44, 0xca, 0x3a, 0x37, 0xf0, 0xfb, 0x3d	**BS.InstallConfigurationTable – InstallConfigurationTable()** gets an existing table at **EFI_TPL_CALLBACK**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** to get the configuration table. The return codes should be **EFI_SUCCESS**.
5.1.5.6.8	0xeb2460f0, 0x07cc, 0x43a5, 0x8d, 0xa9, 0x79, 0xed, 0x3d, 0x1f, 0x08, 0xd0	**BS.InstallConfigurationTable – InstallConfigurationTable()** gets an existing table at **EFI_TPL_NOTIFY**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** to get the configuration table. The return codes should be **EFI_SUCCESS**.
5.1.5.6.9	0xe0e73667, 0x8cb8, 0x4839, 0xa9, 0x7a, 0x99, 0x0e, 0xb4, 0x3b, 0xfc, 0xfd	**BS.InstallConfigurationTable –** After added system table has corrected CRC32 at **EFI_TPL_APPLICATION**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** to get the configuration table. The system table should have a correct CRC32 value.
5.1.5.6.10	0xea5a3a8e, 0x9579, 0x4a3c, 0x84, 0xb3, 0x0f, 0xb9, 0x22, 0x00, 0x99, 0x18	**BS.InstallConfigurationTable –** After added system table has corrected CRC32 at **EFI_TPL_CALLBACK**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** to get the configuration table. The system table should have a correct CRC32 value.
5.1.5.6.11	0xa1cefe6d, 0xe33d, 0x418f, 0x9f, 0xff, 0x29, 0x3e, 0x75, 0xb1, 0x65, 0xe5	**BS.InstallConfigurationTable –** After added system table has corrected CRC32 at **EFI_TPL_NOTIFY**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** to get the configuration table. The system table should have a correct CRC32 value.

Number	GUID	Assertion	Test Description
5.1.5.6.12	0xad025b1b, 0x06e0, 0x4ba9, 0x84, 0xc9, 0x25, 0x0c, 0x70, 0xa1, 0x64, 0x35	**BS.InstallConfigurationTable** – The list of tables is in runtime services data at **EFI_TPL_APPLICATION**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** to get the configuration table. The list of tables should be at **EfiRuntimeServicesData**.
5.1.5.6.13	0xc393e4e6, 0x56eb, 0x46d0, 0x9f, 0xbb, 0xe2, 0x9e, 0xea, 0x06, 0x33, 0xd2	**BS.InstallConfigurationTable** – The list of tables is in runtime services data at **EFI_TPL_CALLBACK**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** to get the configuration table. The list of tables should be at **EfiRuntimeServicesData**.
5.1.5.6.14	0xc068f1a8, 0x0f7a, 0x4b5e, 0xa5, 0x9f, 0xce, 0x17, 0xa4, 0x52, 0xf4, 0xba	**BS.InstallConfigurationTable** – The list of tables is in runtime services data at **EFI_TPL_NOTIFY**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** to get the configuration table. The list of tables should be at **EfiRuntimeServicesData**.
5.1.5.6.15	0xa8e90505, 0x82c6, 0x48b5, 0x93, 0xda, 0xbf, 0xb0, 0x11, 0x9b, 0x52, 0x0f	**BS.InstallConfigurationTable** – **InstallConfigurationTable()** updates an existing table at **EFI_TPL_APPLICATION**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** with same GUID to update the table. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.1.5.6.16	0x6538a9d9, 0x8146, 0x411e, 0xab, 0xa7, 0x90, 0xe5, 0x6e, 0xb5, 0x33, 0x27	`BS.InstallConfigurationTable – InstallConfigurationTable()` updates an existing table at `EFI_TPL_CALLBACK`.	1. Call `InstallConfigurationTable()` to add a new configuration table. 2. Call `InstallConfigurationTable()` with same GUID to update the table. The return code should be `EFI_SUCCESS`.
5.1.5.6.17	0x30a1994a, 0xaf85, 0x41fe, 0x8d, 0xd9, 0x60, 0x83, 0x01, 0x76, 0x96, 0x3d	`BS.InstallConfigurationTable – InstallConfigurationTable()` updates an existing table at `EFI_TPL_NOTIFY`.	1. Call `InstallConfigurationTable()` to add a new configuration table. 2. Call `InstallConfigurationTable()` with same GUID to update the table. The return code should be `EFI_SUCCESS`.
5.1.5.6.18	0xded94f21, 0x2f3d, 0x45aa, 0x86, 0x87, 0xd2, 0x2e, 0x94, 0x2b, 0xa4, 0x3e	`BS.InstallConfigurationTable – InstallConfigurationTable()` gets the updated table at `EFI_TPL_APPLICATION`.	1. Call `InstallConfigurationTable()` to add a new configuration table. 2. Call `InstallConfigurationTable()` with same GUID to update the table. 3. Call `InstallConfigurationTable()` to get the table. The updated table should be gotten.
5.1.5.6.19	0xccd943d1, 0x356a, 0x49da, 0x9e, 0x18, 0xf1, 0x94, 0x64, 0x83, 0x76, 0x7b	`BS.InstallConfigurationTable – InstallConfigurationTable()` gets the updated table at `EFI_TPL_CALLBACK`.	1. Call `InstallConfigurationTable()` to add a new configuration table. 2. Call `InstallConfigurationTable()` with same GUID to update the table. 3. Call `InstallConfigurationTable()` to get the table. The updated table should be gotten.

Number	GUID	Assertion	Test Description
5.1.5.6.20	0x8e1d8ebb, 0x82af, 0x4f46, 0xa4, 0xdc, 0x99, 0x9b, 0x9a, 0x84, 0xb9, 0x6b	`BS.InstallConfigurationTable` – `InstallConfigurationTable()` gets the updated table at `EFI_TPL_NOTIFY`.	1. Call `InstallConfigurationTable()` to add a new configuration table. 2. Call `InstallConfigurationTable()` with same GUID to update the table. 3. Call `InstallConfigurationTable()` to get the table. The updated table should be gotten.
5.1.5.6.21	0x1b6c204d, 0x953c, 0x4c6e, 0x98, 0xbf, 0xdc, 0x84, 0x46, 0x04, 0x05, 0x65	`BS.InstallConfigurationTable` – After updated system table has corrected CRC32 at `EFI_TPL_APPLICATION`.	1. Call `InstallConfigurationTable()` to add a new configuration table. 2. Call `InstallConfigurationTable()` with same GUID to update the table. 3. Call `InstallConfigurationTable()` to get the table. System table should have a correct CRC32 value.
5.1.5.6.22	0xd5cfb42f, 0xc615, 0x4d56, 0x80, 0x54, 0xe5, 0xc1, 0xdd, 0x48, 0xde, 0xf1	`BS.InstallConfigurationTable` – After updated system table has corrected CRC32 at `EFI_TPL_CALLBACK`.	1. Call `InstallConfigurationTable()` to add a new configuration table. 2. Call `InstallConfigurationTable()` with same GUID to update the table. 3. Call `InstallConfigurationTable()` to get the table. System table should have a correct CRC32 value.

Number	GUID	Assertion	Test Description
5.1.5.6.23	0x4615f33a, 0x57bf, 0x4706, 0x94, 0x88, 0x60, 0xb2, 0x30, 0xae, 0x9e, 0xf5	**BS.InstallConfigurationTable** – After updated system table has corrected CRC32 at **EFI_TPL_NOTIFY**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** with same GUID to update the table. 3. Call **InstallConfigurationTable()** to get the table. System table should have a correct CRC32 value.
5.1.5.6.24	0x58fc9921, 0x329f, 0x416b, 0xad, 0xad, 0xc5, 0xdf, 0x03, 0xf7, 0xd4, 0xde	**BS.InstallConfigurationTable** – The list of tables is in runtime services data at **EFI_TPL_APPLICATION**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** with same GUID to update the table. 3. Call **InstallConfigurationTable()** to get the table. The list of table should be **EfiRuntimeServicesData**.
5.1.5.6.25	0x87451a4f, 0xf1e0, 0x4b21, 0x83, 0xcc, 0xa2, 0x9a, 0x3c, 0xfe, 0xde, 0xcf	**BS.InstallConfigurationTable** – The list of tables is in runtime services data at **EFI_TPL_CALLBACK**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** with same GUID to update the table. 3. Call **InstallConfigurationTable()** to get the table. The list of table should be **EfiRuntimeServicesData**.

Number	GUID	Assertion	Test Description
5.1.5.6.26	0x0d42b29c, 0x2eee, 0x4634, 0x8e, 0x8e, 0x4d, 0x7f, 0x9f, 0xc7, 0xb3, 0x65	`BS.InstallConfigurationTable` – The list of tables is in runtime services data at `EFI_TPL_NOTIFY`.	1. Call `InstallConfigurationTable()` to add a new configuration table. 2. Call `InstallConfigurationTable()` with same GUID to update the table. 3. Call `InstallConfigurationTable()` to get the table. The list of table should be `EfiRuntimeServicesData`.
5.1.5.6.27	0xa6753a34, 0xfe86, 0x4905, 0x88, 0x50, 0x2c, 0xfb, 0x36, 0xf4, 0x03, 0xb9	`BS.InstallConfigurationTable` – `InstallConfigurationTable()` removes the existing table at `EFI_TPL_APPLICATION`.	1. Call `InstallConfigurationTable()` to add a new configuration table. 2. Call `InstallConfigurationTable()` with same GUID and `NULL` table to remove the table. The return code should be `EFI_SUCCESS`.
5.1.5.6.28	0x3ed6faf5, 0x0482, 0x43a2, 0x8a, 0x43, 0x61, 0xcd, 0x11, 0x1e, 0x03, 0x65	`BS.InstallConfigurationTable` – `InstallConfigurationTable()` removes the existing table at `EFI_TPL_CALLBACK`.	1. Call `InstallConfigurationTable()` to add a new configuration table. 2. Call `InstallConfigurationTable()` with same GUID and `NULL` table to remove the table. The return code should be `EFI_SUCCESS`.
5.1.5.6.29	0x57293d64, 0x128c, 0x4d07, 0x93, 0x73, 0x1d, 0xea, 0x16, 0x4c, 0x61, 0x96	`BS.InstallConfigurationTable` – `InstallConfigurationTable()` removes the existing table at `EFI_TPL_NOTIFY`.	1. Call `InstallConfigurationTable()` to add a new configuration table. 2. Call `InstallConfigurationTable()` with same GUID and `NULL` table to remove the table. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.1.5.6.30	0x375247e6, 0x440b, 0x439f, 0xa5, 0x6c, 0x0b, 0xe8, 0x13, 0x39, 0xde, 0x2b	BS.InstallConfigurationTable – InstallConfigurationTable() removes the existing table at EFI_TPL_APPLICATION.	1. Call InstallConfigurationTable() to add a new configuration table. 2. Call InstallConfigurationTable() with same GUID and NULL table to remove the table. The table should be removed from configuration tables.
5.1.5.6.31	0x3ddfd695, 0x2338, 0x4582, 0xbf, 0x53, 0x63, 0xd2, 0xc3, 0x38, 0x87, 0x3e	BS.InstallConfigurationTable – InstallConfigurationTable() removes the existing table at EFI_TPL_CALLBACK.	1. Call InstallConfigurationTable() to add a new configuration table. 2. Call InstallConfigurationTable() with same GUID and NULL table to remove the table. The table should be removed from configuration tables.
5.1.5.6.32	0x0988164f, 0xb3e6, 0x40ca, 0x9f, 0x94, 0x19, 0xb2, 0x16, 0x65, 0xb1, 0x70	BS.InstallConfigurationTable – InstallConfigurationTable() removes the existing table at EFI_TPL_NOTIFY.	1. Call InstallConfigurationTable() to add a new configuration table. 2. Call InstallConfigurationTable() with same GUID and NULL table to remove the table. The table should be removed from configuration tables.
5.1.5.6.33	0xfccbfa48, 0x68a6, 0x4d2f, 0xa6, 0x63, 0xf8, 0x40, 0x6e, 0x00, 0x79, 0x2e	BS.InstallConfigurationTable – After removed system table has corrected CRC32 at EFI_TPL_APPLICATION.	1. Call InstallConfigurationTable() to add a new configuration table. 2. Call InstallConfigurationTable() with same GUID and NULL table to remove the table. System table should have a correct CRC32 value.

Number	GUID	Assertion	Test Description
5.1.5.6.34	0xbb1f8b9c, 0x563e, 0x42d9, 0x88, 0x6c, 0x25, 0xa5, 0x1f, 0xbb, 0x26, 0x8f	**BS.InstallConfigurationTable –** After removed system table has corrected CRC32 at **EFI_TPL_CALLBACK**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** with same GUID and **NULL** table to remove the table. System table should have a correct CRC32 value.
5.1.5.6.35	0xf4a0a3df, 0xddf9, 0x467d, 0xb0, 0xd3, 0x73, 0xc1, 0x43, 0xda, 0x59, 0x01	**BS.InstallConfigurationTable –** After removed system table has corrected CRC32 at **EFI_TPL_NOTIFY**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** with same GUID and **NULL** table to remove the table. System table should have a correct CRC32 value.
5.1.5.6.36	0xf2130268, 0x6c2f, 0x4629, 0x9e, 0xef, 0x21, 0xa0, 0x64, 0x95, 0x2e, 0x0b	**BS.InstallConfigurationTable –** The list of tables is in runtime services data at **EFI_TPL_APPLICATION**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** with same GUID and **NULL** table to remove the table. The list of table should be **EfiRuntimeServicesData**.
5.1.5.6.37	0x66333b3e, 0x26f9, 0x4334, 0x9f, 0x90, 0x66, 0x11, 0x05, 0x9d, 0x07, 0xb4	**BS.InstallConfigurationTable –** The list of tables is in runtime services data at **EFI_TPL_CALLBACK**.	1. Call **InstallConfigurationTable()** to add a new configuration table. 2. Call **InstallConfigurationTable()** with same GUID and **NULL** table to remove the table. The list of table should be **EfiRuntimeServicesData**.

Number	GUID	Assertion	Test Description
5.1.5.6.38	0x5fab38c1, 0x5089, 0x488b, 0xb7, 0x65, 0x4c, 0xe9, 0x76, 0xe4, 0x83, 0x6e	**`BS.InstallConfigurationTable` –** The list of tables is in runtime services data at **`EFI_TPL_NOTIFY`**.	1. Call **`InstallConfigurationTable()`** to add a new configuration table. 2. Call **`InstallConfigurationTable()`** with same GUID and **`NULL`** table to remove the table. The list of table should be **`EfiRuntimeServicesData`**.

3.6.7 CalculateCrc32()

Number	GUID	Assertion	Test Description
5.1.5.7.1	0x3a1d2ad6, 0x743c, 0x47f0, 0x87, 0x51, 0x9f, 0x4a, 0x24, 0xc8, 0xcb, 0xf6	`BS.CalculateCrc32 – CalculateCrc32()` returns `EFI_INVALID_PARAMETER` with `Data` is `NULL`.	1. Call `CalculateCrc32()` with the `Data` is `NULL`. The return code must be `EFI_INVALID_PARAMETER`.
5.1.5.7.2	0x44f81362, 0xb579, 0x4691, 0xa0, 0x84, 0x40, 0xc2, 0x24, 0x14, 0x0c, 0x84	`BS.CalculateCrc32 – CalculateCrc32()` returns `EFI_INVALID_PARAMETER` with `Crc32` is `NULL`.	1. Call `CalculateCrc32()` with the `Crc32` is `NULL`. The return code must be `EFI_INVALID_PARAMETER`.
5.1.5.7.3	0xe76d175a, 0xc32f, 0x4279, 0xab, 0x4b, 0x3a, 0x80, 0x6c, 0x97, 0xf4, 0x6b	`BS.CalculateCrc32 – CalculateCrc32()` returns `EFI_INVALID_PARAMETER` with `DataSize` is 0.	1. Call `CalculateCrc32()` when the `DataSize` is 0. The return code must be `EFI_INVALID_PARAMETER`.
5.1.5.7.4	0xffbcedcf, 0xcc49, 0x4b4b, 0xa1, 0x70, 0x2f, 0xa8, 0x57, 0x0b, 0x59, 0xd9	`BS.CalculateCrc32 – CalculateCrc32()` gets correct value to system table at `EFI_TPL_APPLICATION`.	1. Store the CRC32 value of the system table and set the CRC32 value of the system table to 0. 2. Call `CalculateCrc32()` to calculate the CRC32 value of the system table. The calculated value should be the same as the stored value. 3. Restore the CRC32 value of the system table.
5.1.5.7.5	0xeb007e3c, 0xd916, 0x4ae6, 0x82, 0x9a, 0x4c, 0x5a, 0x4d, 0x28, 0x2c, 0x18	`BS.CalculateCrc32 – CalculateCrc32()` gets correct value to system table at `EFI_TPL_CALLBACK`.	1. Store the CRC32 value of the system table and set the CRC32 value of the system table to 0. 2. Call `CalculateCrc32()` to calculate the CRC32 value of the system table. The calculated value should be the same as the stored value. 3. Restore the CRC32 value of the system table.

Number	GUID	Assertion	Test Description
5.1.5.7.6	0x055b72de, 0x02e0, 0x4490, 0xb6, 0x52, 0x95, 0xeb, 0x9e, 0xea, 0x46, 0xc1	`BS.CalculateCrc32 – CalculateCrc32()` gets correct value to system table at `EFI_TPL_NOTIFY`.	1. Store the CRC32 value of the system table and set the CRC32 value of the system table to 0. 2. Call `CalculateCrc32()` to calculate the CRC32 value of the system table. The calculated value should be the same as the stored value. 3. Restore the CRC32 value of the system table.

4 Services Runtime Services Test

4.1 Variable Services Test

Reference Document:

UEFI Specification, Variable Services Section.

Table 6. Variable Services FunctionsGetVariable()

Name	Type	Description
GetVariable()	Runtime	Returns the value of a variable.
GetNextVariableName()	Runtime	Enumerates the current variable names.
SetVariable()	Runtime	Sets the value of a variable.
QueryVariableInfo()	Runtime	Queries the information about the variables.

4.1.1 GetVariable()

Number	GUID	Assertion	Test Description
5.2.1.1.1	0xb0d54fee, 0x2787, 0x4d2d, 0xbf, 0x98, 0x73, 0xa0, 0xcd, 0x7f, 0xe9, 0x5d	**RT.GetVariable – GetVariable()** returns **EFI_INVALID_PARAMETER** with a ***VariableName*** value of **NULL**.	1. Call **GetVariable()** service with a ***VariableName*** value of **NULL**. The return code must be **EFI_INVALID_PARAMETER**.
5.2.1.1.2	0x390c5e26, 0x9b46, 0x4974, 0xb3, 0x2d, 0x2b, 0xb1, 0xd4, 0x05, 0xb0, 0xd7	**RT.GetVariable – GetVariable()** returns **EFI_INVALID_PARAMETER** with a ***VendorGuid*** value of **NULL**.	1. Call **GetVariable()** service with a ***VendorGuid*** value of **NULL**. The return code must be **EFI_INVALID_PARAMETER**.
5.2.1.1.3	0x176354a6, 0x1088, 0x474f, 0xbf, 0x6f, 0x95, 0x8c, 0x1c, 0xc3, 0x40, 0x8f	**RT.GetVariable – GetVariable()** returns **EFI_INVALID_PARAMETER** with a ***DataSize*** value of **NULL**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **GetVariable()** service to get the test variable while the ***DataSize*** is **NULL**. The return code must be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.2.1.1.4	0x400ab801, 0xf6c6, 0x4d04, 0xa0, 0x42, 0xa2, 0x15, 0x0b, 0xd5, 0xb6, 0x2a	`RT.GetVariable – GetVariable()` returns `EFI_INVALID_PARAMETER` with a ***Data*** value of `NULL`.	1. Call `SetVariable()` service to insert a test variable. 2. Call `GetVariable()` service to get the test variable while the ***Data*** is `NULL`. The return code must be `EFI_INVALID_PARAMETER`.
5.2.1.1.5	0x9b704b3d, 0x05a4, 0x4147, 0xb2, 0x55, 0x35, 0xbc, 0x3d, 0xd6, 0xcc, 0x24	`RT.GetVariable – GetVariable()` returns `EFI_NOT_FOUND` with a nonexistent variable.	1. Call `SetVariable()` service to insert a test variable. 2. Call `SetVariable()` service to delete the test variable. 3. Call `GetVariable()` service to get the test variable. The return code must be `EFI_NOT_FOUND`. 4. Call `SetVariable()` services to insert two variables that are similar to the test variable. 5. Call `GetVariable()` service to get the test variable. The return code must be `EFI_NOT_FOUND`.
5.2.1.1.6	0xd3d915a5, 0xe7b0, 0x4417, 0x9c, 0x2e, 0x1a, 0xa8, 0x42, 0x4d, 0x22, 0x2c	`RT.GetVariable – GetVariable()` returns `EFI_NOT_FOUND` with a nonexistent ***VendorGuid***.	1. Call `SetVariable()` service to insert a test variable with `GUID2`. 2. Call `GetVariable()` service to get the variable with `GUID1`. The return code must be `EFI_NOT_FOUND`.
5.2.1.1.7	0x1562ce35, 0x83e7, 0x48a7, 0xad, 0x71, 0xfa, 0xa4, 0xbe, 0x17, 0x88, 0x46	`RT.GetVariable – GetVariable()` returns `EFI_BUFFER_TOO_SMALL` with a ***DataSize*** value of 0.	1. Call `SetVariable()` service to insert a test variable. 2. Call `GetVariable()` service to get the test variable with a ***DataSize*** value of 0. The return code must be `EFI_BUFFER_TOO_SMALL`, and the returned ***DataSize*** should be the inserted value in step 1.

Number	GUID	Assertion	Test Description
5.2.1.1.8	0x121c17d1, 0xbb0e, 0x4e2e, 0xb2, 0xa5, 0x03, 0x86, 0x2f, 0x46, 0xc0, 0x39	**RT.GetVariable – GetVariable()** returns **EFI_BUFFER_TOO_SMALL** with a ***DataSize*** value of -1.	1. Call **SetVariable()** service to insert a test variable. 2. Call **GetVariable()** service to get the test variable with the inserted ***DataSize*** value of –1. The return code must be **EFI_BUFFER_TOO_SMALL**, and the returned ***DataSize*** should be the inserted value in step 1.
5.2.1.1.9	0xe542e81c, 0x2020, 0x4f3e, 0xa9, 0xb, 0x67, 0xd4, 0xa8, 0xd1, 0x70, 0xb4	**RT.GetVariable – GetVariable()** returns **EFI_BUFFER_TOO_SMALL** with a ***DataSize*** value of 0.	1. Call **SetVariable()** service to insert a test variable. 2. Call **GetVariable()** service to get the test variable with a ***DataSize*** value of 0 and NULL ***Data***. The return code must be **EFI_BUFFER_TOO_SMALL** , and the returned ***DataSize*** should be the inserted value in step 1.
5.2.1.1.10	0xaa35cc00, 0xc55c, 0x42d8, 0xa6, 0xd4, 0x1e, 0xb4, 0x9d, 0xe3, 0xd7, 0x54	**RT.GetVariable – GetVariable()** gets the existing variable without attributes at **EFI_TPL_APPLICATION**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **GetVariable()** service to get the test variable without ***Attributes***. The returned status must be **EFI_SUCCESS**, and the returned ***Data*** and ***DataSize*** must be the same as the data written before.

Number	GUID	Assertion	Test Description
5.2.1.1.11	0x742a9651, 0x9783, 0x43b8, 0x8c, 0x18, 0x47, 0x04, 0xae, 0x41, 0xc3, 0x34	**RT.GetVariable – GetVariable()** gets the existing variable without attributes at **EFI_TPL_CALLBACK**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **GetVariable()** service to get the test variable without ***Attributes***. The returned status must be **EFI_SUCCESS**, and the returned ***Data*** and ***DataSize*** must be the same as the data written before.
5.2.1.1.12	0x90e959d0, 0xbe2c, 0x45fd, 0x85, 0x32, 0x85, 0x21, 0xe4, 0xe0, 0xfb, 0x72	**RT.GetVariable – GetVariable()** gets the existing variable with attributes at **EFI_TPL_APPLICATION**	1. Call **SetVariable()** service to insert a test variable. 2. Call **GetVariable()** service to get the test variable with ***Attributes***. The returned status must be **EFI_SUCCESS**, and the returned ***Attributes***, ***Data*** and ***DataSize*** must be the same as the data written before.
5.2.1.1.13	0x5c8b43b7, 0xec6f, 0x4621, 0xb8, 0x48, 0x6a, 0x40, 0x0f, 0xd8, 0xb3, 0x43	**RT.GetVariable – GetVariable()** gets the existing variable with attributes at **EFI_TPL_CALLBACK**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **GetVariable()** service to get the test variable with ***Attributes***. The returned status must be **EFI_SUCCESS**, and the returned ***Attributes***, ***Data*** and ***DataSize*** must be the same as the data written before.

4.1.2 GetNextVariableName()

Number	GUID	Assertion	Test Description
5.2.1.2.1	0x5826847a, 0x9067, 0x4f9f, 0x88, 0x38, 0x0b, 0xf8, 0xec, 0x20, 0x17, 0x1c	RT.GetNextVariableName – GetNextVariableName() returns EFI_INVALID_PARAMETER with a *VariableNameSize* value of NULL.	1. Call GetNextVariableName() service with a *VariableNameSize* value of NULL. The return code must be EFI_INVALID_PARAMETER.
5.2.1.2.2	0x8e8258dc, 0x6634, 0x4de1, 0x85, 0x7a, 0x60, 0x45, 0x7e, 0xfa, 0x7c, 0x21	RT.GetNextVariableName - GetNextVariableName() returns EFI_INVALID_PARAMETER with a *VariableName* value of NULL.	1. Call GetNextVariableName() service with a *VariableName* value of NULL. The returned status should be EFI_INVALID_PARAMETER.
5.2.1.2.3	0x99a357f0, 0xb6c5, 0x4aec, 0x96, 0x48, 0x34, 0x73, 0x2d, 0x2a, 0x49, 0x50	RT.GetNextVariableName - GetNextVariableName() returns EFI_INVALID_PARAMETER with a *VendorGuid* value of NULL.	1. Call GetNextVariableName() service with a *VendorGuid* value of NULL. The returned status should be EFI_INVALID_PARAMETER.
5.2.1.2.4	0x51c19dba, 0xbaf6, 0x4854, 0xac, 0x09, 0x60, 0x45, 0x47, 0x88, 0x67, 0x98	RT.GetNextVariableName – GetNextVariableName() returns EFI_BUFFER_TOO_SMALL with a *VariableNameSize* value of 2.	1. Call SetVariable() service to insert a test variable. 2. Call GetNextVariableName() service with a *VariableNameSize* value of 2. The returned status should be EFI_BUFFER_TOO_SMALL.
5.2.1.2.5	0xfe09ff82, 0xb289, 0x449f, 0xb0, 0x83, 0x98, 0x1d, 0x68, 0xd9, 0x17, 0xb1	RT.GetNextVariableName – GetNextVariableName() returns EFI_NOT_FOUND after the entire variable list returned.	1. Call SetVariable() service to insert a test variable. 2. Call SetVariable() service to delete the test variable. 3. Call GetNextVariableName() service to traverseall variables. The deleted test variable should not be returned. 4. The last returned status of GetNextVariableName() service should be EFI_NOT_FOUND.

Number	GUID	Assertion	Test Description
5.2.1.2.6	0x12071508, 0x16c7, 0x4e5e, 0xa4, 0x22, 0x59, 0xe0, 0x24, 0x1c, 0xc6, 0x28	**RT.GetNextVariableName – GetNextVariableName()** gets the existing variable at **EFI_TPL_APPLICATION**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **GetNextVariableName()** service to traverseall variables. The test variable should be returned in this loop.
5.2.1.2.7	0xa85043bc, 0x4f0d, 0x47b3, 0x8e, 0x9d, 0x2d, 0xb6, 0xc8, 0xf8, 0xfa, 0xef	**RT.GetNextVariableName – GetNextVariableName()** gets the exist variable at **EFI_TPL_CALLBACK**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **GetNextVariableName()** service to traverseall variables. The test variable should be returned in this loop.

4.1.3 SetVariable()

Number	GUID	Assertion	Test Description
5.2.1.3.1	0x73af529b, 0x3ebe, 0x464a, 0xba, 0x6a, 0xfb, 0x04, 0x7b, 0x56, 0x4f, 0x74	**RT.SetVariable – SetVariable()** returns **EFI_INVALID_PARAMETER** when the ***VariableName*** value is an empty string.	1. Call **SetVariable()** service when the ***VariableName*** value is an empty string. The returned status should be **EFI_INVALID_PARAMETER**.
5.2.1.3.2	0x39e95cbb, 0x6b89, 0x473e, 0x91, 0xba, 0x92, 0x08, 0x2d, 0x1b, 0x94, 0xad	**RT.SetVariable – SetVariable()** returns **EFI_INVALID_PARAMETER** with **RA** only ***Attributes***.	1. Call **SetVariable()** service with **EFI_VARIABLE_RUNTIME_ACCESS** attributes. The returned status should be **EFI_INVALID_PARAMETER**.
5.2.1.3.3	0xf6ef5087, 0x4962, 0x4d71, 0x80, 0x09, 0xdb, 0xe2, 0x78, 0x94, 0x53, 0xe6	**RT.SetVariable – SetVariable()** returns **EFI_INVALID_PARAMETER** with **NV|RA** ***Attributes***.	1. Call **SetVariable()** service with **EFI_VARIABLE_NON_VOLATILE** | **EFI_VARIABLE_RUNTIME_ACCESS** attributes. The returned status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.2.1.3.4	0x65973462, 0x6877, 0x408f, 0x9b, 0xe1, 0x46, 0x69, 0x3e, 0xab, 0x03, 0x84	**RT.SetVariable – SetVariable()** returns **EFI_INVALID_PARAMETER** with a variable that exceeds the maximum size.	1. Call **SetVariable()** service to set a test variable with the size of (UINTN)-1. The returned status should be **EFI_INVALID_PARAMETER**.
5.2.1.3.5	0x6c9cf2ea, 0xcabd, 0x4312, 0xb9, 0xcf, 0x0a, 0x96, 0xc4, 0xf1, 0xea, 0x8b	**RT.SetVariable – SetVariable()** sets a nonexistent variable at **EFI_TPL_APPLICATION**.	1. Call **SetVariable()** service to insert a test variable with **GUID2**. 2. Call **SetVariable()** service to insert a test variable with **GUID1**. The returned status must be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get the test variable with **GUID1** and **GUID2**. The data of both variables should be the same as the values written before.
5.2.1.3.6	0x3ae09eaf, 0x07cd, 0x4320, 0x92, 0xfd, 0xe9, 0xe6, 0x4b, 0x31, 0x6f, 0xe1	**RT.SetVariable – SetVariable()** sets a nonexistent variable at **EFI_TPL_CALLBACK**.	1. Call **SetVariable()** service to insert a test variable with **GUID2**. 2. Call **SetVariable()** service to insert a test variable with **GUID1**. The returned status must be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get the test variable with **GUID1** and **GUID2**. The data of both variables should be the same as the values written before.
5.2.1.3.7	0x7ccde75b, 0x4ef2, 0x40ec, 0x9a, 0xcb, 0x84, 0x7b, 0xb5, 0x29, 0x73, 0xbe	**RT.SetVariable – SetVariable()** sets the existing variable with the data from **EFI_TPL_APPLICATION**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **SetVariable()** service to insert the test variable again with the same data. The returned status should be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get the test variable. The data of the test variable should be unchanged.

Number	GUID	Assertion	Test Description
5.2.1.3.8	0x5b720ad1, 0xd0cc, 0x4be0, 0x93, 0x18, 0x20, 0x1b, 0xac, 0x32, 0x8d, 0x4f	**RT.SetVariable – SetVariable()** sets the existing variable with the data from **EFI_TPL_CALLBACK**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **SetVariable()** service to insert the test variable again with the same data. The returned status should be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get the test variable. The data of the test variable should be unchanged.
5.2.1.3.9	0x2dee62d3, 0xbab7, 0x4d91, 0x8b, 0x47, 0x3e, 0x38, 0x35, 0xd3, 0x88, 0xae	**RT.SetVariable – SetVariable()** sets the existing variable value which is different from the one at **EFI_TPL_APPLICATION**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **SetVariable()** service to insert the test variable again with the different data in which the left part of new data is the same as old data. The returned status should be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get the test variable. The data of the test variable should be changed to the new one.
5.2.1.3.10	0x861a0691, 0x6590, 0x4a28, 0xae, 0x56, 0xaa, 0xcb, 0xf3, 0xf2, 0xbe, 0x99	**RT.SetVariable – SetVariable()** sets the existing variable value which is different from the one at **EFI_TPL_CALLBACK**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **SetVariable()** service to insert the test variable again with the different data in which the left part of new data is the same as the old data The returned status should be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get the test variable. The data of the test variable should be changed to the new one.

Number	GUID	Assertion	Test Description
5.2.1.3.11	0x76198a1a, 0xc63a, 0x4a3b, 0x88, 0xb0, 0xc4, 0x45, 0x39, 0xdd, 0xff, 0x5d	RT.SetVariable – SetVariable() sets the existing variable with different data at EFI_TPL_APPLICATION.	1. Call SetVariable() service to insert a test variable. 2. Call SetVariable() service to insert the test variable again with the different data in which the left part of old data is the same as the new data The returned status should be EFI_SUCCESS. 3. Call GetVariable() service to get the test variable. The data of the test variable should be changed to the new one.
5.2.1.3.12	0xcefbdb2c, 0x0c7d, 0x4dcf, 0xae, 0x16, 0x32, 0xa8, 0x78, 0xca, 0x2d, 0x3e	RT.SetVariable – SetVariable() sets the existing variable with different data at EFI_TPL_CALLBACK.	1. Call SetVariable() service to insert a test variable. 2. Call SetVariable() service to insert the test variable again with the different data in which the left part of old data is the same as the new data. The returned status should be EFI_SUCCESS. 3. Call GetVariable() service to get the test variable. The data of the test variable should be changed to the new one.
5.2.1.3.13	0xc457149c, 0x75d0, 0x48b5, 0xa1, 0x6c, 0x7e, 0x9f, 0x14, 0x4a, 0xab, 0x15	RT.SetVariable – SetVariable() sets similar existing variables at EFI_TPL_APPLICATION.	1. Call SetVariable() service to insert two similar variables. 2. Call SetVariable() service to insert a test variable. The returned status should be EFI_SUCCESS. 3. Call GetVariable() service to get the test variable. The returned data should be those written before.

Number	GUID	Assertion	Test Description
5.2.1.3.14	0x89f533da, 0x20ee, 0x41f8, 0x8c, 0x60, 0xc3, 0xc4, 0x14, 0x19, 0x05, 0x15	`RT.SetVariable –` `SetVariable()` sets similar existing variables at `EFI_TPL_CALLBACK`.	1. Call `SetVariable()` service to insert two similar variables. 2. Call `SetVariable()` service to insert a test variable. The returned status should be `EFI_SUCCESS`. 3. Call `GetVariable()` service to get the test variable. The returned data should be those written before.
5.2.1.3.15	0xfc5f89d1, 0x4fce, 0x4fe9, 0xa2, 0xfd, 0xa2, 0xfe, 0x69, 0x5b, 0xaa, 0x35	`RT.SetVariable –` `SetVariable()` sets similar existing variables at `EFI_TPL_APPLICATION`.	1. Call `SetVariable()` service to insert a similar variable, whose name is the test variable's name plus character 'A'. 2. Call `SetVariable()` service to insert a test variable. The returned status should be `EFI_SUCCESS`. 3. Call `GetVariable()` service to get the similar variable. The returned data should be unchanged.
5.2.1.3.16	0xfa5f4961, 0xdfaf, 0x425f, 0x95, 0x14, 0x14, 0x52, 0x5c, 0x69, 0xc7, 0x83	`RT.SetVariable –` `SetVariable()` sets similar existing variables at `EFI_TPL_CALLBACK`.	1. Call `SetVariable()` service to insert a similar variable, whose name is the test variable's name + 'A'. 2. Call `SetVariable()` service to insert a test variable. The returned status should be `EFI_SUCCESS`. 3. Call `GetVariable()` service to get the similar variable. The returned data should be unchanged.

Number	GUID	Assertion	Test Description
5.2.1.3.17	0x3cf290ca, 0x49e9, 0x43c0, 0x8a, 0x0c, 0x46, 0xea, 0x17, 0x53, 0x41, 0x08	**RT.SetVariable – SetVariable()** sets similar existing variables at **EFI_TPL_APPLICATION**.	1. Call **SetVariable()** service to insert a similar variable, whose name is the test variable's name minus character 'A'. 2. Call **SetVariable()** service to insert a test variable. The returned status should be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get a similar variable. The returned data should be unchanged.
5.2.1.3.18	0xc1f69f8f, 0xa6ed, 0x4823, 0x88, 0xd9, 0x9a, 0x23, 0x8e, 0x6a, 0x11, 0x00	**RT.SetVariable – SetVariable()** sets similar existing variables at **EFI_TPL_CALLBACK**.	1. Call **SetVariable()** service to insert a similar variable, whose name is the test variable's name minus character 'A'. 2. Call **SetVariable()** service to insert a test variable. The returned status should be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get the similar variable. The returned data should be unchanged.
5.2.1.3.19	0x7b893a77, 0x70ca, 0x48e4, 0xad, 0x1d, 0xe4, 0x31, 0x15, 0xb1, 0xce, 0x5e	**RT.SetVariable – SetVariable()** removes all variables with a ***DataSize*** value of 0 at **EFI_TPL_APPLICATION**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **SetVariable()** service with a ***DataSize*** value of 0 to delete the test variable. The returned status should be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get the test variable. The returned status should be **EFI_NOT_FOUND**.

Number	GUID	Assertion	Test Description
5.2.1.3.20	0x8fcc7182, 0x4f77, 0x4841, 0xbb, 0x81, 0x20, 0xe5, 0x30, 0x5e, 0xa9, 0xda	**RT.SetVariable – SetVariable()** removes all variables with a ***DataSize*** value of 0 at **EFI_TPL_CALLBACK**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **SetVariable()** service with a ***DataSize*** value of 0 to delete the test variable. The returned status should be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get the test variable. The returned status should be **EFI_NOT_FOUND**.
5.2.1.3.21	0x931b363e, 0x8ab4, 0x49db, 0x82, 0x21, 0x2f, 0xdd, 0x9d, 0xa4, 0x36, 0x6c	**RT.SetVariable – SetVariable()** removes all variables with ***Attributes*** values of 0 at **EFI_TPL_APPLICATION**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **SetVariable()** service with ***Attributes*** values of 0 to delete the test variable. The returned status should be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get the test variable. The returned status should be **EFI_NOT_FOUND**.
5.2.1.3.22	0x7eac83e5, 0x0e54, 0x4812, 0x9b, 0xb0, 0x6f, 0xf6, 0xdc, 0x7d, 0xeb, 0x8f	**RT.SetVariable – SetVariable()** removes all variables with ***Attributes*** values of 0 at **EFI_TPL_CALLBACK**.	1. Call **SetVariable()** service to insert a test variable. 2. Call **SetVariable()** service with ***Attributes*** values of 0 to delete the test variable. The returned status should be **EFI_SUCCESS**. 3. Call **GetVariable()** service to get the test variable. The returned status should be **EFI_NOT_FOUND**.

Number	GUID	Assertion	Test Description
5.2.1.3.23	0x6afdea5e, 0x1030, 0x48ab, 0x91, 0xdd, 0x7c, 0xd3, 0x53, 0x7c, 0xad, 0x3b	**RT.SetVariable –** checks Non-volatile variable exists after system reset at **EFI_TPL_APPLICATION**.	1. Call **SetVariable()** service to insert a non-volatile test variable. The returned status must be **EFI_SUCCESS**. 2. Reset the system. 3. Call **GetVariable()** service to get the test variable. The returned status should be **EFI_SUCCESS**, and the returned data should be the same as the original data set.
5.2.1.3.24	0x653f14cc, 0x8ecd, 0x4aaf, 0xad, 0xd6, 0x96, 0xc5, 0x07, 0x11, 0x2d, 0x67	**RT.SetVariable –** checks Non-volatile variable exists after system reset at **EFI_TPL_CALLBACK**.	1. Call **SetVariable()** service to insert a non-volatile test variable. The returned status must be **EFI_SUCCESS**. 2. Reset the system. 3. Call **GetVariable()** service to get the test variable. The returned status should be **EFI_SUCCESS**, and the returned data should be the same as the orginal data set.
5.2.1.3.25	0xb93d2b03, 0x5943, 0x4c7d, 0x98, 0xec, 0xc5, 0xfe, 0x4c, 0x6e, 0x10, 0xc9	**RT.SetVariable –** checks Volatile variable does not exist after system reset at **EFI_TPL_APPLICATION**.	1. Call **SetVariable()** service to insert a volatile test variable. The returned status must be **EFI_SUCCESS**. 2. Reset the system. 3. Call **GetVariable()** service to get the test variable. The returned status should be **EFI_NOT_FOUND**.
5.2.1.3.26	0x9ec88dbe, 0xa0e4, 0x43a2, 0xaa, 0x2b, 0x60, 0xbd, 0xe6, 0xb0, 0x14, 0x1a	**RT.SetVariable –** Volatile variable does not exist after system reset at **EFI_TPL_CALLBACK**.	1. Call **SetVariable()** service to insert a volatile test variable. The returned status must be **EFI_SUCCESS**. 2. Reset the system. 3. Call **GetVariable()** service to get the test variable. The returned status should be **EFI_NOT_FOUND**.

Number	GUID	Assertion	Test Description
5.2.1.3.27	0x98ca8089, 0x7f55, 0x4427, 0x8c, 0x15, 0xaf, 0xa6, 0x3d, 0x78, 0x48, 0xb0	RT.SetVariable - With DataSize is 0	1. Call SetVariable() service to insert a volatile test variable. The returned status must be EFI_SUCCESS. 2. Call SetVariable() service to remove this variable with DataSize being 0. The return status should be EFI_SUCCESS. 3. Call SetVariable() service to remove this variable with DataSize being 0. The returned status should be EFI_NOT_FOUND.
5.2.1.3.28	0x008e18a5, 0xc345, 0x48ae, 0x91, 0x34, 0x61, 0xa6, 0x92, 0xe3, 0xb, 0x87	RT.SetVariable - Must return EFI_SUCCESS when creating one time-based Auth Variable.	Call SetVariable to create a time-based authenticated variable. The expected return status is EFI_SUCCESS.
5.2.1.3.29	0x20678b3e, 0xbcca, 0x4186, 0x84, 0xaf, 0x47, 0x16, 0xe7, 0xaf, 0xde, 0x85	RT.SetVariable - The created time-based Auth Variable should pass the data validation.	Call GetVariable to retrieve the Auth Variable, and validate the Auth Variable data.
5.2.1.3.30	0xaa6bf36f, 0xdae5, 0x43ed, 0x95, 0x4d, 0xc1, 0xc7, 0x97, 0x9d, 0x32, 0xa0	RT.SetVariable - The second Call SetVariable() with the same Data. The return status is EFI_SECURITY_VIOLATION.	The second Call SetVariable() with the same Data. The return status is EFI_SECURITY_VIOLATION.
5.2.1.3.31	0x2bc131ec, 0x0530, 0x4994, 0xbb, 0x81, 0x15, 0x35, 0x5c, 0xef, 0xe5, 0x88	RT.SetVariable - Call SetVariable() with modified/invalid Data. The expected status is EFI_SECURITY_VIOLATION	Call SetVariable() with modified/invalid Data. The expected status is EFI_SECURITY_VIOLATION.
5.2.1.3.32	0x0e49b21e, 0x409c, 0x4502, 0x9e, 0xc6, 0x55, 0xfe, 0x85, 0xf8, 0x54, 0x95	RT.SetVariable - Call SetVariable() with new/valid Data. The expected status is EFI_SUCCESS.	Call SetVariable() with new/valid Data. The expected status is EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.2.1.3.33	0xadabac45, 0x1e0d, 0x40b0, 0x9b, 0xd1, 0x8c, 0x3a, 0xd7, 0xfb, 0x69, 0xd6	RT.SetVariable - The renewed time-based Auth Variable should pass the data validation.	Call GetVariable to retrieve the renewed Auth Variable, and validate the Auth Variable data.
5.2.1.3.34	0x6339807b, 0x0741, 0x45c4, 0x81, 0xa8, 0xe2, 0xde, 0x5a, 0x0b, 0xfb, 0x55	RT.SetVariable – call SetVariable() with the old Data/timestamp. The expected status is EFI_SECURITY_VIOLATION.	Call SetVariable() with the old Data/ timestamp. The expected status is EFI_SECURITY_VIOLATION
5.2.1.3.35	0xa2d53dea, 0x8275, 0x4b9a, 0xbd, 0xa0, 0xac, 0x86, 0xfb, 0x4e, 0x0f, 0x30	RT.SetVariable – call SetVariable() with the Data signed by another key, the expect status should be EFI_SECURITY_VIOLATION	Call SetVariable() with the Data signed by another key, the expect status should be EFI_SECURITY_VIOLATION
5.2.1.3.36	0x28c7f0db, 0x2546, 0x4374, 0x8f, 0xf9, 0x75, 0x80, 0xc4, 0x68, 0x9b, 0x93	RT.SetVariable – call SetVariable() to do the append operation, the expect status should be EFI_SUCCESS	Call SetVariable() to do the append operation, the expect status should be EFI_SUCCESS
5.2.1.3.37	0x1e87dbe9, 0x234b, 0x4c82, 0x8c, 0x86, 0x2f, 0x26, 0xfa, 0xc6, 0x60, 0x2e	RT.SetVariable –The appended time base Auth Variable should pass the data validation	Call GetVariable() to retrieve the appended Auth Variable, and validate the Auth Variable data.
5.2.1.3.38	0x3cc4add2, 0x0ed7, 0x4837, 0xb4, 0x63, 0xbc, 0x46, 0xd1, 0x3b, 0x2f, 0x65	RT.SetVariable – call SetVariable() to do the delete operation. The expected status is EFI_SUCCESS	Call SetVariable() to do the delete operation. The expected status is EFI_SUCCESS
5.2.1.3.39	0xfa50a705, 0x5d95, 0x4cad, 0xb4, 0x6c, 0xa0, 0x12, 0x9b, 0x68, 0x22, 0x8e	RT.SetVariable –The deleted time-based Auth Variable should not be found.	Call GetVariable() to retrieve the deleted Auth Variable. The return status should be EFI_NOT_FOUND.

Number	GUID	Assertion	Test Description
5.2.1.3.40	0x27e8e4de, 0x56ed, 0x4710, 0xa6, 0x3a, 0xc6, 0x35, 0xe3, 0x9d, 0x33, 0x64	**RT.SetVariable -** must return **EFI_SUCCESS** when creating a time-based Auth Variable with one different key.	Call **SetVariable()** to create a time-based Auth Variable with one different key. The expected return status is **EFI_SUCCESS**.
5.2.1.3.41	0xba99e7f8, 0x8018, 0x46a2, 0xb2, 0xe5, 0x8b, 0xde, 0x42, 0xc1, 0xe6, 0xd5	**RT.SetVariable –** call **SetVariable()** to do the append operation with the new data. The expected status is **EFI_SUCCESS**	Call **SetVariable()** to do the append operation with the new data. The expected status is **EFI_SUCCESS**
5.2.1.3.42	0xc764906d, 0x73bb, 0x44b7, 0xae, 0x40, 0x0c, 0x51, 0xde, 0xc3, 0xc7, 0x51	**RT.SetVariable –** call **SetVariable()** to set the **Data** with one old timestamp. The return status should be **EFI_SECURITY_VIOLATION**	Call **SetVariable()** to set the **Data** with one old timestamp. The return status should be **EFI_SECURITY_VIOLATION**
5.2.1.3.43	0x1a28fa01, 0x135c, 0x4aeb, 0xa1, 0xb4, 0x68, 0x6a, 0x0b, 0x53, 0xb2, 0x9	**RT.SetVariable –** call **SetVariable()** to do the delete operation. The expected status is **EFI_SUCCESS**	Call **SetVariable()** to do the delete operation. The expected status is **EFI_SUCCESS**
5.2.1.3.44	0xe9893bcb, 0xef2b, 0x495c, 0x82, 0xf0, 0xd0, 0x63, 0x0d, 0xa7, 0x94, 0x76	**RT.SetVariable –** must return **EFI_SECURITY_VIOLATION**	Call **SetVariable()** to enroll an invalid time-based authenticated variable but several bits changed.
5.2.1.3.45	0x2534abc0, 0x1f01, 0x48a0, 0x96, 0xde, 0xf8, 0xbb, 0xa7, 0x45, 0xc3, 0x64	**RT.SetVariable –** must return **EFI_SECURITY_VIOLATION**	Call **SetVariable()** to enroll a time-based authenticated variable with an invalid attribute.
5.2.1.3.46	0x896f8325, 0xed28, 0x4af5, 0x96, 0xba, 0x3b, 0xe3, 0xf2, 0x97, 0x74, 0x8b	RT.SetVariable – **SetVariable()** returns EFI_INVALID_PARAMETER When it wants to change the attribute of one existed variable .	1. Call **SetVariable()** service to modify the attribute of one existed variable. The returned status should be **EFI_INVALID_PARAMETER**.

4.1.4 QueryVariableInfo()

Number	GUID	Assertion	Test Description
5.2.1.4.1	0xad9578bf, 0x7a02, 0x4ef0, 0x8f, 0xe8, 0xd9, 0x45, 0x91, 0xa1, 0xe9, 0x31	**RT.QueryVariableInfo –** Query variable info with a ***MaximumVariableStorageSize*** value of **NULL**.	1. Call **QueryVariableInfo** service with a ***MaximumVariableStorageSize*** value of **NULL**. The returned code must be **EFI_INVALID_PARAMETER**.
5.2.1.4.2	0x5d13a732, 0x60ea, 0x42d5, 0xa0, 0x01, 0x43, 0x63, 0xd9, 0xb1, 0x8b, 0xf4	**RT.QueryVariableInfo –** Query variable info with a ***RemainingVariableStorageSize*** value of **NULL**.	1. Call **QueryVariableInfo** service with a ***RemainingVariableStorageSize*** value of **NULL**. The returned code must be **EFI_INVALID_PARAMETER**.
5.2.1.4.3	0xd3247b73, 0x5eb9, 0x4594, 0x8a, 0xb3, 0x27, 0xd9, 0x38, 0x4f, 0x3f, 0x13	**RT.QueryVariableInfo –** Query variable info with ***MaximumVariableSize*** value of **NULL**.	1. Call **QueryVariableInfo** service with a ***MaximumVariableSize*** value of **NULL**. The returned code must be **EFI_INVALID_PARAMETER**.
5.2.1.4.4	0xe7f2eb9f, 0x1624, 0x45a9, 0xa2, 0x87, 0x3e, 0xa6, 0xf2, 0xf7, 0x4c, 0x5f	**RT.QueryVariableInfo –** Query variable info when ***Attributes*** is not a combination of **EFI_VARIABLE_RUNTIME_ACCESS** , **EFI_VARIABLE_BOOTSERVICE_ACCESS** and **EFI_VARIABLE_NON_VOLATILE.**	1. Call **QueryVariableInfo** service with ***Attributes*** values of 0. The returned code must be **EFI_UNSUPPORTED**.
5.2.1.4.5	0x2f9966ba, 0x0091, 0x4085, 0xbf, 0x9d, 0x09, 0xaa, 0x80, 0x9f, 0x94, 0x2e	**RT.QueryVariableInfo –** Query variable info with an invalid combination of ***Attributes***.	1. Call **QueryVariableInfo** service with the ***Attributes***: **EFI_VARIABLE_NON_VOLATILE** **EFI_VARIABLE_RUNTIME_ACCESS** **EFI_VARIABLE_NON_VOLATILE|EFI_VARIABLE_RUNTIME_ACCESS** The returned code must be **EFI_INVALID_PARAMETER.**

Number	GUID	Assertion	Test Description
5.2.1.4.8	0xad6e6a8f, 0x3a05, 0x4183,0xb6, 0x90, 0x40, 0xa8, 0x91, 0xd8, 0x62, 0xae	**RT.QueryVariableInfo –** Query variable info with a valid ***Attributes*** in Run time.	For each TPL less than or equal to **TPL_CALLBACK** and each ***Attributes*** of **BA, NV|BA, BA|RA** and **NV|BA|RA** do: 1. Call **QueryVariableInfo** with the ***Attributes*** selected. Check.(Number1) 2. Call **SetVariable** service to insert a variable. Check. 3. Call **QueryVariableInfo** with the ***Attributes*** selected. Check. (Number2) 4. Call **SetVariable** service to delete the variable inserted. Check. 5. Call **QueryVariableInfo** service with the ***Attributes*** selected. Check.(Number3) For Number1, Number2, Number3, the following items need to be checked: 1. returned codes must be **EFI_SUCCESS**. 2. returned ***MaximumVariableStorageSize** must be the same. 3. returned ***MaximumVariableSize** must be the same, and they all are equal to **MAX_VARIABLE_SIZE**. 4. Number2 returned ***RemainingVariableStorageSize** must be the value of Number1 minus the size of the variable inserted in step 2. 5. Number3 returned ***RemainingVariableStorageSize** must be the value of Number1

4.1.5 HardwareErrorRecord

Number	GUID	Assertion	Test Description
5.2.1.5.1	0xc8126edc, 0x7197, 0x4113, 0xb7, 0xb6, 0xd5, 0x3d, 0x53, 0xe6, 0x72, 0xea	`HWErrRecTest – Func Test`	1. Call `GetVariable()` to check the `HardwareErrorRecord` support of platform. 2. Call `QueryVariableInfo()` to detect the storage size. 3. Get a useable `HWErrRec` variable name and call `SetVariable()` to set it with data. 4. Reset system, call `GetVariable()` to get the data. 5. Compare the data, they should be same.
5.2.1.5.2	0xd8bd5c0a, 0x192f, 0x4501, 0xbc, 0x58, 0x89, 0xd3, 0x18, 0x60, 0x24, 0x5e	`HWErrRecTest – Conf Test` HardwareErrorRecord with invalid attributes.	1. Call `GetVariable()` to check the `HardwareErrorRecord` support of platform. 2. Call `QueryVariableInfo()` to detect the storage size. 3. Get a useable `HWErrRec` variable name and call `SetVariable()` to set it with invalid attributes. The returned code must be `EFI_INVALID_PARAMETER`.
5.2.1.5.3	0xe1259932, 0xf39c, 0x465b, 0xb4, 0xe3, 0xa1, 0xb2, 0x77, 0x8b, 0xa1, 0x04	`HWErrRecTest – Conf Test` HardwareErrorRecord with twice deletion.	1. Call `GetVariable()` to check the `HardwareErrorRecord` support of platform. 2. Call `QueryVariableInfo()` to detect the storage size. 3. Get a useable `HWErrRec` variable name and call `SetVariable()` to set it. 4. Delete the variable twice. The first time, the returned code must be `EFI_SUCCESS;` the second time, the returned code must be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.2.1.5.4	0xf5b942c9, 0x1f0c, 0x4c45, 0x85, 0x72, 0xc4, 0x53, 0x79, 0x51, 0x50, 0xdf	**HWErrRecTest – Conf Test** Retrive the Hardware Error Record variables, check the name of them.	1. Call **GetVariable()** to check the **HardwareErrorRecord** support of platform. 2. Call **QueryVariableInfo()** to detect the storage size. 3. Retrive the Hardware Error Record variables, check the name of them

4.2 Time Services Test

Reference Document:

UEFI Specification, Time Services Section

Table 7. Time Services Functions

Name	Type	Description
GetTime	Runtime	Returns the current time and date, and the time-keeping capabilities of the platform.
SetTime	Runtime	Sets the current local time and date information.
GetWakeupTime	Runtime	Returns the current wakeup alarm clock setting.
SetWakeupTime	Runtime	Sets the system wakeup alarm clock time.

4.2.1 GetTime()

Number	GUID	Assertion	Test Description
5.2.2.1.1	0x105de1dc, 0x32b2, 0x4d85, 0x9b, 0x30, 0xd4, 0x41, 0x80, 0x0f, 0xdc, 0x4c	**RT.GetTime – GetTime()** returns **EFI_INVALID_PARAMETER** with ***Time*** is **NULL**.	1. Call **GetTime()** with ***Time*** is **NULL**. The return code must be **EFI_INVALID_PARAMETER**.
5.2.2.1.2	0x51437f55, 0x25e1, 0x43eb, 0xae, 0x76, 0x0d, 0x32, 0x1c, 0x12, 0xf6, 0x38	**RT.GetTime – GetTime()** gets the system time at **EFI_TPL_APPLICATION**.	1. Call **GetTime()** with valid parameters. The return code must be **EFI_SUCCESS**.
5.2.2.1.3	0x1a6e41f0, 0x361e, 0x4c46, 0xa2, 0xc4, 0x35, 0x42, 0xb3, 0x6f, 0xa5, 0xb6	**RT.GetTime – GetTime()** gets the system time at **EFI_TPL_CALLBACK**.	1. Call **GetTime()** with valid parameters. The return code must be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.2.2.1.4	0x3568b497, 0x6524, 0x4415, 0xac, 0xaa, 0xa8, 0xee, 0x24, 0x83, 0x9b, 0xdd	**RT.GetTime –** **GetTime()** gets the system time at **EFI_TPL_APPLICATION**.	1. Call **GetTime()** with valid parameters. The return time should be valid.
5.2.2.1.5	0xa2c13016, 0x01d4, 0x4ea7, 0xb0, 0x8e, 0xb7, 0x74, 0x22, 0x4d, 0x7e, 0xa5	**RT.GetTime –** **GetTime()** gets the system time at **EFI_TPL_CALLBACK**.	1. Call **GetTime()** with valid parameters. The return time should be valid.
5.2.2.1.6	0x2cd14974, 0x4937, 0x4817, 0x91, 0xb0, 0x82, 0x2f, 0x40, 0xca, 0x22, 0xbc	**RT.GetTime –** **GetTime()** gets the system time with ***Capabilities*** is **NULL** at **EFI_TPL_APPLICATION**.	1. Call **GetTime()** with a ***Capabilities*** value of **NULL**. The return code should be **EFI_SUCCESS**.
5.2.2.1.7	0x9bbabc14, 0xced2, 0x48fc, 0xbb, 0x9e, 0x79, 0x37, 0x49, 0xe8, 0x1f, 0xe2	**RT.GetTime –** **GetTime()** gets the system time with ***Capabilities*** is **NULL** at **EFI_TPL_CALLBACK**.	1. Call **GetTime()** with a ***Capabilities*** value of **NULL**. The return code should be **EFI_SUCCESS**.
5.2.2.1.8	0x938366e9, 0x3311, 0x4007, 0x87, 0xc3, 0xa2, 0x18, 0x7f, 0x05, 0x14, 0xe3	**RT.GetTime –** **GetTime()** gets the system time with ***Capabilities*** is **NULL** at **EFI_TPL_APPLICATION**.	1. Call **GetTime()** with a ***Capabilities*** value of **NULL**. The return time should be valid.
5.2.2.1.9	0x565f4b15, 0xb132, 0x4c74, 0x97, 0xc2, 0xf3, 0xa6, 0xf5, 0xbf, 0xd2, 0x21	**RT.GetTime –** **GetTime()** gets the system time with ***Capabilities*** is **NULL** at **EFI_TPL_CALLBACK**.	1. Call **GetTime()** with a ***Capabilities*** value of **NULL**. The return time should be valid.

4.2.2 SetTime()

Number	GUID	Assertion	Test Description
5.2.2.2.1	0x6f96cde3, 0x6067, 0x4213, 0x81, 0xf8, 0x45, 0x90, 0x1d, 0x92, 0x1a, 0x12	**RT.SetTime –** **SetTime()** returns **EFI_INVALID_PARAMETER** with ***Year*** is less than the low range.	1. Call **SetTime()** with ***Time.Year*** is 1899. The return code must be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.2.2.2.2	0x8ce9f594, 0x2d49, 0x4436, 0xb1, 0xd1, 0xe4, 0xd4, 0xbf, 0x55, 0x41, 0xdc	**RT.SetTime – SetTime()** returns **EFI_INVALID_PARAMETER** with ***Year*** is greater than the upper range.	1. Call **SetTime()** with ***Time.Year*** is 10000. The return code must be **EFI_INVALID_PARAMETER**.
5.2.2.2.3	0x972fadc8, 0x5cc4, 0x4cbe, 0xbe, 0xd6, 0x76, 0xca, 0xef, 0x2d, 0x1b, 0x1a	**RT.SetTime – SetTime()** returns **EFI_INVALID_PARAMETER** with ***Year*** is invalid.	1. Call **SetTime()** with ***Time.Year*** is -1. The return code must be **EFI_INVALID_PARAMETER**.
5.2.2.2.4	0xcaac8a85, 0x26c2, 0x43e7, 0x83, 0x40, 0x5a, 0x78, 0x85, 0x43, 0xef, 0x81	**RT.SetTime – SetTime()** returns **EFI_INVALID_PARAMETER** with ***Month*** is less than the low range.	1. Call **SetTime()** with ***Time.Month*** is 0. The return code must be **EFI_INVALID_PARAMETER**.
5.2.2.2.5	0x12470ee0, 0x19e1, 0x49ff, 0xbc, 0x1e, 0x8e, 0xb3, 0x6f, 0xab, 0xf0, 0xfc	**RT.SetTime – SetTime()** returns **EFI_INVALID_PARAMETER** with ***Month*** is greater than the upper range.	1. Call **SetTime()** with ***Time.Month*** is 13. The return code must be **EFI_INVALID_PARAMETER**.
5.2.2.2.6	0xae7293c9, 0x0cbd, 0x4317, 0xb6, 0xeb, 0x33, 0xe1, 0x83, 0x46, 0x8d, 0x9e	**RT.SetTime – SetTime()** returns **EFI_INVALID_PARAMETER** with ***Month*** is invalid.	1. Call **SetTime()** with ***Time.Month*** is -1. The return code must be **EFI_INVALID_PARAMETER**.
5.2.2.2.7	0xb8048c3c, 0xbf1f, 0x477d, 0xb7, 0x17, 0x55, 0x41, 0xfc, 0xa7, 0xb5, 0x61	**RT.SetTime – SetTime()** returns **EFI_INVALID_PARAMETER** with ***Day*** is less than the low range.	1. Call **SetTime()** with ***Time.Day*** is 0. The return code must be **EFI_INVALID_PARAMETER**.
5.2.2.2.8	0x0d2c6265, 0xad3a, 0x4554, 0xb0, 0x16, 0x6c, 0xb7, 0xff, 0x59, 0x1f, 0x78	**RT.SetTime – SetTime()** returns **EFI_INVALID_PARAMETER** with ***Day*** is greater than the upper range.	1. Call **SetTime()** with ***Time.Day*** is 32. The return code must be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.2.2.2.9	0x0467b0c4, 0xdf8c, 0x4bfc, 0xa8, 0x4b, 0xef, 0xa6, 0x90, 0x5b, 0xde, 0xd9	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***Day*** is invalid.	1. Call `SetTime()` with ***Time.Day*** is -1. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.10	0x1e433b44, 0xa599, 0x4dcd, 0x9c, 0x38, 0xe7, 0xc0, 0x97, 0xf2, 0x56, 0x4b	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***Day*** is greater than the upper range.	1. Call `SetTime()` with ***Time.Month*** is 4 and ***Time.Day*** is 31. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.11	0xc9bfb088, 0x07ba, 0x413c, 0xa4, 0x72, 0xbd, 0x17, 0x92, 0xdd, 0xc6, 0xec	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***Hour*** is greater than the upper range.	1. Call `SetTime()` with ***Time.Hour*** is 24. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.12	0xd7b3ca07, 0xa484, 0x4604, 0x83, 0x37, 0x6f, 0x13, 0x4f, 0x88, 0xb3, 0x5a	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***Hour*** is invalid.	1. Call `SetTime()` with ***Time.Hour*** is -1. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.13	0xc645baaa, 0x3eb6, 0x4577, 0x97, 0x5d, 0x21, 0x05, 0x04, 0x83, 0x64, 0x2b	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***Minute*** is greater than the upper range.	1. Call `SetTime()` with ***Time.Minute*** is 60. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.14	0xa42f7c8e, 0xfa7a, 0x4026, 0xb9, 0x6b, 0x66, 0xe3, 0xf2, 0xe9, 0x93, 0x55	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***Minute*** is invalid.	1. Call `SetTime()` with ***Time.Minute*** is -1. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.15	0xd37d5f03, 0x6dbb, 0x4724, 0x9e, 0xc1, 0xed, 0x13, 0x6b, 0x17, 0x22, 0xe9	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***Second*** is greater than the upper range.	1. Call `SetTime()` with ***Time.Second*** is 60. The return code must be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.2.2.2.16	0xcd47c7aa, 0x6522, 0x45ed, 0xa7, 0xb4, 0x29, 0x6d, 0x57, 0x43, 0xc7, 0x78	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***Second*** is invalid.	1. Call `SetTime()` with ***Time.Second*** is -1. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.17	0x14bccf9f, 0xda75, 0x46db, 0xb1, 0xfc, 0x7e, 0x67, 0x3b, 0x37, 0x25, 0x6e	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***Nanosecond*** is greater than the upper range.	1. Call `SetTime()` with ***Time.Nanosecond*** is 1000000000. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.18	0x966cf8d6, 0xf952, 0x4770, 0xa1, 0x9e, 0xf8, 0x78, 0xbc, 0x60, 0xbc, 0xeb	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***Nanosecond*** is invalid.	1. Call `SetTime()` with ***Time.Nanosecond*** is -1. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.19	0x59a9febb, 0xf6d1, 0x4b13, 0xae, 0xcd, 0xf3, 0x65, 0xc2, 0x11, 0xa4, 0xed	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***TimeZone*** is less than the low range.	1. Call `SetTime()` with ***Time.TimeZone*** is -1441. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.20	0x5786f2c1, 0x48a7, 0x4856, 0x89, 0xe7, 0xba, 0xce, 0xc0, 0x85, 0xf3, 0xf9	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with ***TimeZone*** is greater than the upper range.	1. Call `SetTime()` with ***Time.TimeZone*** is 1441. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.21	0xd3a1cbdd, 0x1df5, 0x4d24, 0x97, 0x53, 0xc3, 0xae, 0xa2, 0x7a, 0xab, 0x46	`RT.SetTime` – `SetTime()` returns `EFI_INVALID_PARAMETER` with invalid leap day.	1. Call `SetTime()` with ***Time*** is 2001/2/29. The return code must be `EFI_INVALID_PARAMETER`.
5.2.2.2.22	0x29151ae4, 0x7a5e, 0x42d9, 0x84, 0xf8, 0xe9, 0xc5, 0x67, 0x87, 0xb7, 0xe8	`RT.SetTime` – `SetTime()` returns `EFI_SUCCESS` to update the ***Year*** at `EFI_TPL_APPLICATION`.	1. Call `SetTime()` to update the ***Time.Year***. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.2.2.2.23	0x75e988ee, 0xec78, 0x4190, 0x9a, 0x09, 0xb1, 0x31, 0x5c, 0x20, 0x25, 0xa5	**RT.SetTime – SetTime()** returns **EFI_SUCCESS** to update the ***Year*** at **EFI_TPL_CALLBACK**.	1. Call **SetTime()** to update the ***Time.Year***. The return code should be **EFI_SUCCESS**.
5.2.2.2.24	0x3b96a20c, 0x2b1f, 0x44ea, 0xba, 0xa9, 0xf9, 0x6f, 0xee, 0x13, 0x1d, 0x05	**RT.SetTime – SetTime()** returns **EFI_SUCCESS** to update the ***Year*** at **EFI_TPL_APPLICATION**.	1. Call **SetTime()** to update the ***Time.Year***. 2. Call **GetTime()** to verify the updated ***Year***. The return ***Time*** should be set before.
5.2.2.2.25	0xe664e1d7, 0xb733, 0x410d, 0xbc, 0x53, 0xd4, 0xcf, 0xf2, 0x46, 0x43, 0x55	**RT.SetTime – SetTime()** returns **EFI_SUCCESS** to update the ***Year*** at **EFI_TPL_CALLBACK**.	1. Call **SetTime()** to update the ***Time.Year***. 2. Call **GetTime()** to verify the updated ***Year***. The return ***Time*** should be set before.
5.2.2.2.26	0x4e123824, 0x8636, 0x4426, 0x81, 0xe6, 0x16, 0x75, 0x62, 0x8c, 0xde, 0x69	**RT.SetTime – SetTime()** returns **EFI_SUCCESS** to update the ***Month*** at **EFI_TPL_APPLICATION**.	1. Call **SetTime()** to update the ***Time.Month***. The return code should be **EFI_SUCCESS**.
5.2.2.2.27	0x8f0bfe23, 0xb6ec, 0x4ea2, 0x8e, 0x03, 0x0a, 0x7a, 0x5e, 0x36, 0x45, 0xb3	**RT.SetTime – SetTime()** returns **EFI_SUCCESS** to update the ***Month*** at **EFI_TPL_CALLBACK**.	1. Call **SetTime()** to update the ***Time.Month***. The return code should be **EFI_SUCCESS**.
5.2.2.2.28	0x2d5cdbe5, 0x1055, 0x4ef6, 0x8e, 0x90, 0x0c, 0x99, 0x3f, 0x93, 0xf6, 0x98	**RT.SetTime – SetTime()** returns **EFI_SUCCESS** to update the ***Month*** at **EFI_TPL_APPLICATION**.	1. Call **SetTime()** to update the ***Time.Month***. 2. Call **GetTime()** to verify the updated ***Month***. The return ***Time*** should be set before.
5.2.2.2.29	0xda4b19e7, 0xf605, 0x4fb9, 0xa1, 0x81, 0xcc, 0xd3, 0x35, 0x29, 0x0b, 0xfe	**RT.SetTime – SetTime()** returns **EFI_SUCCESS** to update the ***Month*** at **EFI_TPL_CALLBACK**.	1. Call **SetTime()** to update the ***Time.Month***. 2. Call **GetTime()** to verify the updated Month. The return Time should be set before.

Number	GUID	Assertion	Test Description
5.2.2.2.30	0x7af90ce7, 0x1fed, 0x4101, 0x82, 0xdc, 0xcc, 0x63, 0x4c, 0xdf, 0x20, 0x4e	`RT.SetTime –` `SetTime()` returns `EFI_SUCCESS` to update the daylight at `EFI_TPL_APPLICATION`.	1. Call `SetTime()` to update the *`Time.Daylight`*. The return code should be `EFI_SUCCESS`.
5.2.2.2.31	0xfa81d174, 0x5743, 0x485f, 0xb2, 0x48, 0xaa, 0xea, 0xdd, 0x7c, 0x1e, 0x51	`RT.SetTime –` `SetTime()` returns `EFI_SUCCESS` to update the daylight at `EFI_TPL_CALLBACK`.	1. Call `SetTime()` to update the *`Time.Daylight`*. The return code should be `EFI_SUCCESS`.
5.2.2.2.32	0xb39bc904, 0x55e7, 0x4b9b, 0xb4, 0xd8, 0x27, 0x4a, 0xdd, 0x71, 0xd6, 0x25	`RT.SetTime –` `SetTime()` returns `EFI_SUCCESS` to update the daylight at `EFI_TPL_APPLICATION`.	1. Call `SetTime()` to update the *`Time.Daylight`*. 2. Call `GetTime()` to verify the updated *`Daylight`*. The return *`Time`* should be set before.
5.2.2.2.33	0x54daf29b, 0x48e6, 0x4fa4, 0xad, 0x00, 0xb8, 0xd6, 0x48, 0xaf, 0x7d, 0x88	`RT.SetTime –` `SetTime()` returns `EFI_SUCCESS` to update the daylight at `EFI_TPL_CALLBACK`.	1. Call `SetTime()` to update the *`Time.Daylight`*. 2. Call `GetTime()` to verify the updated *`Daylight`*. The return *`Time`* should be set before.
5.2.2.2.34	0xcdbbda04, 0x4f7c, 0x4ba5, 0x8b, 0xcf, 0xc0, 0x50, 0xe5, 0xa9, 0x76, 0xc7	`RT.SetTime –` `SetTime()` returns `EFI_SUCCESS` to update the *`TimeZone`* at `EFI_TPL_APPLICATION`.	1. Call `SetTime()` to update the *`Time.TimeZone`*. The return code should be `EFI_SUCCESS`.
5.2.2.2.35	0xf749b4f1, 0x537d, 0x4ddf, 0x85, 0x45, 0xc0, 0xa4, 0x19, 0x93, 0xce, 0xe4	`RT.SetTime –` `SetTime()` returns `EFI_SUCCESS` to update the *`TimeZone`* at `EFI_TPL_CALLBACK`.	1. Call `SetTime()` to update the *`Time.TimeZone`*. The return code should be `EFI_SUCCESS`.
5.2.2.2.36	0xea99dec5, 0xb879, 0x4c8d, 0xbf, 0xd1, 0xf6, 0x3f, 0xe7, 0x58, 0x99, 0xbf	`RT.SetTime –` `SetTime()` returns `EFI_SUCCESS` to update the *`TimeZone`* at `EFI_TPL_APPLICATION`.	1. Call `SetTime()` to update the *`Time.TimeZone`*. 2. Call `GetTime()` to verify the updated *`TimeZone`*. The return *`Time`* should be set before.

Number	GUID	Assertion	Test Description
5.2.2.2.37	0xd9c645b9, 0x52de, 0x415c, 0xab, 0xdc, 0x72, 0x26, 0xce, 0x6a, 0x30, 0xb1	**RT.SetTime –** SetTime() returns EFI_SUCCESS to update the *TimeZone* at EFI_TPL_CALLBACK.	1. Call SetTime() to update the *Time.TimeZone*. 2. Call GetTime() to verify the updated *TimeZone*. The return *Time* should be set before.

4.2.3 GetWakeupTime()

Number	GUID	Assertion	Test Description
5.2.2.3.1	0xbb9fd931, 0xd3c0, 0x43cd, 0xb0, 0xa7, 0xfe, 0x17, 0xdc, 0xd7, 0x4d, 0x53	**RT.GetWakeupTime –** GetWakeupTime() returns EFI_INVALID_PARAMETER with *Enabled* is NULL.	1. Call GetWakeupTime() with *Enabled* is NULL. The return code must be EFI_UNSUPPORTED or EFI_INVALID_PARAMETER.
5.2.2.3.2	0x200b6e00, 0x9e1b, 0x4891, 0x83, 0x01, 0xef, 0x46, 0x9f, 0x31, 0x17, 0x08	**RT.GetWakeupTime –** GetWakeupTime() returns EFI_INVALID_PARAMETER with *Pending* is NULL.	1. Call GetWakeupTime() with *Pending* is NULL. The return code must be EFI_UNSUPPORTED or EFI_INVALID_PARAMETER.
5.2.2.3.3	0x209435c5, 0xfa4f, 0x405d, 0x80, 0xa6, 0x9e, 0xdc, 0x9d, 0x38, 0x8c, 0xc6	**RT.GetWakeupTime –** GetWakeupTime() returns EFI_INVALID_PARAMETER with *Time* is NULL.	1. Call GetWakeupTime() with *Time* is NULL. The return code must be EFI_UNSUPPORTED or EFI_INVALID_PARAMETER.
5.2.2.3.4	0xe553c375, 0xd529, 0x4610, 0xad, 0xb5, 0x3a, 0x56, 0xc3, 0xec, 0xcb, 0xe9	**RT.GetWakeupTime –** GetWakeupTime() returns EFI_SUCCESS at EFI_TPL_APPLICATION.	1. Call GetWakeupTime() with valid parameters. The return code must be EFI_UNSUPPORTED or EFI_SUCCESS.
5.2.2.3.5	0x36414d2a, 0xf932, 0x43ca, 0xab, 0x08, 0x41, 0x8e, 0x59, 0xd9, 0xa4, 0xa2	**RT.GetWakeupTime –** GetWakeupTime() returns EFI_SUCCESS at EFI_TPL_CALLBACK.	1. Call GetWakeupTime() with valid parameters. The return code must be EFI_UNSUPPORTED or EFI_SUCCESS.
5.2.2.3.6	0x6092de6c, 0x062f, 0x4adb, 0xab, 0x4b, 0xb4, 0xda, 0x69, 0xd2, 0x8e, 0xd8	**RT.GetWakeupTime –** GetWakeupTime() gets the wakeup status at EFI_TPL_APPLICATION.	1. Call GetWakeupTime() with valid parameters. If the *Enabled* is TRUE, the return time should be valid.

Number	GUID	Assertion	Test Description
5.2.2.3.7	0x8061bae9, 0x341c, 0x48ab, 0xad, 0x37, 0x15, 0x5c, 0x6b, 0x0f, 0x13, 0x34	**RT.GetWakeupTime –** `GetWakeupTime()` gets the wakeup status at `EFI_TPL_CALLBACK`.	1. Call `GetWakeupTime()` with valid parameters. If the *`Enabled`* is `TRUE`, the return time should be valid.

4.2.4 SetWakeupTime()

Number	GUID	Assertion	Test Description
5.2.2.4.1	0x41d27daf, 0xe088, 0x441c, 0xb2, 0x05, 0x6d, 0xd7, 0xa4, 0xac, 0x08, 0xb1	**RT.SetWakeupTime –** `SetWakeupTime()` returns `EFI_INVALID_PARAMETER` with *`Year`* is less than the low range.	1. Call `SetWakeupTime()` with *`Time.Year`* is 1997. The return code must be `EFI_UNSUPPORTED` or `EFI_INVALID_PARAMETER`.
5.2.2.4.2	0xe2dbc697, 0xc56a, 0x4c58, 0xa2, 0x74, 0x58, 0x99, 0x94, 0x1c, 0x7e, 0x02	**RT.SetWakeupTime –** `SetWakeupTime()` returns `EFI_INVALID_PARAMETER` with *`Year`* is greater than the upper range.	1. Call `SetWakeupTime()` with *`Time.Year`* is 2100. The return code must be `EFI_UNSUPPORTED` or `EFI_INVALID_PARAMETER`.
5.2.2.4.3	0x2ef795b9, 0xdfac, 0x4334, 0xa2, 0x43, 0x55, 0xbe, 0x0d, 0x0c, 0x3b, 0x44	**RT.SetWakeupTime –** `SetWakeupTime()` returns `EFI_INVALID_PARAMETER` with *`Year`* is invalid.	1. Call `SetWakeupTime()` with *`Time.Year`* is -1. The return code must be `EFI_UNSUPPORTED` or `EFI_INVALID_PARAMETER`.
5.2.2.4.4	0x8f7fe2f6, 0xd96d, 0x4765, 0x96, 0x42, 0x05, 0xae, 0x30, 0x66, 0xd8, 0xb9	**RT.SetWakeupTime –** `SetWakeupTime()` returns `EFI_INVALID_PARAMETER` with *`Month`* is less than the low range.	1. Call `SetWakeupTime()` with *`Time.Month`* is 0. The return code must be `EFI_UNSUPPORTED` or `EFI_INVALID_PARAMETER`.
5.2.2.4.5	0xc398668f, 0x03c2, 0x4cac, 0x81, 0x18, 0x7c, 0xbe, 0xab, 0xd1, 0xb9, 0x67	**RT.SetWakeupTime –** `SetWakeupTime()` returns `EFI_INVALID_PARAMETER` with *`Month`* is greater than the upper range.	1. Call `SetWakeupTime()` with *`Time.Month`* is 13. The return code must be `EFI_UNSUPPORTED` or `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.2.2.4.6	0x57a4eedd, 0xafa6, 0x4233, 0xb2, 0xeb, 0x79, 0xe4, 0x5e, 0x3d, 0xc0, 0x2d	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Month*** is invalid.	1. Call **SetWakeupTime()** with ***Time.Month*** is -1. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.7	0x61dd2e73, 0x0c29, 0x436a, 0x80, 0x73, 0x3c, 0xe4, 0xde, 0xc7, 0x0d, 0xf2	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Day*** is less than the low range.	1. Call **SetWakeupTime()** with ***Time.Day*** is 0. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.8	0x7c532de7, 0x3d59, 0x4a43, 0x9c, 0xf1, 0x8c, 0x35, 0x51, 0x70, 0xbc, 0x86	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Day*** is greater than the upper range.	1. Call **SetWakeupTime()** with ***Time.Day*** is 32. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.9	0xb07ea402, 0x8403, 0x4c42, 0xa4, 0x11, 0x23, 0x2c, 0x37, 0xf9, 0xc5, 0x27	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Day*** is invalid.	1. Call **SetWakeupTime()** with ***Time.Day*** is -1. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.10	0xc86e5f11, 0x2e97, 0x4cee, 0x9c, 0xc8, 0xd3, 0xf5, 0x7f, 0xa6, 0x46, 0x75	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Day*** is greater than the upper range.	1. Call **SetWakeupTime()** with **Time.Month** is 4 and **Time.Day** is 31. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.11	0x0ef3f79c, 0x9399, 0x47f8, 0xab, 0x3b, 0xa6, 0x6c, 0x2f, 0x78, 0x1f, 0x9e	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Hour*** is greater than the upper range.	1. Call **SetWakeupTime()** with ***Time.Hour*** is 24. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.12	0x9f61f3ac, 0x059b, 0x4658, 0x98, 0x2d, 0x61, 0x6e, 0xab, 0x25, 0xcb, 0x6d	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Hour*** is invalid.	1. Call **SetWakeupTime()** with ***Time.Hour*** is -1. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.13	0xa05b10e8, 0x098e, 0x4c02, 0xad, 0x30, 0xef, 0xac, 0x58, 0xf4, 0x07, 0x56	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Minute*** is greater than the upper range.	1. Call **SetWakeupTime()** with ***Time.Minute*** is 60. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.2.2.4.14	0xbca1c0cf, 0xe121, 0x42fc, 0xba, 0x49, 0x2b, 0xd0, 0xad, 0x74, 0x3d, 0x60	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Minute*** is invalid.	1. Call **SetWakeupTime()** with ***Time.Minute*** is -1. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.15	0x89c7e1f1, 0x98cb, 0x4f3c, 0x96, 0xc7, 0x03, 0x59, 0x22, 0xd0, 0xce, 0x34	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Second*** is greater than the upper range.	1. Call **SetWakeupTime()** with ***Time.Second*** is 60. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.16	0x59b0d53d, 0xffac, 0x4c1a, 0xb9, 0xb0, 0x2c, 0xe6, 0xfc, 0x93, 0x8f, 0x0e	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Second*** is invalid.	1. Call **SetWakeupTime()** with ***Time.Second*** is -1. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.17	0x98737393, 0x45af, 0x4945, 0xa7, 0xd2, 0xe2, 0x92, 0xfd, 0x4e, 0x8d, 0x20	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Nanosecond*** is greater than the upper range.	1. Call **SetWakeupTime()** with ***Time.Nanosecond*** is 1000000000. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.18	0xc9eff904, 0x5d44, 0x451c, 0x94, 0xd2, 0x66, 0x73, 0xe1, 0x8e, 0x65, 0x05	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***Nanosecond*** is invalid.	1. Call **SetWakeupTime()** with ***Time.Nanosecond*** is -1. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.19	0x4cf4b039, 0xf2aa, 0x4f8a, 0x9c, 0xec, 0x0a, 0x80, 0x2c, 0xea, 0xd7, 0x5f	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***TimeZone*** is less than the low range.	1. Call **SetWakeupTime()** with ***Time.TimeZone*** is -1441. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.20	0xabd093eb, 0x7d84, 0x4ebc, 0xb3, 0x24, 0xc2, 0x85, 0x79, 0x5b, 0xde, 0x34	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with ***TimeZone*** is greater than the upper range.	1. Call **SetWakeupTime()** with ***Time.TimeZone*** is 1441. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.
5.2.2.4.21	0x0fce1f4c, 0x41f6, 0x4de4, 0x80, 0xa7, 0x77, 0x14, 0xa0, 0x35, 0x6d, 0x9b	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_INVALID_PARAMETER** with invalid leap day.	1. Call **SetWakeupTime()** with ***Time*** is 2001/2/29. The return code must be **EFI_UNSUPPORTED** or **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.2.2.4.22	0x4b660fec, 0xc2d0, 0x423f, 0xa3, 0x87, 0x07, 0x80, 0x41, 0xa1, 0x83, 0xb7	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_SUCCESS** with valid parameters at **EFI_TPL_APPLICATION**.	1. Call **SetWakeupTime()** with valid parameters. The return code should be **EFI_SUCCESS**.
5.2.2.4.23	0x218d16a6, 0xf52a, 0x4e42, 0x80, 0x52, 0x1a, 0x4d, 0x5d, 0x4a, 0x19, 0x60	**RT.SetWakeupTime – SetWakeupTime()** returns **EFI_SUCCESS** with valid parameters at **EFI_TPL_CALLBACK**.	1. Call **SetWakeupTime()** with valid parameters. The return code should be **EFI_SUCCESS**.
5.2.2.4.24	0x0da0ec8a, 0xb748, 0x4c42, 0xa8, 0xc6, 0x71, 0x03, 0x75, 0x32, 0x90, 0x71	**RT.SetWakeupTime – SetWakeupTime()** enables the wakeup time at **EFI_TPL_APPLICATION**.	1. Call **SetWakeupTime()** with valid parameters. 2. Call **GetWakeupTime()** to get the wakeup time. The return ***Enabled*** should be **TRUE**.
5.2.2.4.25	0x34aaf995, 0xd29b, 0x4892, 0xa4, 0x18, 0x99, 0x2c, 0xb0, 0xee, 0x29, 0xea	**RT.SetWakeupTime – SetWakeupTime()** enables the wakeup time at **EFI_TPL_CALLBACK**.	1. Call **SetWakeupTime()** with valid parameters. 2. Call **GetWakeupTime()** to get the wakeup time. The return ***Enabled*** should be **TRUE**.
5.2.2.4.26	0x49f3c56e, 0x013b, 0x4fa8, 0x8a, 0xb2, 0x17, 0x70, 0xd5, 0x37, 0x3d, 0x74	**RT.SetWakeupTime – SetWakeupTime()** enables the wakeup time at **EFI_TPL_APPLICATION**.	1. Call **SetWakeupTime()** with valid parameters. 2. Call **GetWakeupTime()** to get the wakeup time. The return ***Pending*** should be **FALSE**.
5.2.2.4.27	0xb39225e6, 0x3d06, 0x401c, 0xad, 0x26, 0x3e, 0xa9, 0x23, 0x71, 0xf3, 0xdc	**RT.SetWakeupTime – SetWakeupTime()** enables the wakeup time at **EFI_TPL_CALLBACK**.	1. Call **SetWakeupTime()** with valid parameters. 2. Call **GetWakeupTime()** to get the wakeup time. The return ***Pending*** should be **FALSE**.
5.2.2.4.28	0x6fd3d6d4, 0x2694, 0x4677, 0x87, 0x76, 0x3d, 0xd6, 0x2e, 0x3a, 0x8c, 0xa0	**RT.SetWakeupTime – SetWakeupTime()** enables the wakeup time at **EFI_TPL_APPLICATION**.	1. Call **SetWakeupTime()** with valid parameters. 2. Call **GetWakeupTime()** to get the wakeup time. The return ***Time*** should be set before.

Number	GUID	Assertion	Test Description
5.2.2.4.29	0xdf714d88, 0x9ee9, 0x4027, 0xa3, 0x70, 0xe5, 0xa2, 0x83, 0x56, 0x5c, 0xed	**RT.SetWakeupTime – SetWakeupTime()** enables the wakeup time at **EFI_TPL_CALLBACK**.	1. Call **SetWakeupTime()** with valid parameters. 2. Call **GetWakeupTime()** to get the wakeup time. The return ***Time*** should be set before.
5.2.2.4.30	0xd3835a5c, 0xb4be, 0x4f6c, 0xab, 0xf0, 0x29, 0x52, 0x52, 0x37, 0x14, 0x06	**RT.SetWakeupTime – SetWakeupTime()** disables the wakeup time with ***Enable*** is **FALSE** at **EFI_TPL_APPLICATION**.	1. Call **SetWakeupTime()** with ***Enable*** is **FALSE**. The return code must be **EFI_SUCCESS**.
5.2.2.4.31	0xeb8730ec, 0x578d, 0x41b1, 0xa2, 0xbe, 0x4a, 0x9f, 0xf6, 0x03, 0xdb, 0x22	**RT.SetWakeupTime – SetWakeupTime()** disables the wakeup time with ***Enable*** is **FALSE** at **EFI_TPL_CALLBACK**.	1. Call **SetWakeupTime()** with ***Enable*** is **FALSE**. The return code must be **EFI_SUCCESS**.
5.2.2.4.32	0xffaa1029, 0x16ae, 0x4d5c, 0xba, 0x74, 0x86, 0x80, 0xf4, 0xba, 0x9c, 0xd0	**RT.SetWakeupTime – SetWakeupTime()** disables the wakeup time with ***Enable*** is **FALSE** at **EFI_TPL_APPLICATION**.	1. Call **SetWakeupTime()** with ***Enable*** is **FALSE**. 2. Call **GetWakeupTime()** to get the wakeup time. The return ***Enabled*** must be **FALSE**.
5.2.2.4.33	0x8a70609a, 0xab54, 0x475e, 0x8d, 0xf2, 0xc3, 0xf9, 0x11, 0x58, 0xc4, 0xa8	**RT.SetWakeupTime – SetWakeupTime()** disables the wakeup time with ***Enable*** is **FALSE** at **EFI_TPL_CALLBACK**.	1. Call **SetWakeupTime()** with ***Enable*** is **FALSE**. 2. Call **GetWakeupTime()** to get the wakeup time. The return ***Enable*** must be **FALSE**.

4.3 Virtual Memory Services Test

Reference Document:

UEFI Specification, Virtual Memory Services Section.

Table 8. Virtual Memory Functions

Name	Type	Description
SetVirtualAddressMap	Runtime	Used by an OS loader to convert from physical addressing to virtual addressing.
ConvertPointer	Runtime	Used by EFI components to convert internal pointers when switching to virtual addressing.

No test case is designed to verify these functions in the EFI SCT.

4.4 Misc Runtime Services Test

Reference Document:

UEFI Specification, Miscellaneous Runtime Services Section.

Table 9. Miscellaneous Runtime Services Functions

Name	Type	Description
ResetSystem	Runtime	Reset the entire platform.
UpdateCapsule	Runtime	Passes capsules to the firmware with both virtual and physical mapping.
QueryCapsuleCapabilities	Runtime	Estimate if a capsule or capsules can be updated via UpdateCapsule()
GetNextHighMonotonicCount	Runtime	Returns the next high 32 bits of the platform's monotonic counter.

4.4.1 ResetSystem()

Number	GUID	Assertion	Test Description
5.2.4.1.1	0x26feed7e, 0x1501, 0x4c0a, 0xae, 0xf3, 0x86, 0xd6, 0x6b, 0xe2, 0xfc, 0xd0	**RT.ResetSystem – ResetSystem()** resets the platform with ***ResetType*** is **EfiResetCold** at **EFI_TPL_APPLICATION**.	1. Call **ResetSystem()** with a ***ResetType*** value of **EfiResetCold**. The system should be reset.
5.2.4.1.2	0x567f8ee9, 0x4e5e, 0x4278, 0x86, 0x3d, 0xdb, 0xc4, 0xd7, 0x4f, 0x0f, 0xba	**RT.ResetSystem – ResetSystem()** resets the platform with ***ResetType*** is **EfiResetCold** at **EFI_TPL_CALLBACK**.	1. Call **ResetSystem()** with a ***ResetType*** value of **EfiResetCold**. The system should be reset.
5.2.4.1.3	0xb7a21919, 0xf358, 0x4a1d, 0x85, 0x26, 0xcc, 0x52, 0x4c, 0x52, 0x94, 0xb2	**RT.ResetSystem – ResetSystem()** resets the platform with ***ResetType*** is **EfiResetCold** at **EFI_TPL_NOTIFY**.	1. Call **ResetSystem()** with a ***ResetType*** value of **EfiResetCold**. The system should be reset.
5.2.4.1.4	0x7bbad1aa, 0x88b4, 0x4d66, 0x95, 0x94, 0xdb, 0x7e, 0x65, 0xe1, 0xd3, 0xa4	**RT.ResetSystem – ResetSystem()** resets the platform with ***ResetType*** is **EfiResetWarm** at **EFI_TPL_APPLICATION**.	1. Call **ResetSystem()** with a ***ResetType*** value of **EfiResetWarm**. The system should be reset.

Number	GUID	Assertion	Test Description
5.2.4.1.5	0xdbe1128b, 0x5155, 0x4241, 0x84, 0x1e, 0x54, 0xea, 0x76, 0x3a, 0x85, 0xc9	**RT.ResetSystem – ResetSystem()** resets the platform with ***ResetType*** is **EfiResetWarm** at **EFI_TPL_CALLBACK**.	1. Call **ResetSystem()** with a ***ResetType*** value of **EfiResetWarm**. The system should be reset.
5.2.4.1.6	0x8128b536, 0x0b56, 0x480b, 0xa2, 0xd4, 0xcd, 0x79, 0xf8, 0xfa, 0xcb, 0x3b	**RT.ResetSystem – ResetSystem()** resets the platform with ***ResetType*** is **EfiResetWarm** at **EFI_TPL_NOTIFY**.	1. Call **ResetSystem()** with a ***ResetType*** value of **EfiResetWarm**. The system should be reset.
5.2.4.1.7	0x1189a0df, 0xe9cc, 0x45e6, 0xbb, 0x94, 0x21, 0xa7, 0xb3, 0x42, 0x70, 0x96	**RT.ResetSystem – ResetSystem()** resets the platform with ***ResetType*** is **EfiResetShutdown** at **EFI_TPL_APPLICATION**.	1. Call **ResetSystem()** with a ***ResetType*** value of **EfiResetShutdown**. The system should be reset or shut down.
5.2.4.1.8	0x22b8b295, 0x62a2, 0x4e14, 0xb8, 0x5b, 0xd2, 0xde, 0x36, 0x37, 0x15, 0xb5	**RT.ResetSystem – ResetSystem()** resets the platform with ***ResetType*** is **EfiResetShutdown** at **EFI_TPL_CALLBACK**.	1. Call **ResetSystem()** with a ***ResetType*** value of **EfiResetShutdown**. The system should be reset or shut down.
5.2.4.1.9	0x1ed1babb, 0x6521, 0x4515, 0x93, 0x9a, 0x39, 0x26, 0xc8, 0xe3, 0x12, 0xff	**RT.ResetSystem – ResetSystem()** resets the platform with ***ResetType*** is **EfiResetShutdown** at **EFI_TPL_NOTIFY**.	1. Call **ResetSystem()** with a ***ResetType*** value of **EfiResetShutdown**. The system should be reset or shut down.

4.4.2 UpdateCapsule()

Number	GUID	Assertion	Test Description
5.2.4.2.1	0xf48a2ac4, 0xbce7, 0x4fa7, 0x9e, 0x1b, 0xb9, 0x6f, 0xf8, 0x60, 0xe3, 0x0a	**RT.UpdateCapsule – UpdateCapsule()** returns **EFI_INVALID_PARAMETER** or **EFI_UNSUPPORTED** with ***CapsuleCount*** is **NULL**.	1. Call **UpdateCapsule()** with a ***CapsuleCount*** value of **NULL**. The return value should be **EFI_INVALID_PARAMETER** or **EFI_UNSUPPORTED**.

Number	GUID	Assertion	Test Description
5.2.4.2.2	0x304f6960, 0x79d0, 0x4f17, 0x88, 0x11, 0x62, 0x0f, 0xc6, 0xbd, 0xb0, 0xd4	**RT. UpdateCapsule-UpdateCapsule()** returns **EFI_INVALID_PARAMETER** or **EFI_UNSUPPORTED** when a capsule has the **CAPSULE_FLAGS_PERSIST_ACROSS_RESET** in its header, but the ***ScatterGatherList*** is **NULL**.	1. Call **UpdateCapsule()** with ***ScatterGatherList*** is **NULL** and a capsule has the flag of **CAPSULE_FLAGS_PERSIST_ACROSS_RESET** in its header.The return value should be **EFI_INVALID_PARAMETER** or **EFI_UNSUPPORTED**.
5.2.4.2.3	0x18f86bf8, 0x76cf, 0x4225, 0x8e, 0x3e, 0x1b, 0x1f, 0x63, 0x43, 0x26, 0x00	**RT.UpdateCapsule-UpdateCapsule()** returns **EFI_INVALID_PARAMETER** or **EFI_UNSUPPORTED** when a capsule has the flag of **CAPSULE_FLAGS_POPULATE_SYSTEM_TABLE** in its header only.	1. Call **UpdateCapsule()** when a capsule has the flag of **CAPSULE_FLAGS_POPULATE_SYSTEM_TABLE** in its header only. The return value should be **EFI_INVALID_PARAMETER** or **EFI_UNSUPPORTED**

4.4.3 QueryCapsuleCapabilities()

Number	GUID	Assertion	Test Description
5.2.4.3.1	0x5b5f42d4, 0x8985, 0x45a0, 0x9d, 0xf2, 0x21, 0xaf, 0x74, 0xb1, 0xf5, 0xf6	**RT.QueryCapsuleCapabilities-QueryCapsuleCapabilities()** query for generic capsule capability with a fake **EFI_CAPSULE_HEADER**. **CAPSULE_FLAGS_PERSIST_ACROSS_RESET** is set in the flags in the header.	1. Call **QueryCapsuleCapabilities()** with a fake **EFI_CAPSULE_HEADER.** The return value should be **EFI_SUCCESS** or **EFI_UNSUPPORTED**.
5.2.4.3.2	0x13826168, 0xfef6, 0x407e, 0x93, 0x7c, 0x6d, 0x5e, 0x32, 0x34, 0x9d, 0x5c	**RT.QueryCapsuleCapabilities-QueryCapsuleCapabilities()** query for generic capsule capability with a fake **EFI_CAPSULE_HEADER**. **0** is set in the flags in the header.	1. Call **QueryCapsuleCapabilities()** with a fake **EFI_CAPSULE_HEADER.** The return value should be **EFI_SUCCESS** or **EFI_UNSUPPORTED**.
5.2.4.3.3	0x67c3c36d, 0x4cf8, 0x41fb, 0xa7, 0x8a, 0x86, 0x36, 0x84, 0xe9, 0xe6, 0xe4	**RT.QueryCapsuleCapabilities-QueryCapsuleCapabilities()** query for generic capsule capability with ***MaximumCapsuleSize*** is **NULL**.	1. Call **QueryCapsuleCapabilities()** with ***MaximumCapsuleSize*** is **NULL**. The return value should be **EFI_INVALID_PARAMETER** or **EFI_UNSUPPORTED**

4.4.4 GetNextHighMonotonicCount()

This function may only be available in Runtime. No test case is designed to verify it.

5 Protocols EFI Loaded Image Test

5.1 EFI_LOADED_IMAGE Protocol Test

Reference Document:

UEFI Specification, EFI_LOADED_IMAGE_PROTOCOL Section.

Number	GUID	Assertion	Test Description
5.3.1.1.1	0xb324a56f, 0x5714, 0x44b4, 0xa2, 0x0f, 0x6e, 0x9b, 0x13, 0x7b, 0x8d, 0xf9	**EFI_LOADED_IMAGE_PROTOCOL –** **BS.HandleProtocol()** to handle Loaded Image Protocol returns **EFI_SUCCESS**.	1. Call **BS.LoadImage()** to get image handle by filename. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. The return code should be **EFI_SUCCESS**.
5.3.1.1.2	0xbce0c845, 0x4ce1, 0x4c3b, 0x9f, 0x94, 0x84, 0x6c, 0x27, 0x9c, 0x93, 0xd0	**EFI_LOADED_IMAGE_PROTOCOL –** *Revision* is equal to **EFI_IMAGE_INFORMATION_REVISION**	1. Call **BS.LoadImage()** to get image handle by filename. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. *Revision* on each image handle should equal **EFI_IMAGE_INFORMATION_REVISION**.
5.3.1.1.3	0x12b28b7b, 0x8255, 0x4fad, 0xb3, 0x05, 0x81, 0x31, 0x16, 0x71, 0xb2, 0xe1	**EFI_LOADED_IMAGE_PROTOCOL –** *ParentHandle* is equal to the test driver's image handle	1. Call **BS.LoadImage()** to get image handle by filename. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. *ParentHandle* should be equal to the test driver's image handle.
5.3.1.1.4	0xb8e8ce9f, 0x3324, 0x4134, 0xab, 0x08, 0x3f, 0x3c, 0x9e, 0xe2, 0x5c, 0x27	**EFI_LOADED_IMAGE_PROTOCOL –** *SystemTable* is not **NULL**.	1. Call **BS.LoadImage()** to get image handle by filename. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. *SystemTable* should not be **NULL**.
5.3.1.1.5	0x3bf1e23d, 0x86e1, 0x4f8a, 0x8c, 0x1a, 0x7f, 0xdc, 0x5c, 0x49, 0x11, 0xb9	**EFI_LOADED_IMAGE_PROTOCOL – DeviceHandle** is not **NULL**.	1. Call **BS.LoadImage()** to get image handle by filename. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. **DeviceHandle** should not be **NULL**.

Number	GUID	Assertion	Test Description
5.3.1.1.6	0x7df05248, 0x72ff, 0x40a5, 0x94, 0x8c, 0xc6, 0x47, 0xd1, 0xfd, 0xc1, 0xae	**EFI_LOADED_IMAGE_PROTOCOL** – *ImageBase* is not **NULL** and *ImageSize* is not 0.	1. Call **BS.LoadImage()** to get image handle by filename. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. *ImageBase* is not **NULL** and *ImageSize* is not 0.
5.3.1.1.7	0xfede5dd0, 0x92f6, 0x42de, 0x81, 0x4f, 0xf2, 0xe3, 0x33, 0x9b, 0x5d, 0xe1	**EFI_LOADED_IMAGE_PROTOCOL** – Application image's *ImageCodeType* equals **EfiLoaderCode** and *ImageDataType* equals **EfiLoaderData**.	1. Call **BS.LoadImage()** to get image handle by filename. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. Application image's *ImageCodeType* should be **EfiLoaderCode** and *ImageDataType* should be **EfiLoaderData**.
5.3.1.1.8	0x9ead501b, 0x4a09, 0x4c24, 0xba, 0x47, 0xcf, 0x27, 0xbf, 0xf0, 0x66, 0xdb	**EFI_LOADED_IMAGE_PROTOCOL** – **BootService** image's *ImageCodeType* equals **EfiBootServiceCode** and *ImageDataType* equals **EfiBootServiceData**.	1. Call **BS.LoadImage()** to get image handle by filename. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. **BootService** image's *ImageCodeType* equals **EfiBootServiceCode** and *ImageDataType* equals **EfiBootServiceData**.
5.3.1.1.9	0x064e5c37, 0xcfaf, 0x4b5a, 0xa2, 0xa0, 0xf6, 0x17, 0xdd, 0x41, 0xa4, 0x12	**EFI_LOADED_IMAGE_PROTOCOL** – **RuntimeService** image's *ImageCodeType* equals **EfiRuntimeServiceCode** and *ImageDataType* equals **EfiRuntimeServiceData**.	1. Call **BS.LoadImage()** to get image handle by filename. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. **RuntimeService** image's *ImageCodeType* equals **EfiRuntimeServiceCode** and *ImageDataType* equals **EfiRuntimeServiceData**.
5.3.1.1.10	0xc7606256, 0x8a89, 0x48ce, 0xb5, 0x7b, 0xa1, 0xb0, 0x6b, 0x3c, 0x62, 0x3b	**EFI_LOADED_IMAGE_PROTOCOL** – **Unload()** is **NULL** if the image has no Unload function.	1. Call **BS.LoadImage()** to get image handle by filename. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. Check on Application Images which have no unload function. Unload field should be **NULL**.

Number	GUID	Assertion	Test Description
5.3.1.1.11	0xfc2330ce, 0xaa7a, 0x4c64, 0xac, 0x5e, 0xfe, 0xb1, 0xf0, 0xf7, 0xda, 0xc7	**EFI_LOADED_IMAGE_PROTOCOL – Unload()** is not **NULL** and its address is valid if the image has Unload function.	1. Call **BS.LoadImage()** to get image handle by filename. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. Check on Application Images which have Unload function. Unload field should be valid and its entry address should be within the range of [*ImageBase*, *ImageBase*+*ImageSize*]
5.3.1.1.12	0x69cb9798, 0x5b57, 0x4381, 0xb9, 0xb2, 0x54, 0xb9, 0xa2, 0x4b, 0x8d, 0x16	**EFI_LOADED_IMAGE_PROTOCOL –** *LoadOptions* is used in notify function.	1. Call **BS.LoadImage()** to get image handle by filename with specified *LoadOptions*. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. Call **BS.StartImage()**. *LoadOptions* should be used.
5.3.1.1.13	0x6da9aef4, 0xdadd, 0x4bda, 0xa7, 0x0d, 0x29, 0x47, 0x0e, 0x05, 0xf3, 0x17	**EFI_LOADED_IMAGE_PROTOCOL –** *LoadOptions* is used in notify function.	1. Call **BS.LoadImage()** to get image handle by filename with specified *LoadOptions*. 2. Call **BS.HandleProtocol()** to handle Loaded Image Protocol on each image handle. 3. Call **BS.StartImage()**. *LoadOptions* should be used. 4. Unload Image. 5. Change *LoadOptions* and call **BS.LoadImage()** again. 6. Call **BS.HandleProtocol()** and **BS.StartImage()**. Updated *LoadOptions* value should be used.
5.3.1.1.14	0x0caae7f5, 0x0742, 0x458f, 0xbf, 0x02, 0x65, 0x2d, 0x33, 0xa4, 0xf1, 0xab	**EFI_LOADED_IMAGE_PROTOCOL –** *SystemTable* is not **NULL**	1. Check on all images in system. *SystemTable* should not be **NULL**.
5.3.1.1.15	0xa7bc2e01, 0x3162, 0x482c, 0xa6, 0x8b, 0x93, 0x9d, 0x0c, 0xf7, 0x9a, 0x45	**EFI_LOADED_IMAGE_PROTOCOL –** *ImageBase* is not **NULL** and *ImageSize* is not 0	1. Check on all images in system. *ImageBase* is not **NULL** and *ImageSize* is not 0.

Number	GUID	Assertion	Test Description
5.3.1.1.16	0xa3ada89a, 0xef4e, 0x475b, 0xbc, 0x53, 0x00, 0x98, 0xd5, 0xc6, 0x5b, 0xee	**EFI_LOADED_IMAGE_PROTOCOL** – *ImageCodeType* matches with the *ImageDataType*.	1. Check on all images in system. If *ImageCodeType* is **EfiLoaderCode**, *ImageDataType* should be **EfiLoaderData**; If *ImageCodeType* is **EfiBootServicesCode**, *ImageDataType* should be **EfiBootServicesData**; If *ImageCodeType* is **EfiRuntimeServicesCode**, *ImageDataType* should be **EfiRuntimeServicesData**;
5.3.1.1.17	0xda215e1d, 0x5ac8, 0x480a, 0xa7, 0x9e, 0xa0, 0x66, 0xb9, 0x74, 0x58, 0x65	**EFI_LOADED_IMAGE_PROTOCOL** – If **Unload()** function is not **NULL**, its address is valid.	1. Check on all images in system. If **Unload()** function is not **NULL**, its address should be within the range of [*ImageBase*, *ImageBase*+*ImageSize*]
5.3.1.1.18	0xe2f6c4a6, 0xe2a8, 0x4bab, 0x94, 0xbb, 0x70, 0x44, 0x54, 0xd6, 0x2a, 0xea	**EFI_LOADED_IMAGE_PROTOCOL** – *Revision* equals **EFI_IMAGE_INFORMATION_REVISION**.	1. Check *Revision* on all file images. *Revision* should be equal to **EFI_IMAGE_INFORMATION_REVISION**

6 Protocols Device Path Protocol Test

6.1 Device Path Node Conformance Test

Reference Document:

UEFI Specification, Device Path Nodes Section.

Number	GUID	Assertion	Test Description
5.4.1.1.1	0x91064ab1, 0x5408, 0x48c1, 0xbb, 0xd9, 0x2a, 0x49, 0xee, 0xe2, 0x1d, 0xc9	**EFI_DEVICE_PATH_PROTOCOL –** Check End of Hardware Device Path - End This Device Path.	Verify the device path nodes. Type: 0x7F or 0xFF. Sub-Type: 0xFF. Length: 4 bytes.
5.4.1.1.2	0xb5a0ee55, 0x0070, 0x472d, 0x84, 0xcd, 0xbc, 0xb1, 0xe2, 0xbc, 0x25, 0xc0	**EFI_DEVICE_PATH_PROTOCOL –** Check Hardware Device Path - PCI Device Path.	Verify the device path nodes. Type: 1.Sub-Type: 1.Length: 6 bytes.
5.4.1.1.3	0x2902b389, 0xe4e7, 0x43cd, 0x9e, 0xff, 0xdc, 0x3f, 0xaa, 0xff, 0x12, 0xfa	**EFI_DEVICE_PATH_PROTOCOL –** Check Hardware Device Path - PCCARD Device Path.	Verify the device path nodes. Type: 1.Sub-Type: 2.Length: 5 bytes
5.4.1.1.4	0x745df5f1, 0x7d97, 0x4297, 0xaf, 0x5a, 0xc5, 0xca, 0x67, 0x28, 0x39, 0x18	**EFI_DEVICE_PATH_PROTOCOL –** Check Hardware Device Path - Memory Mapped Device Path.	Verify the device path nodes. Type: 1.Sub-Type: 3.Length: 24 bytes. Memory Type < **EfiMaxMemoryType**, or Memory Type > 0x7FFFFFFF.End Address >= Start Address.
5.4.1.1.5	0xc8f02111, 0x1de9, 0x4df2, 0x8f, 0x17, 0xbb, 0x12, 0x9b, 0xa6, 0x4d, 0xfe	**EFI_DEVICE_PATH_PROTOCOL –** Check Hardware Device Path - Vendor Device Path.	Verify the device path nodes. Type: 1.Sub-Type: 4.Length>=20 bytes.

Number	GUID	Assertion	Test Description
5.4.1.1.6	0x1c206e49, 0x6638, 0x469d, 0x8c, 0x9c, 0x26, 0x13, 0x85, 0x8e, 0x4d, 0x77	EFI_DEVICE_PATH_PROTOCOL – Check Hardware Device Path - Controller Device Path.	Verify the device path nodes. Type: 1.Sub-Type: 5.Length: 8 bytes.
5.4.1.1.7	0xcedef0c0, 0x24cc, 0x4d36, 0x9d, 0x31, 0x9b, 0x9a, 0xf4, 0x63, 0xe6, 0x95	EFI_DEVICE_PATH_PROTOCOL – Check ACPI Device Path - ACPI Device Path.	Verify the device path nodes. Type: 2.Sub-Type: 1.Length: 12 bytes.
5.4.1.1.8	0xf497a21b, 0x8bb4, 0x4310, 0xba, 0xcf, 0xf6, 0xfc, 0x18, 0xda, 0x46, 0x9e	EFI_DEVICE_PATH_PROTOCOL – Check ACPI Device Path - Expanded ACPI Device Path.	Verify the device path nodes. Type: 2.Sub-Type: 2.Length>=19 bytes.
5.4.1.1.9	0xc3b2ba41, 0x7126, 0x4b7a, 0xab, 0xdc, 0x7d, 0x1b, 0x46, 0x3d, 0x9b, 0xd7	EFI_DEVICE_PATH_PROTOCOL – Check ACPI _ADR Device Path - ACPI _ADR Device Path.	Verify the device path nodes. Type: 2.Sub-Type: 3. Length>=8 bytes.
5.4.1.1.10	0xf52ef05c, 0x4a10, 0x4857, 0xb9, 0x8c, 0x01, 0xd8, 0x15, 0x6e, 0xf8, 0x3f	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - ATAPI Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 1.Length: 8 bytes. PrimarySecondary: 0 or 1.SlaveMaster: 0 or 1.
5.4.1.1.11	0x3e3eaf27, 0xf811, 0x4060, 0x97, 0xe1, 0x13, 0xfc, 0x5a, 0x51, 0x6c, 0x0c	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - SCSI Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 2.Length: 8 bytes.
5.4.1.1.12	0x8f24a32d, 0xb167, 0x42df, 0x85, 0xc3, 0xa3, 0xec, 0x68, 0x4a, 0x79, 0x80	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - Fibre Channel Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 3.Length: 24 bytes.

Number	GUID	Assertion	Test Description
5.4.1.1.13	0xfd1e18a9, 0x0fd6, 0x4ea4, 0xac, 0xea, 0xe6, 0xc4, 0xd1, 0x73, 0x97, 0x52	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - 1394 Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 4.Length: 16 bytes.
5.4.1.1.14	0x758cfe7a, 0x1463, 0x4f29, 0x8c, 0x5b, 0x0e, 0x3a, 0x04, 0x17, 0x5d, 0xf8	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - USB Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 5.Length: 6 bytes.
5.4.1.1.15	0xd1527a5c, 0xc1bd, 0x4585, 0x93, 0x23, 0xa5, 0xea, 0xc7, 0xd5, 0x12, 0x7b	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - USB Device Path (WWID).	Verify the device path nodes. Type: 3.Sub-Type: 16. Length >=10bytes.
5.4.1.1.16	0x50e59956, 0x46fd, 0x4b21, 0xb5, 0x57, 0x9a, 0x33, 0xb2, 0x08, 0xd3, 0x41	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - Device Logical Unit.	Verify the device path nodes. Type: 3.Sub-Type: 17. Length: 5 bytes.
5.4.1.1.17	0x2eb2da32, 0x351d, 0x4743, 0x80, 0x55, 0xea, 0x23, 0x75, 0x69, 0x61, 0xc2	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path – USB Device Path (Class).	Verify the device path nodes. Type: 3.Sub-Type: 15.Length: 11 bytes.
5.4.1.1.18	0xba91dcd7, 0x719d, 0x4803, 0xaf, 0xe2, 0x61, 0x02, 0x1b, 0x31, 0x9b, 0x1f	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - I2O Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 6.Length: 8 bytes.

Number	GUID	Assertion	Test Description
5.4.1.1.19	0xb10c12a3, 0x8faa, 0x408a, 0x83, 0x63, 0x35, 0x6c, 0x74, 0x95, 0xe6, 0x80	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - MAC Address Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 11.Length: 37 bytes.
5.4.1.1.20	0xdd68e9c3, 0x28e1, 0x44c7, 0x9c, 0x31, 0xba, 0xcc, 0x80, 0x4e, 0xe4, 0xb3	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - IPv4 Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 12.Length: 19 bytes. StaticIPAddress: 0x00 or 0x01.
5.4.1.1.21	0x2da145c3, 0x7d26, 0x4715, 0x8e, 0xfb, 0xf2, 0x35, 0xd5, 0x51, 0xe0, 0x77	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - IPv6 Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 13.Length: 43 bytes. StaticIPAddress: 0x00 or 0x01.
5.4.1.1.22	0xfcac17d1, 0xc792, 0x417a, 0x86, 0x99, 0x26, 0x11, 0xd0, 0xae, 0xc5, 0xba	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - InfiniBand Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 9.Length: 48 bytes.
5.4.1.1.23	0x5f832ee4, 0x1d93, 0x42bf, 0x94, 0xea, 0xf8, 0x1b, 0x30, 0x1a, 0x9e, 0xf7	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - UART Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 14.Length: 19 bytes. Parity: 0x00~0x05.Stop Bits: 0x00~0x03.
5.4.1.1.24	0x86499222, 0x650a, 0x4492, 0x92, 0x2d, 0x46, 0x84, 0x4b, 0x1e, 0xb2, 0x0f	EFI_DEVICE_PATH_PROTOCOL – Check Messaging Device Path - Vendor-Defined Device Path.	Verify the device path nodes. Type: 3.Sub-Type: 10.Length>=20 bytes.

Number	GUID	Assertion	Test Description
5.4.1.1.25	0x4c19f495, 0x7214, 0x48da, 0xb4, 0xc5, 0x2e, 0x6c, 0xae, 0xd2, 0x8f, 0xc9	**EFI_DEVICE_PATH_PROTOCOL** – Check Messaging Device Path - UART Flow Control Messaging Path.	Verify the device path nodes. Type: 3.Sub-Type: 10.Length: 24 bytes. Vendor_GUID: DEVICE_PATH_MESSAGING_UART_FLOW_CONTROL. Flow_Control_Map: 0 or 1.
5.4.1.1.26	0x8e637c03, 0xa1df, 0x4ab6, 0xae, 0x29, 0x5b, 0x9c, 0xd8, 0x6c, 0x6d, 0x1e	**EFI_DEVICE_PATH_PROTOCOL** – Check Messaging Device Path - Serial Attached SCSI (SAS) Device Path.	Verify the device path nodes. Type*:* 3.Sub-Type*:* 10.*Length*: 44 bytes. Vendor_GUID*:* DEVICE_PATH_MESSAGING_SAS
5.4.1.1.27	0x885db334, 0x940b, 0x4ec3, 0x82, 0xe5, 0xc5, 0xf1, 0x1d, 0xdb, 0x2a, 0x42	**EFI_DEVICE_PATH_PROTOCOL** – Check Messaging Device Path - iSCSI Device Path.	Verify the device path nodes. Type: 3.Sub-Type*:* 19.*Length>=18* bytes. Options*:* Bit0=0x0, Bit2=0x0, Bit10=0x0
5.4.1.1.28	0x1856d9b9, 0x57db, 0x49eb, 0x97, 0x35, 0x68, 0x8a, 0xee, 0x43, 0x76, 0xf6	**EFI_DEVICE_PATH_PROTOCOL** – Check Media Device Path - Hard Drive Media Device Path.	Verify the device path nodes. Type: 4.Sub-Type: 1.Length: 42 bytes. MBR Type: 0x01 or 0x02.Signature Type: 0x00, 0x01 or 0x02.
5.4.1.1.29	0x8b53dc1e, 0xb9be, 0x49d7, 0x86, 0xad, 0xd5, 0x12, 0x8e, 0x1f, 0x08, 0x34	**EFI_DEVICE_PATH_PROTOCOL** – Check Media Device Path - CD-ROM Media Device Path.	Verify the device path nodes. Type: 4.Sub-Type: 2.Length: 24 bytes.
5.4.1.1.30	0x4c60bb0c, 0x8c00, 0x40f8, 0xa7, 0x35, 0x13, 0x4a, 0x56, 0x28, 0xe5, 0x21	**EFI_DEVICE_PATH_PROTOCOL** – Check Media Device Path - Vendor-Defined Media Device Path.	Verify the device path nodes. Type: 4.Sub-Type: 3.Length>=20 bytes.
5.4.1.1.31	0xde41b8cb, 0x401f, 0x4b7f, 0xb2, 0x34, 0xf8, 0xfb, 0x29, 0x3f, 0xc5, 0x23	**EFI_DEVICE_PATH_PROTOCOL** – Check Media Device Path - File Path Media Device Path.	Verify the device path nodes. Type: 4.Sub-Type: 4.Length>=4 bytes.

Number	GUID	Assertion	Test Description
5.4.1.1.32	0xc9969745, 0x6507, 0x4695, 0xb1, 0x26, 0xc3, 0xf8, 0xe6, 0xd2, 0x86, 0xec	**EFI_DEVICE_PATH_PROTOCOL –** Check Media Device Path - Media Protocol Device Path.	Verify the device path nodes. Type: 4.Sub-Type: 5.Length: 20 bytes.
5.4.1.1.33	0x014988e5, 0xc211, 0x478d, 0x90, 0x6d, 0xf1, 0x6a, 0xb0, 0x73, 0x85, 0x0c	**EFI_DEVICE_PATH_PROTOCOL –** Check BIOS Boot Specification Device Path.	Verify the device path nodes. Type: 5.Sub-Type: 1.Length>=8 bytes.
5.4.1.1.34	0x3152ee5d, 0xd161, 0x4916, 0xa4, 0x13, 0x44, 0xa7, 0x79, 0x39, 0x16, 0x7f	**EFI_DEVICE_PATH_PROTOCOL –** Check End of Hardware Device Path - End Entire Device Path.	Verify the device path nodes. Type: 0x7F or 0xFF.Sub-Type: 0xFF.Length: 4 bytes.
5.4.1.1.35	0xab5c791b, 0x015c, 0x41b2, 0x93, 0xdf, 0x70, 0xf5, 0xc8, 0xaf, 0x3a, 0xec	EFI_DEVICE_PATH_PROTOCOL – Check SATA Device Path.	Verify the device path nodes. Type: 3. SubType: 18. Length: 10 bytes
5.4.1.1.36	0x2bbca783, 0x4c23, 0x477d, 0xa7, 0x50, 0xf3, 0xda, 0xfa, 0xbc, 0x38, 0xf6	EFI_DEVICE_PATH_PROTOCOL – Check PIWG Fireware Volume	Verify the device path nodes. Type: 4. SubType: 6. Length >= 4 bytes.
5.4.1.1.37	0xbaaf24e1, 0x0c59, 0x4494, 0xaf, 0xef, 0x53, 0x02, 0xc1, 0x90, 0x57, 0x29	EFI_DEVICE_PATH_PROTOCOL – Check PIWG Fireware File	Verify the device path nodes. Type: 4. SubType: 7. Length >= 4 bytes.

Number	GUID	Assertion	Test Description
5.4.1.1.38	0xbe55aaa6, 0x7510, 0x4904, 0x98, 0x65, 0x8c, 0xa7, 0x16, 0x34, 0xd2, 0x03	EFI_DEVICE_PATH_PROTOCOL - Controller Device Path Node.	Verify the device path nodes. Type: 3. SubType: 20. 0 < VlanId < 4095
5.4.1.1.39	0x5658c849, 0xd7ed, 0x4780, 0x8e, 0xe7, 0x6d, 0xf2, 0x62, 0x48, 0x1d, 0xdb	EFI_DEVICE_PATH_PROTOCOL – Check Fibre Channel Ex	Verify the device path nodes. Type: 3. SubType: 21.
5.4.1.1.40	0x3f412961, 0x4872, 0x4aa9, 0xbe, 0xd2, 0x2b, 0x03, 0x5f, 0xbc, 0xcc, 0xb6	EFI_DEVICE_PATH_PROTOCOL – Check Serial Attached SCSI(SAS) Ex.	Verify the device path nodes. Type: 3. SubType: 22.
5.4.1.1.41	0x2ed116cb, 0x1ec7, 0x468a, 0x9c, 0xf8, 0x0f, 0xf4, 0x41, 0x2a, 0x4b, 0xb1	EFI_DEVICE_PATH_PROTOCOL – Check NVM Express.	Verify the device path nodes. Type: 3. SubType: 23, Length = 16 bytes.
5.4.1.1.42	0x64770fbb, 0x280f, 0x40d5, 0x80, 0x33, 0x7, 0x82, 0x44, 0x7b, 0x3a, 0x2b	EFI_DEVICE_PATH_PROTOCOL - Check Hardware Device Path - BMC Device Path.	Verify the device path nodes. Type: 1. Sub-Type: 6. Length: 13 bytes. InterfaceType >= 0 and InterfaceType <= 3
5.4.1.1.43	0x88882137, 0x4e4d, 0x445a, 0xa1, 0xae, 0x11, 0xd8, 0xc2, 0xe1, 0xcf, 0xac	EFI_DEVICE_PATH_PROTOCOL - Check Messaging Device Path - Uniform Resource Identifiers (URI) Device Path	Verify the device path nodes. Type: 3. Sub-Type: 24. Length: >= 4 bytes.
5.4.1.1.44	0xda928c4a, 0x6d22, 0x4091, 0x95, 0x8c, 0xe, 0xde, 0xa5, 0x3b, 0xc8, 0x2e	EFI_DEVICE_PATH_PROTOCOL - Check Messaging Device Path - Universal Flash Storage (UFS) Device Path	Verify the device path nodes. Type: 3. Sub-Type: 25. Length: 6 bytes

Number	GUID	Assertion	Test Description
5.4.1.1.45	0x71e0582d, 0x983, 0x468e, 0x9a, 0x5d, 0xd2, 0xe5, 0xbb, 0x8c, 0x52, 0x6c	**EFI_DEVICE_PATH_PROTOCOL -** Check Messaging Device Path - Secure Digital (SD) Device Path	Verify the device path nodes. Type: 3. Sub-Type: 26. Length: 5 bytes
5.4.1.1.46	0x3d20f5d0, 0x670a, 0x4923, 0x91, 0x78, 0xb0, 0x1e, 0x6d, 0xe8, 0xee, 0x13	**EFI_DEVICE_PATH_PROTOCOL -** Check Messaging Device Path - Bluetooth Device Path	Verify the device path nodes. Type: 3. Sub-Type: 27. Length: 10 bytes
5.4.1.1.47	0x136c50de, 0xb2d4, 0x4416, 0xb4, 0x90, 0xe, 0x32, 0x85, 0xf1, 0x6a, 0x7	**EFI_DEVICE_PATH_PROTOCOL -** Check Messaging Device Path - WIFI Device Path	Verify the device path nodes. Type: 3. Sub-Type: 28. Length: 36 bytes
5.4.1.1.48	0x973269de, 0xdca6, 0x4ad9, 0x9b, 0x9b, 0x6, 0x40, 0xfa, 0x4d, 0xbd, 0xf5	**EFI_DEVICE_PATH_PROTOCOL -** Check Relative Offset Range Device Path	Verify the device path nodes. Type: 4. Sub-Type: 8. Length: 24 bytes
5.4.1.1.49	0x6e817459, 0x21fd, 0x4923, 0x89, 0xe7, 0xca, 0xf9, 0x7d, 0x9d, 0xc2, 0x27	**EFI_DEVICE_PATH_PROTOCOL -** Check RAM Disk Device Path	Verify the device path nodes. Type: 4. Sub-Type: 9. Length: 38 bytes
5.4.1.1.50	0xdf69547d, 0xd032, 0x44bd, 0xb0, 0x54, 0x7f, 0x34, 0x3c, 0x2c, 0x7d, 0x95	**EFI_DEVICE_PATH_PROTOCOL –** Check eMMC Device Path.	Verify the device path node. Type: 3. Sub-Type: 29. Length: 5 bytes

6.2 Whole Device Path Conformance Test

Number	GUID	Assertion	Test Description
5.4.2.1.1	0x4d36889a, 0x938a, 0x45ae, 0xaa, 0x79, 0x89, 0x7f, 0xa3, 0x7e, 0x15, 0x99	EFI_DEVICE_PATH_PROTOCOL – BIOS Root Specification Device Path.	A Device Path containing the BIOS Boot Specification Device Path should contain only the required End Device Path structure and no other Device Path structures.
5.4.2.1.2	0xf141747c, 0xf5f8, 0x43b9, 0x99, 0x9e, 0x45, 0xad, 0x37, 0xe1, 0x2a, 0x49	EFI_DEVICE_PATH_PROTOCOL – PCI Root Bus Device Path Node.	The device path node for PCI root bus is: ACPI Device Path: _HID PNP0A03. It must be the first device path node.
5.4.2.1.3	0xc44987b4, 0x9a29, 0x4b10, 0x82, 0xd3, 0xe9, 0x46, 0x81, 0x7e, 0x3c, 0x02	EFI_DEVICE_PATH_PROTOCOL – ACPI Device Path Node.	ACPI _CRS Device Path Node must include Floppy – ACPI Device Path: _HID PNP0604 Keyboard – ACPI Device Path: _HID PNP0301 Serial Port – ACPI Device Path: _HID PNP0501 Parallel Port – ACPI Device Path: _HID PNP0401. EISA Device Path Nodes other than PCI Root Bus must be preceded by an ACPI Device Path Node.
5.4.2.1.4	0xb28b09c6, 0x3b60, 0x48ce, 0xbf, 0x66, 0xac, 0xa1, 0xf6, 0x20, 0x6b, 0x01	EFI_DEVICE_PATH_PROTOCOL – PCI Device Path Node.	The PCI Device Path Node must be preceded by an ACPI Device Path Node that uniquely identifies the PCI root bus (Acpi(PNP0A03,0)) or another PCI Device Path Node.
5.4.2.1.5	0x47f98975, 0x2945, 0x4198, 0x99, 0xa0, 0x7b, 0x07, 0xfe, 0xe0, 0x9b, 0x85	EFI_DEVICE_PATH_PROTOCOL – Memory Mapped Device Path Node.	The Memory Mapped Device Path Node must be the first device path node.
5.4.2.1.6	0xfc86d0ef, 0xb3da, 0x4377, 0x99, 0x36, 0x56, 0x85, 0xb4, 0x59, 0x9e, 0x24	EFI_DEVICE_PATH_PROTOCOL – ATAPI Device Path Node.	The ATAPI Device Path Node must be preceded by a PCI Device Path Node.
5.4.2.1.7	0x390d6af3, 0x78a8, 0x41ed, 0x99, 0x78, 0x16, 0x4d, 0xfe, 0x2b, 0x30, 0xc8	EFI_DEVICE_PATH_PROTOCOL – SCSI Device Path Node.	The SCSI Device Path Node must be preceded by a PCI Device Path Node.

Number	GUID	Assertion	Test Description
5.4.2.1.8	0xd456e708, 0x5b3c, 0x4f72, 0xae, 0xbb,0x7f, 0x94, 0x92, 0x76, 0x7b, 0xe1	**EFI_DEVICE_PATH_PROTOCOL –** USB Device Path Node.	The USB Device Path Node must be preceded by a PCI Device Path Node.
5.4.2.1.9	0x436486e1, 0x4426, 0x427f, 0xa5, 0xc5, 0x45, 0xf2, 0x13, 0xef, 0x15, 0x88	**EFI_DEVICE_PATH_PROTOCOL –** PCI Option ROM Device Path Node.	The PCI Option ROM Device Path Node must be preceded by a PCI Device Path Node
5.4.2.1.10	0x9619e2ad, 0x0358, 0x4aef, 0x98, 0x60, 0xb9, 0x08, 0xa3, 0xcc, 0x08, 0x7e	**EFI_DEVICE_PATH_PROTOCOL –** Device Path must be terminated.	The Device Path must be terminated by an End of Device Path node with a sub-type of End the Entire Device Path. A **NULL** Device Path consists of a single End Device Path Node. A Device Path that contains a **NULL** pointer and no Device Path structures is illegal.
5.4.2.1.11	0x59116d82, 0xaf34, 0x48a2, 0xaa, 0x22, 0xe4, 0x83, 0x7a, 0xd8, 0xe5, 0x8d	**EFI_DEVICE_PATH_PROTOCOL –** Controller Device Path Node.	The Controller Device Path Node must be preceded by a PCI Device Path Node.

6.3 Device Path Utilities Protocol Interface Function Test

6.3.1 CreatDeviceNode Functionality

Number	GUID	Assertion	Test Description
5.4.3.1.1	0x9831dfbb, 0x008e, 0x4b37, 0xb2, 0x3c, 0x76, 0x43, 0x7c, 0xa4, 0xee, 0x91	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL.** CreatDeviceNode - CreatDeviceNode() must set *Type*, *SubType* and *Length* correctly, return EFI_DEVICE_PATH.	1. Call **CreatDeviceNode()** with a *NodeType* value of 1, a *NodeSubType* value of 1, and a *NodeLength* value of 6. 2. The return **EFI_DEVICE_PATH** structure should have *Type*, *SubType* and *Length* values that are the same as the ones set in **CreatDeviceNode()**.
5.4.3.1.2	0xf7c1a5dd, 0x3683, 0x43a6, 0x8d, 0x90, 0x6b, 0x79, 0x12, 0xbd, 0x32, 0x1d	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL.** CreatDeviceNode - CreatDeviceNode() must set *Type*, *SubType* and *Length* correctly, return **EFI_DEVICE_PATH** (another case).	1. Call **CreatDeviceNode()** with a *NodeType* value of 2, a *NodeSubType* value of 1 and a *NodeLength* value of 12. 2. The return **EFI_DEVICE_PATH** structure should have *Type*, *SubType* and *Length* values the same as the ones set in **CreatDeviceNode()**.

6.3.2 AppendDeviceNode Functionality

Number	GUID	Assertion	Test Description
5.4.3.2.1	0x0deb01c9, 0x16db, 0x42ac, 0x99, 0x99, 0x27, 0x7b, 0x61, 0x96, 0xf4, 0xb8	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL. AppendDeviceNode - AppendDeviceNode()** called by the End of Device Path node must set *Type*, *SubType* and *Length* correctly in the first device path node, return **EFI_DEVICE_PATH** structure.	1. Call **CreatDeviceNode()** to create an End of Device Path node. 2. Call **CreatDeviceNode()** with a *NodeType* value of 2,a *NodeSubType* value of 1 and a **NodeLength** value of 12. 3. Call *AppendDeviceNode()* with a *DeviceNode* value of the return pointer of **CreatDeviceNode()**. 4. The first device path node in the return **EFI_DEVICE_PATH** structure should have *Type*, *SubType* and *Length* values the same as the ones set in CreatDeviceNode().

Number	**GUID**	**Assertion**	**Test Description**
5.4.3.2.2	0xc2fa4f0f, 0xd2f0, 0x44b1, 0xa8, 0x69, 0x04, 0xeb, 0xc8, 0x88, 0xa6, 0xb6	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL.AppendDeviceNode - AppendDeviceNode()** must set *Type*, *SubType* and *Length* correctly in the last but the End of Device Path node in the return **EFI_DEVICE_PATH** structure.	1. Call **CreatDeviceNode()**, **AppendDeviceNode()** repeatedly to create a new device path. 2. The last but the end-of-device-path node in the return **EFI_DEVICE_PATH** structure should have *Type*, *SubType* and *Length* values the same as set in the last **CreatDeviceNode()**.

6.3.3 GetDevicePathSize Functionality

Number	GUID	Assertion	Test Description
5.4.3.3.1	0x4257efa5, 0xd844, 0x4361, 0x98, 0xb9, 0x0d, 0x0e, 0x09, 0xf6, 0x8f, 0x78	`EFI_DEVICE_PATH_UTILITIES_PROTOCOL.` GetDevicePathSize - GetDevicePathSize`()` should return the correct value and the return status should increase after AppendDeviceNode() is called.	1. Call `CreatDeviceNode()` to create an End of Device Path node. 2. Call GetDevicePathSize(). 3. Call `AppendDeviceNode()` with a *`DeviceNode`* value of a return pointer of `CreatDeviceNode()`. 4. Call GetDevicePathSize() again. 5. The return status should be 4 after `GetDevicePathSize()` was called the first time. 6. The return status should show an increase of the new device path node's length after `GetDevicePathSize()` was called the second time.

6.3.4 DuplicateDevicePath Functionality

Number	GUID	Assertion	Test Description
5.4.3.4.1	0x065a0a89, 0x3594, 0x440e, 0x82, 0xe6, 0x9e, 0xaf, 0x74, 0xc7, 0xb7, 0x2f	`EFI_DEVICE_PATH_UTILITIES_PROTOCOL. DuplicateDevicePath - DuplicateDevicePath()` must correctly set the return `EFI_DEVICE_PATH` structure the same as the original one.	1. Call `CreatDeviceNode()`, `AppendDeviceNode()` repeatedly to create a new device path. 2. Call `GetDevicePathSize()` first. 3. Call `DupilicateDevicePath()`. 4. Call `GetDevicePathSize()` with a *`DevicePath`* value of the return value of `DupilicateDevicePath()`. 5. The return value of `GetDevicePathSize()` should keep the same as the first return value, and the two device paths should be identical.

6.3.5 DuplicateDevicePath Conformance

Number	GUID	Assertion	Test Description
5.4.3.2.6	0x97363972, 0x64cd, 0x4af8, 0xa7, 0x07, 0x41, 0x49, 0x81, 0xad, 0x4a, 0xb2	`EFI_DEVICE_PATH_UTILITIES_PROTOCOL. DuplicateDevicePath - DuplicateDevicePath()` should return `NULL` if *`DevicePath`* is `NULL`	1. Call `DupilicateDevicePath()` with a *`DevicePath`* value of `NULL`. 2. The return value should be `NULL.`

6.3.6 AppendDevicePath Functionality

Number	GUID	Assertion	Test Description
5.4.3.5.1	0x7da4d0e1, 0x2d1b, 0x4b60, 0xaa, 0xb2, 0xf3, 0xc1, 0x35, 0xf1, 0xf3, 0x21	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL.AppendDevicePath - AppendDevicePath()** must correctly set the return **EFI_DEVICE_PATH** structure as the new device path that appends the second device path to the first.	1. Call **CreatDeviceNode()**, **AppendDeviceNode()** repeatedly to create a new device path. 2. Call **CreatDeviceNode()**, **AppendDeviceNode()** repeatedly to create another device path. 3. Call **AppendDevicePath()** with *Src1* and *Src2* set respectively. 4. Call **GetDevicePathSize()** with a *DevicePath* value of the return value of **AppendDevicePath ()**. 5. The return value of **GetDevicePathSize()** should show an increase of the new device path's length with the size of *Src1*'s End of Device Path device node subtracted after **GetDevicePathSize()** is called the second time.

6.3.7 AppendDevicePathInstance Functionality

Number	GUID	Assertion	Test Description
5.4.3.6.1	0x8d72d028, 0x1e92, 0x4a79, 0x8d, 0xbe, 0xab, 0xc9, 0x3a, 0x47, 0xed, 0xee	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL.AppendDevicePathInstance - AppendDevicePathInstance()** must correctly set the return **EFI_DEVICE_PATH** structure as the new device path that appends the specific device path instance to the specific device path.	1. Call **CreatDeviceNode()**, **AppendDeviceNode()** repeatedly to create a new device path. 2. Call **CreatDeviceNode()**, **AppendDeviceNode()** repeatedly to create another device path as a new device path instance. 3. Call **AppendDevicePathInstance()** with a *DevicePathInstance* value of the new device path instance. 4. The last device path instance of the returned **EFI_DEVICE_PATH** structure should be the same as the newly created one.

6.3.8 GetNextDevicePathInstance Functionality

Number	GUID	Assertion	Test Description
5.4.3.7.1	0x4c914601, 0x681c, 0x48e5, 0xbe, 0xbd, 0x72, 0xdf, 0xfb, 0x1b, 0x42, 0x63	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL.GetNextDevicePathInstance - GetNextDevicePathInstance()** must get the next device path instance and return a pointer to the copy of the current device path instance.	1. Call **CreatDeviceNode()**, **AppendDeviceNode()** repeatedly to create a new device path. 2. Call **CreatDeviceNode()**, **AppendDeviceNode()** repeatedly to create another device path as a new device path instance. 3. Call **AppendDevicePathInstance()** with a *DevicePathInstance* value of the new device path instance. 4. Call **GetNextDevicePathInstance()**. 5. The return **EFI_DEVICE_PATH** structure should include a device path instance the same as the first instance of the new device path and *DevicePathInstanceSize* should become the size of the first instance, and at the same time, the *DevicePathInstance* should point to the second instance.

6.3.9 IsDevicePathMultiInstance Functionality

Number	GUID	Assertion	Test Description
5.4.3.8.1	0x2e9e1bb4, 0x5e2f, 0x4a26, 0xbb, 0x16, 0xf8, 0x0f, 0xf8, 0xdf, 0x6c, 0xdd	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL.IsDevicePathMultiInstance - IsDevicePathMultiInstance()** must judge whether a device path is a multi-instance.	1. Call **CreatDeviceNode()** to create an End of Device Path node. 2. Call **IsDevicePathMultiInstance()**. 3. Call **CreatDeviceNode()**, **AppendDeviceNode()** repeatedly to create a new device path that includes only one device path instance. 4. Call **IsDevicePathMultiInstance()**. 5. Call **AppendDevicePathInstance()** with a *DevicePathInstance* value of a new device path instance. 6. Call IsDevicePathMultiInstance(). 7. The return values of *IsDevicePathMultiInstance* should be **FALSE**, **FALSE** and **TRUE** respectively.

6.4 Device Path Utilities Protocol Interface Conformance Test

6.4.1 CreatDeviceNode Conformance

Number	GUID	Assertion	Test Description
5.4.4.1.1	0x44a2c284, 0xb019, 0x441b, 0x9e, 0xe0, 0x15, 0x14, 0x96, 0x51, 0xc8, 0x1f	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL. CreatDeviceNode** - **CreatDeviceNode()** should fail with an invalid *NodeLength* value	1. Call **CreatDeviceNode()** with a *NodeLength* value of 3. 2. The return pointer should be **NULL**.

6.4.2 AppendDeviceNode Conformance

Number	GUID	Assertion	Test Description
5.4.4.2.1	0x795510e5, 0xdd0e, 0x403e, 0xa3, 0x4c, 0x67, 0x64, 0x2f, 0xe6, 0x2b, 0x46	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL. AppendDeviceNode** - **AppendDeviceNode()** should return the copy of *DeviceNode* with a *DevicePath* value of **NULL**	1. Call **CreatDeviceNode()** with a *NodeType* value of 1, a *NodeSubType* value of 1 and a *NodeLength* value of 6. 2. Call **AppendDeviceNode()** with DevicePath value of **NULL** and a DeviceNode value of the return pointer of **CreatDeviceNode()**. 3. The return pointer should return the copy of the *DeviceNode* parameter .
5.4.4.2.2	0x54f1f4cc, 0xa193, 0x4023, 0xa1, 0x68, 0x96, 0x9a, 0xa8, 0x2d, 0xdd, 0x13	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL. AppendDeviceNode** - **AppendDeviceNode()** should return the copy of *DevicePath* with *DeviceNode* set to **NULL**	4. Call **CreatDeviceNode()** to create an End of Device Path node. 5. Call **AppendDeviceNode()** with a *DeviceNode* value of **NULL**. 6. The return should be the copy of *DevicePath*.
5.4.4.2.3	0xbb6ae1b8, 0xb420, 0x4f94, 0xb7, 0x88, 0xc4, 0xcc, 0x3a, 0xda, 0x53, 0x05	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL. AppendDeviceNode** - **AppendDeviceNode()** should return end-of-device-path device node if both *DevicePath* and *DeviceNode* are **NULL**	1. Call **CreatDeviceNode()**, *AppendDeviceNode* with both *DevicePath* and *DeviceNode* are **NULL** 2. The return **EFI_DEVICE_PATH_PROTOCOL** structure should be end-of-device-path device node.

6.4.3 AppendDevicePath Conformance

Number	GUID	Assertion	Test Description
5.4.4.3.1	0xba53eab4, 0xa3b2, 0x4ed3, 0xae, 0x7e, 0x77, 0xa3, 0x6a, 0x86, 0x1d, 0xb0	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL.AppendDevicePath** - **AppendDevicePath()** should ignore *Src1* when it is set to **NULL**.	1. Call **CreatDeviceNode()** **AppendDeviceNod()** repeatedly to create a new device path. 2. Call **GetDevicePathSize()**. 3. Call **AppendDeviceNode()** with a *Src1* value of **NULL** and a valid *Src2* value. 4. Call **GetDevicePathSize()** with a *DevicePath* value of the return value of **AppendDeviceNode()**. 5. The return value of **GetDevicePathSize()** should be the same as the first return value.
5.4.4.3.2	0x49fbe4f2, 0xb963, 0x4a01, 0xbb, 0xd0, 0xc2, 0x9d, 0x11, 0x17, 0x4f, 0x6d	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL.AppendDevicePath** - **AppendDevicePath()** should ignore *Src2* when it is set to **NULL**.	1. Call **CreatDeviceNode()** **AppendDeviceNode()** repeatedly to create a new device path. 2. Call **GetDevicePathSize()**. 3. Call **AppendDeviceNode()** with a valid *Src1* value and a *Src2* value of **NULL**. 4. Call **GetDevicePathSize()** with a *DevicePath* value of the return value of **AppendDeviceNode()**. 5. The return value of **GetDevicePathSize()** should be the same as the first return value.
5.4.4.3.3	0x546bd0e4, 0xd288, 0x461f, 0x8a, 0xac, 0x67, 0x75, 0xc6, 0x96, 0x83, 0xe4	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL.AppendDeviceNode** - **AppendDeviceNode()** should return end-of-device-path if both *Src1* and *Src2* are **NULL**	1. Call **CreatDeviceNode()**, **AppendDeviceNode** with both *Src1* and *Src2* are **NULL** 2. The return **EFI_DEVICE_PATH** structure should be end-of-device-path.

6.4.4 AppendDevicePathInstance Conformance

Number	GUID	Assertion	Test Description
5.4.4.4.1	0xfe34dfb2, 0x7b8d, 0x42c7, 0x8a, 0x8a, 0x00, 0xea, 0x1b, 0xe6, 0xe5, 0x44	**EFI_DEVICE_PATH_UTILITIES_PROTOCOL.AppendDevicePathInstance** - **AppendDevicePathInstance()** should fail with a *DevicePathInstance* value of **NULL**.	1. Call **CreatDeviceNode()** with a *DevicePathInstance* value of **NULL**. 2. The return pointer should be **NULL**.

6.4.5 GetNextDevicePathInstance Conformance

Number	GUID	Assertion	Test Description
5.4.4.5.1	0x25acf6b7, 0xd5c8, 0x4fb0, 0xa6, 0x89, 0xaf, 0x8c, 0x03, 0x4e, 0x5e, 0xdc	`EFI_DEVICE_PATH_UTILITIES_PROTOCOL.` GetNextDevicePathInstance - GetNextDevicePathInstance`()` should fail with *DevicePathInstance* set to `NULL`.	1. Call `GetNextDevicePathInstance()` with a *`DevicePathInstance`* value of `NULL`. 2. The return pointer should be `NULL`.

6.5 Device Path To Text Protocol Interface Function Test

6.5.1 ConvertDeviceNodeToText Functionality

Number	GUID	Assertion	Test Description
5.4.5.1.1	0x68d2e9f6, 0xb5f0, 0x4660, 0xbd, 0xf7, 0x74, 0x97, 0x43, 0xce, 0xb1, 0xb4	`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL`. `ConvertDeviceNodeToText` - `ConvertDeviceNodeToText()` must set a string to describe the device node structure.	1. Call `CreatDeviceNode()` and set the values of this device path node's specific device path data to create a device path node of PCI Root Device Path. 2. Call `ConvertDeviceNodeToText()` with a *`DisplayOnly`* value of `FALSE` and a *`AllowShortcuts`* value of `TRUE` and `FALSE` respectively. 3. The return string should be the same as the expected one.
5.4.5.1.2	0x09a4021d, 0x2804, 0x49fa, 0x82, 0x95, 0x30, 0xb1, 0xcf, 0x27, 0xf7, 0x88	`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL`. `ConvertDeviceNodeToText -` `ConvertDeviceNodeToText()` must set a string to describe the device node structure.	4. Call `CreatDeviceNode()` and set the values of this device path node's specific device path data to create a device path node of PCI Device Path. 5. Call `ConvertDeviceNodeToText()` with a *DisplayOnly* value of `FALSE` and a *`AllowShortcuts`* value of `TRUE` and `FALSE` respectively. 6. The return string should be the same as the expected one.
5.4.5.1.3	0x97deff32, 0xa4d0, 0x4909, 0xa7, 0xfa, 0x98, 0xcf, 0x3e, 0xcf, 0xf5, 0xf0	`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.` `ConvertDeviceNodeToText` - `ConvertDeviceNodeToText()` must set a string to describe the device node structure.	7. Call `CreatDeviceNode()` and set the values of this device path node's specific device path data to create a device path node of ATAPI Device Path. 8. Call `ConvertDeviceNodeToText()` with a *`DisplayOnly`* value of `FALSE` and a *`AllowShortcuts`* value of `TRUE` and `FALSE` respectively. 9. The return string should be the same as the expected one.

6.5.2 ConvertDevicePathToText Functionality

Number	GUID	Assertion	Test Description
5.4.5.2.1	0x11993701, 0x534b, 0x4804, 0xb9, 0x17, 0x72, 0x6b, 0xc9, 0x57, 0x43, 0x13	**EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertDevicePathToText - ConvertDevicePathToText()** must set a string to describe the device path structure.	1. Call **CreatDeviceNode()**, **AppendDeviceNode()** and **AppendDevicePathInstance()** repeatedly to create a legacy floppy device path. 2. Call **ConvertDevicePathToText()** with a *DisplayOnly* value of **FALSE** and a *AllowShortcuts* value of **TRUE** and **FALSE** respectively. 3. The return string should be the same as the expected one.
5.4.5.2.2	0xdb90a554, 0xc75f, 0x409e, 0x9d, 0x40, 0xcc, 0xcd, 0x6a, 0xc6, 0xd0, 0x57	**EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertDevicePathToText - ConvertDevicePathToText()** must set a string to describe the device path structure.	1. Call **CreatDeviceNode()**, **AppendDeviceNode()** and **AppendDevicePathInstance()** repeatedly to create an IDE disk device path. 2. Call **ConvertDevicePathToText()** with a *DisplayOnly* value of **FALSE** and a *AllowShortcuts* value of **TRUE** and **FALSE** respectively. 3. The return string should be the same as the expected one.
5.4.5.2.3	0x532045b2, 0x8cb7, 0x4c27, 0x83, 0x72, 0xc2, 0x80, 0xe4, 0xe1, 0xf9, 0x29	**EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertDevicePathToText - ConvertDevicePathToText()** must set a string to describe the device path structure.	1. Call **CreatDeviceNode()**, **AppendDeviceNode()** and **AppendDevicePathInstance()** repeatedly to create a secondary root PCI bus with a PCI to PCI bridge device path. 2. Call **ConvertDevicePathToText()** with a *DisplayOnly* value of **FALSE** and a *AllowShortcuts* value of **TRUE** and **FALSE** respectively. 3. The return string should be the same as the expected one.

6.6 Device Path To Text Protocol Interface Conformance Test

6.6.1 ConvertDeviceNodeToText Conformance

Number	GUID	Assertion	Test Description
5.4.6.1.1	0x945a93f7, 0xedac, 0x4893, 0xb2, 0xd2, 0x84, 0x0c, 0x39, 0xbb, 0x78, 0x24	**`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText()`** should return **`NULL`** with *`DeviceNode`* set to **`NULL`**.	1. Call **`ConvertDeviceNodeToText ()`** with a *`DeviceNode`* value of **`NULL`**. 2. The return pointer should be **`NULL`**.

6.6.2 ConvertDevicePathToText Conformance

Number	GUID	Assertion	Test Description
5.4.6.2.1	0x2570911f, 0x1a08, 0x4f96, 0x92, 0xf5, 0x26, 0x7e, 0xc0, 0x8d, 0x75, 0xb0	**`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertDevicePathToText - ConvertDevicePathToText()`** should return **`NULL`** with *`DevicePath`* set to **`NULL`**.	1. Call **`ConvertDevicePathToText ()`** with a *`DevicePath`* value of **`NULL`**. 2. The return pointer should be **`NULL`**.

6.7 Device Path To Text Protocol Interface Coverage Test

6.7.1 ConvertDeviceNodeToText Coverage

Number	GUID	Assertion	Test Description
5.4.7.1.1	0xca28d9a9, 0x6159, 0x4b70, 0xb5, 0xa0, 0x6f, 0xb3, 0x68, 0x63, 0x02, 0xd2	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe PcCard device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original.
5.4.7.1.2	0x203b6963, 0x5013, 0x4683, 0x95, 0x8b, 0xd4, 0xa2, 0x1c, 0xcc, 0xbb, 0x8d	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Memory Mapped device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.3	0xc05c7ebe, 0x69a4, 0x4fcc, 0xb8, 0x29, 0x25, 0x77, 0x54, 0xf3, 0xb4, 0x3e	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Vendor defined device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.4	0x36de850b, 0xb28d, 0x4bfd, 0x9e, 0xff, 0xbc, 0xd8, 0x05, 0xa4, 0xa2, 0xf3	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Controller device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.7.1.5	0xa20c1075, 0x9bde, 0x42db, 0x83, 0x28, 0x62, 0x6a, 0x18, 0xe6, 0x07, 0x9e	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the ACPI Expended device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.6	0xd448b8f6, 0x2d7e, 0x473d, 0xae, 0x66, 0x9e, 0xc7, 0xba, 0xa7, 0xf9, 0x9c	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a shortcut form of text string to describe the ACPI Expended device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.7	0xc4ef8ea1, 0x6fa7, 0x4e49, 0xa1, 0x7a, 0x30, 0xa0, 0xed, 0xd2, 0x3c, 0x6b	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText()` must recover the conversion `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the SCSI device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.8	0xac5859c4, 0x99a9, 0x43bc, 0xbd, 0x20, 0x76, 0xd4, 0x36, 0xa8, 0xf9, 0x71	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Fibre Channel device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.7.1.9	0xd00934b4, 0x846e, 0x4f8b, 0xa6, 0xc9, 0x13, 0xb, 0x19, 0x13, 0x49, 0x3c	`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the AcpiAdr device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.10	0xe49fdcdb, 0xbadb, 0x48c7, 0xbe, 0x8b, 0xbc, 0xce, 0x19, 0x0f, 0x2b, 0x79	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the USB device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.11	0xb21543cc, 0x4090, 0x4e28, 0x88, 0xc5, 0x5b, 0xd6, 0x29, 0x17, 0x7b, 0xd9	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the I2O device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.12	0x4bf7bbff, 0x783f, 0x4ab0, 0xb5, 0x2a, 0x3e, 0xab, 0x1d, 0x6e, 0xdd, 0x02	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Infiniband device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.7.1.13	0xd7a537b7, 0x96a2, 0x478d, 0xa2, 0xd3, 0x67, 0xca, 0x68, 0x93, 0x8e, 0xe2	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the PC-ANSI device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.14	0xeaba3b8d, 0x0aad, 0x4729, 0xb0, 0x2e, 0xb6, 0xa4, 0x89, 0xdc, 0x17, 0x4d	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the *`UartFlowCtrl`* device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.15	0xd751aa0e, 0xb0ea, 0x43ee, 0x89, 0x65, 0x5, 0x4c, 0x97, 0x1, 0xa, 0x32	`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the AcpiExp device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.16	0x51a639b6, 0x878d, 0x4118, 0x88, 0x6b, 0x15, 0x4f, 0x84, 0x5e, 0xfd, 0xfd	`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the PciRoot device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.7.1.17	0xe23c5141, 0xac77, 0x42f4, 0xb4, 0x18, 0x9e, 0xd3, 0x76, 0xbc, 0xcf, 0xd7	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the MAC device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.7.1.18	0x77ebce11, 0x3621, 0x4900, 0xbd, 0xb2, 0x95, 0x01, 0x2a, 0xcd, 0xca, 0x46	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the IPv4 device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.7.1.19	0xef32be73, 0xf5b7, 0x4545, 0xaf, 0xd7, 0x5e, 0xfb, 0xdc, 0x01, 0x8f, 0x16	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the IPv6 device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.7.1.20	0xbdf0860e, 0x12b6, 0x4c2a, 0xa2, 0x6c, 0x8e, 0x25, 0x87, 0x99, 0xa8, 0xd6	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the UART device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.7.1.21	0x340f6746, 0x662f, 0x4613, 0x89, 0x5a, 0x16, 0x57, 0x7d, 0xe0, 0x76, 0x99	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the USB Class device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.22	0x16001709, 0x687d, 0x4880, 0x89, 0xc4, 0x1c, 0x63, 0x1e, 0xb5, 0x2e, 0x2d	`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the PcieRoot device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.23	0xf375ad05, 0xd5ae, 0x408f, 0x8a, 0xa5, 0x21, 0xb8, 0xd1, 0xe9, 0xfd, 0x75	`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Floppy device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.24	0xa4c0ed2e, 0x1438, 0x44cc, 0x97, 0x10, 0x1e, 0x2e, 0x29, 0xe3, 0xbd, 0xe6	`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Keyboard device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.7.1.25	0x2ccd0cbb, 0x395f, 0x4b76, 0x8a, 0xe8, 0x3f, 0x4a, 0x07, 0x98, 0x4f, 0x3a	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Logical Unit device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.26	0x13625cd7, 0x79d1, 0x4f0b, 0x80, 0xe0, 0xb5, 0x54, 0x94, 0xae, 0xc6, 0xb6	`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Serial device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.27	0x2001ae80, 0x7309, 0x4b70, 0x9f, 0x4e, 0x7b, 0xad, 0x66, 0x9d, 0xc0, 0x43	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Hard Drive with GUID device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.28	0xf37b8ee5, 0xfb01, 0x41e3, 0xa2, 0x6a, 0xa1, 0x99, 0xd9, 0x59, 0x24, 0x74	`EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Parallel Port device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.7.1.29	0xe171c43f, 0x9aaf, 0x4133, 0x95, 0x80, 0xfb, 0xb5, 0xa7, 0x0b, 0x88, 0x72	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the CD-ROM device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as theoriginal one.
5.4.7.1.30	0x596665ca, 0x74e6, 0x4f6e, 0x88, 0xd8, 0x6e, 0x26, 0xe5, 0x3a, 0x42, 0xab	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the FibreEx device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.31	0x5b136106, 0xcee0, 0x46d9, 0x87, 0xa9, 0x68, 0x1d, 0x70, 0xf7, 0x1f, 0x17	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Media device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.7.1.32	0xdb0e6e8b, 0x1d57, 0x41e5, 0xb8, 0x74, 0x4c, 0xe8, 0x5a, 0xd5, 0x76, 0x4c	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText()` must recover the conversion `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the SAS device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.7.1.33	0x44f98053, 0xbbf7, 0x4002, 0x9a, 0x7e, 0x6b, 0x4d, 0x37, 0x3e, 0x18, 0xff	EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the Media Relative Offset Range device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.7.1.34	0x44ed02e4, 0x48c7, 0x42df, 0xbe, 0x12, 0x60, 0xc1, 0xb2, 0x7f, 0xe8, 0xab	EFI_DEVICE_PATH_TO_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the Vlan device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.7.1.35	0x4e3dfefc, 0xeebb, 0x46d0, 0xa1, 0xc3, 0x83, 0xaa, 0x2, 0x6d, 0xf1, 0x1b	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the SASEx device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.7.1.36	0x21e74335, 0x50c9, 0x4deb, 0x8a, 0x9d, 0xf4, 0x2, 0x97, 0xfc, 0xa2, 0x26	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the NVMe device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.7.1.37	0x252df981, 0x416a, 0x486d, 0x8c, 0x78, 0xde, 0xae, 0x72, 0x4a, 0x68, 0xeb	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL. ConvertDeviceNodeToText - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string	1. Set a text string to describe the BMC device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.7.1.38	0x77cdae2c, 0x642c, 0x4113, 0xb6, 0x59, 0x25, 0x23, 0x42, 0xb1, 0x16, 0xb6	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL. ConvertDeviceNodeToText - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string	1. Set a text string to describe the RamDisk device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.7.1.39	0xd823b4b, 0x58b4, 0x4882, 0x9f, 0x38, 0xb, 0xfb, 0x3, 0xa0, 0x29, 0xa3	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL. ConvertDeviceNodeToText - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string	1. Set a text string to describe the Uri device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.7.1.40	0x4136553e, 0x8284, 0x409c, 0x90, 0x56, 0xcb, 0xbc, 0x91, 0xc5, 0xea, 0xa1	**EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText()** must recover the conversion that **ConvertTextToDeviceNode()** has performed on the device node string	1. Set a text string to describe the SD device path node. 2. Call **ConvertTextToDeviceNode()**. 3. Call **ConvertDeviceNodeToText()**. 4. The return string should be the same as the original one.
5.4.7.1.41	0x23bcd190, 0x10b4, 0x4063, 0x95, 0x2, 0xea, 0x5c, 0x14, 0xfc, 0x72, 0x1e	**EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText()** must recover the conversion that **ConvertTextToDeviceNode()** has performed on the device node string	1. Set a text string to describe the BlueTooth device path node. 2. Call **ConvertTextToDeviceNode()**. 3. Call **ConvertDeviceNodeToText()**. 4. The return string should be the same as the original one.
5.4.7.1.42	0x6faccc19, 0x7785, 0x49e6, 0xaf, 0x86, 0x9b, 0x5f, 0x69, 0x53, 0x60, 0x7d	**EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertDeviceNodeToText - ConvertDeviceNodeToText()** must recover the conversion that **ConvertTextToDeviceNode()** has performed on the device node string	1. Set a text string to describe the Wi-Fi device path node. 2. Call **ConvertTextToDeviceNode()**. 3. Call **ConvertDeviceNodeToText()**. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.7.1.43	0x60e2e2ac, 0xf5f9, 0x4ecf, 0xac, 0xb1, 0x79, 0xa1, 0xe5, 0xcc, 0xbc, 0xf6	**EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL. ConvertDeviceNodeToText - ConvertDeviceNodeToText()** must recover the conversion that **ConvertTextToDeviceNode()** has performed on the device node string	1. Set a text string to describe the eMMC device path node. 2. Call **ConvertTextToDeviceNode()**. 3. Call **ConvertDeviceNodeToText()**. 4. The return string should be the same as the original one.

6.7.2 ConvertDevicePathToText Coverage

Number	GUID	Assertion	Test Description
5.4.7.2.1	0x4af4f3cb, 0x4afa, 0x43b5, 0xb3, 0x83, 0x2e, 0x08, 0x57, 0x15, 0xf7, 0xa6	**EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL. ConvertDevicePathToText - ConvertDevicePathToText()** must recover the conversion that **ConvertTextToDevicePath()** has performed on the device node string.	1. Set a text string to describe a device path with multiple device path instances. 2. Call **ConvertTextToDevicePath()**. 3. Call **ConvertDevicePathToText()**. 4. The return string should be the same as the original one.

6.8 Device Path From Text Protocol Interface Function Test

6.8.1 ConvertTextToDeviceNode Functionality

Number	GUID	Assertion	Test Description
5.4.8.1.1	0x6ea38cc6, 0x6b02, 0x4ee7, 0x84, 0xcc, 0x37, 0xc0, 0x07, 0x55, 0xef, 0xa3	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertTextToDeviceNode()` must set a device node structure.	1. Set a text string to describe the PCI Root device path node. 2. Call `ConvertTextToDeviceNode()`. 3. The return structure should be the same as the expected one.
5.4.8.1.2	0xe025cd1b, 0xda51, 0x4496, 0xac, 0xa0, 0xf6, 0x18, 0x3e, 0x67, 0xb6, 0x78	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertTextToDeviceNode()` must set a device node structure.	1. Set a text string to describe the PCI device path node. 2. Call `ConvertTextToDeviceNode()`. 3. The return structure should be the same as the expected one.
5.4.8.1.3	0xe924b842, 0x2e27, 0x4d39, 0x98, 0x7d, 0x3a, 0x64, 0xd7, 0x45, 0x0e, 0xda	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertTextToDeviceNode()` must set a device node structure.	1. Set a text string to describe the ATAPI device path node. 2. Call `ConvertTextToDeviceNode()`. 3. The return structure should be the same as the expected one.

6.8.2 ConvertTextToDevicePath Functionality

Number	GUID	Assertion	Test Description
5.4.8.2.1	0xa2215ca2, 0x965a, 0x4ae3, 0xae, 0x58, 0xca, 0xd1, 0x20, 0xb3, 0xf5, 0x87	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDevicePath - ConvertTextToDevicePath()` must set a device node structure.	1. Set a text string to describe the legacy floppy device path. 2. Call `ConvertTextToDevicePath()`. 3. The return structure should be the same as the expected one.
5.4.8.2.2	0x34dcb77c, 0x782f, 0x429a, 0x92, 0xfc, 0xa0, 0x02, 0xae, 0xfb, 0xcb, 0xd7	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDevicePath - ConvertTextToDevicePath()` must set a device node structure.	1. Set a text string to describe the IDE disk device path. 2. Call `ConvertTextToDevicePath()`. 3. The return structure should be the same as the expected one.
5.4.8.2.3	0xbf4b5c33, 0x7cc4, 0x412b, 0xb6, 0x88, 0x14, 0x0a, 0x17, 0x3f, 0x4f, 0x5a	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDevicePath - ConvertTextToDevicePath()` must set a device node structure.	1. Set a text string to describe the secondary root PCI bus with a PCI to PCI bridge device path. 2. Call `ConvertTextToDevicePath()`. 3. The return structure should be the same as the expected one.

6.9 Device Path From Text Protocol Interface Conformance Test

6.9.1 ConvertTextToDeviceNode Conformance

Number	GUID	Assertion	Test Description
5.4.9.1.1	0x112d380b, 0x1f72, 0x41d4, 0xa3, 0x5a, 0xd3, 0x61, 0x72, 0xce, 0x42, 0x60	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertTextToDeviceNode()` should return `NULL` with *`TextDeviceNode`* set to `NULL`.	1. Call `ConvertTextToDeviceNode()` with a *`TextDeviceNode`* value of `NULL`. 2. The return pointer should be `NULL`.

6.9.2 ConvertTextToDevicePath Conformance

Number	GUID	Assertion	Test Description
5.4.9.2.1	0x6de40774, 0x269d, 0x4c52, 0x9e, 0xce, 0xe4, 0x01, 0x95, 0xc4, 0x09, 0xed	**`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL`**. **`ConvertTextToDevicePath`** - **`ConvertTextToDevicePath()`** should return **`NULL`** with *`TextDevicePath`* set to be **`NULL`**.	1. Call **`ConvertTextToDevicePath()`** with a *`TextDevicePath`* value of **`NULL`**. 2. The return pointer should be **`NULL`**.

6.10 Device Path From Text Protocol Interface Coverage Test

6.10.1 ConvertTextToDeviceNode Coverage

Number	GUID	Assertion	Test Description
5.4.10.1.1	0xabd4778e, 0xc1c5, 0x4dcb, 0xa5, 0x75, 0x4a, 0x2e, 0x83, 0x68, 0x01, 0x82	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe PcCard device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the one originally set.
5.4.10.1.2	0x384a0f7f, 0x3aed, 0x4942, 0xbf, 0x29, 0xed, 0x70, 0x7c, 0xb8, 0x96, 0xc3	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Memory Mapped device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.3	0x5ea2ddfd, 0xd264, 0x46d5, 0x99, 0x97, 0x17, 0xb2, 0x36, 0xe4, 0x46, 0xee	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Vendor defined device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original onet.
5.4.10.1.4	0xeeaad308, 0x9461, 0x42dc, 0x95, 0x2a, 0x25, 0xe3, 0xfb, 0x34, 0xc6, 0x4d	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Controller device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.10.1.5	0x5adc74cf, 0x0a05, 0x4689, 0xa0, 0xd0, 0xf3, 0x71, 0x10, 0x05, 0x24, 0xf4	**EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.** ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the ACPI Expended device path node. 1. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.10.1.6	0xac15c6df, 0x10f5, 0x40f1, 0x9e, 0xdc, 0x16, 0xa4, 0x22, 0x86, 0xe2, 0xae	**EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.** ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a shortcut form of text string to describe the ACPI Expended device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.10.1.7	0xd6769fb3, 0x6f40, 0x441e, 0xbc, 0x16, 0xdb, 0xab, 0xc5, 0x1f, 0xbc, 0x8e	**EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.** ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the SCSI device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.10.1.8	0x5a6105d4, 0x6c72, 0x4842, 0xbb, 0xf9, 0x16, 0xb4, 0x63, 0xc5, 0x65, 0x21	**EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.** ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the Fibre Channel device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.10.1.9	0x370abd68, 0xd84c, 0x4247, 0xbd, 0xbd, 0xb4, 0xbc, 0x2a, 0x1f, 0x74, 0x9d	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the 1394 device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.10.1.10	0x4b30ff6b, 0x0495, 0x4a88, 0x89, 0x24, 0xed, 0x47, 0xb4, 0x70, 0x3a, 0xea	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the USB device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.10.1.11	0x7c010d41, 0x940f, 0x4ab7, 0x99, 0xb3, 0x56, 0x29, 0xfe, 0xe2, 0xb3, 0xe8	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the I2O device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.10.1.12	0x3aff77da, 0x5f86, 0x4145, 0x84, 0xfa, 0x7e, 0x24, 0x64, 0x1a, 0xef, 0x67	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the Infiniband device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.10.1.13	0x57945d65, 0x2cd1, 0x44cb, 0x95, 0xa2, 0x85, 0x3d, 0x6b, 0x45, 0xc2, 0x10	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has peformed on the device node string.	1. Set a text string to describe the PC-ANSI device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.14	0x99fe3cd1, 0x9015, 0x4995, 0xb9, 0x6c, 0x03, 0x37, 0x1c, 0xc0, 0x26, 0xc5	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the UartFlowCtrl device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.15	0xbe92f84c, 0x3922, 0x426b, 0xa0, 0x2a, 0x1b, 0x1b, 0xeb, 0xf9, 0x9d, 0x7c	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the SAS device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.16	0x453b6f77, 0xd3bf, 0x4f23, 0x80, 0x35, 0x0f, 0x61, 0xdf, 0xe0, 0x16, 0xe1	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the DebugPort device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.10.1.17	0xdc026cfc, 0xc681, 0x43af, 0xb3, 0x73, 0xed, 0x8c, 0x1f, 0x7e, 0xaa, 0x6d	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the MAC device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original onet.
5.4.10.1.18	0x94dca74e, 0xacdd, 0x4fc2, 0xab, 0xb8, 0x48, 0xb1, 0x1b, 0xe0, 0x77, 0x57	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the IPv4 device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.19	0x447fabae, 0x7a70, 0x43df, 0x9f, 0x07, 0xc3, 0x07, 0x85, 0x24, 0x87, 0xd5	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the IPv6 device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.20	0xba0fc861, 0xd2ce, 0x4c70, 0x8b, 0xec, 0xaa, 0x89, 0xbc, 0x7d, 0x11, 0x0f	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe UART device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.10.1.21	0x2eba02bb, 0xa904, 0x4949, 0xa4, 0x6a, 0x41, 0x1f, 0xd8, 0xa8, 0xdd, 0xaf	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode – ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe USB Class device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as theoriginal one.
5.4.10.1.22	0x50cf1d50, 0xb560, 0x4a1a, 0x96, 0xc2, 0x01, 0x10, 0xf1, 0x25, 0xe3, 0x53	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode – ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the USB Video device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original onet.
5.4.10.1.23	0xd77e99e4, 0xe619, 0x4773, 0xa4, 0xa0, 0xbe, 0x55, 0x21, 0x4b, 0x01, 0xf0	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode – ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the UsbTest And Measurement device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original onet.
5.4.10.1.24	0xe5490e03, 0x83be, 0x4642, 0x98, 0xc5, 0x26, 0xae, 0x4f, 0xa4, 0x5d, 0xe4	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode – ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the AcpiAdr device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.10.1.25	0xe1042ce4, 0x760e, 0x433d, 0xb1, 0x7b, 0x9d, 0x02, 0x14, 0xf3, 0x2a, 0x12	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Logical Unit device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.26	0x1e3c0327, 0x7081, 0x4b7f, 0xab, 0xfa, 0xff, 0x01, 0xc2, 0x8c, 0xbe, 0x3f	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the iSCSI device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.27	0x37beed32, 0x165b, 0x480a, 0x91, 0x9b, 0xf5, 0xf2, 0x46, 0x07, 0xc7, 0x11	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Hard Drive with GUID device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.28	0x20884e00, 0x4471, 0x4e65, 0x84, 0xae, 0x51, 0x5d, 0x92, 0xc1, 0xe4, 0xf6	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Hard Drive with MBR device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.10.1.29	0xfdca47e4, 0x9965, 0x41dc, 0xbb, 0x01, 0x19, 0x10, 0x54, 0x41, 0x69, 0x60	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the covnersion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the CD-ROM device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.30	0xa0fc2a05, 0x01e1, 0x4a96, 0xb8, 0x8d, 0xa7, 0x73, 0x33, 0x25, 0xaf, 0x6e	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the File Path device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.31	0x0a0fc261, 0x193b, 0x4136, 0x82, 0xe3, 0x41, 0x32, 0x62, 0x36, 0xc6, 0x10	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Media device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.32	0xb59ff699, 0x4dc5, 0x45b8, 0x8b, 0xe6, 0x25, 0x36, 0x2e, 0xda, 0x59, 0xf3	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the BBS path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.10.1.33	0x2379a6e4, 0x3b61, 0x471c, 0x87, 0xb9, 0xff, 0xe6, 0x6a, 0x98, 0x79, 0x13	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Media Relative Offset Range device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.34	0x74f16d4f, 0xcbc4, 0x42f0, 0x99, 0x16, 0xae, 0x35, 0xa6, 0xd7, 0x5e, 0xb7	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Vlan device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.35	0xa6a5af57, 0xca9b, 0x42c1, 0x9b, 0xcd, 0xe3, 0xdb, 0xdf, 0x2, 0xf3, 0x8b	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the PciRoot device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.36	0x850d81ee, 0xe3d5, 0x468f, 0x83, 0x80, 0x25, 0x3e, 0xcb, 0xeb, 0xf2, 0x07	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the PcieRoot device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.10.1.37	0x1f72c17d, 0x9f1a, 0x4f57, 0xac, 0xb5, 0x2b, 0xfb, 0x3d, 0xe, 0x5b, 0x67	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode – ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Floppy device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.38	0x64dbbe77, 0x819e, 0x4cd9, 0x90, 0x88, 0xd9, 0x3d, 0x8f, 0x99, 0x9, 0x33	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode – ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Keyboard device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.39	0x62970cad, 0xb9ae, 0x459e, 0x94, 0xc7, 0x97, 0x37, 0x3, 0xc5, 0xda, 0x43	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode – ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Serial device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.
5.4.10.1.40	0x2c0e3e0c, 0x28f4, 0x4284, 0xbb, 0x54, 0x4, 0x2b, 0x6b, 0x26, 0xd3, 0x4e	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode – ConvertDeviceNodeToText()` must recover the conversion that `ConvertTextToDeviceNode()` has performed on the device node string.	1. Set a text string to describe the Parallel Port device path node. 2. Call `ConvertTextToDeviceNode()`. 3. Call `ConvertDeviceNodeToText()`. 4. The return string should be the same as the original one.

Number	GUID	Assertion	Test Description
5.4.10.1.41	0x826c2efe, 0xc377, 0x4594, 0x99, 0x42, 0xe1, 0xef, 0x07, 0x5d, 0xd1, 0x2f	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the FIbreEx device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.10.1.42	0xad957706, 0xb29a, 0x4184, 0xb8, 0x42, 0xf6, 0xf1, 0xa4, 0xe0, 0x57, 0x9b	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the SasEx device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.10.1.43	0x5fda2be2, 0x242a, 0x4c81, 0xa9, 0x7c, 0xfb, 0x2e, 0xe9, 0x94, 0x14, 0xf6	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the NVM express device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one.
5.4.10.1.44	0x6bc6e55b, 0xaa2c, 0x4853, 0x88, 0xbd, 0x7e, 0x79, 0xc8, 0xd3, 0xae, 0x58	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the BMC device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one

Number	GUID	Assertion	Test Description
5.4.10.1.45	0x177fd920, 0xb733, 0x4841, 0x9a, 0x10, 0xdb, 0x7b, 0x37, 0x4b, 0x47, 0x7c	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the UFS device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one
5.4.10.1.46	0x84e9f8, 0x6b65, 0x48e1, 0x92, 0x32, 0x4, 0x6e, 0xb4, 0x56, 0xd1, 0xe3	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the SD device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one
5.4.10.1.47	0x25c2071e, 0xedc, 0x403f, 0x89, 0x4a, 0xa4, 0x84, 0x25, 0xcc, 0xca, 0x80	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the Bluetooth device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one
5.4.10.1.48	0x84a73ccc, 0x2468, 0x440a, 0x93, 0xa1, 0xe2, 0x37, 0x35, 0xe5, 0x9f, 0x66	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the Wi-Fi device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one

Number	GUID	Assertion	Test Description
5.4.10.1.49	0x671ecea, 0x309c, 0x4398, 0x8c, 0x1, 0xed, 0x15, 0x37, 0xed, 0xaa, 0x40	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the RamDisk device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one
5.4.10.1.50	0x7e00edfb, 0x4ef8, 0x45da, 0x9e, 0x54, 0x8e, 0xf, 0x1b, 0xa5, 0xc3, 0xde	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the Uri device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one
5.4.10.1.51	0x882a6001, 0xae82, 0x4bb5, 0x83, 0xd, 0x6c, 0x2a, 0xd7, 0x68, 0x44, 0xec	EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDeviceNode - ConvertDeviceNodeToText() must recover the conversion that ConvertTextToDeviceNode() has performed on the device node string.	1. Set a text string to describe the eMMC device path node. 2. Call ConvertTextToDeviceNode(). 3. Call ConvertDeviceNodeToText(). 4. The return string should be the same as the original one

6.10.2 ConvertTextToDevicePath Coverage

Number	GUID	Assertion	Test Description
5.4.10.2.1	0x1759828d, 0x3377, 0x4473, 0x84, 0x8a, 0x1a, 0x92, 0x6f, 0x2e, 0x5b, 0xc5	`EFI_DEVICE_PATH_FROM_TEXT_PROTOCOL.ConvertTextToDevicePath - ConvertDevicePathToText()` must recover the conversion that `ConvertTextToDevicePath()` has performed on the device node string.	1. Set a text string to describe a device path with multiple device path instances. 2. Call `ConvertTextToDevicePath()`. 3. Call `ConvertDevicePathToText()`. 4. The return string should be the same as the original one.

7 Protocols EFI Driver Model Test

7.1 EFI_DRIVER_BINDING_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_DRIVER_BINDING_PROTOCOL Section.

This test will change the system data during testing. It is not included in the EFI SCT.

7.2 EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL Section.

7.2.1 GetDriver()

Number	GUID	Assertion	Test Description
5.5.2.1.1	0x013a1d94, 0x42ec, 0x429c, 0xb4, 0x99, 0x9d, 0x67, 0x5c, 0xea, 0x32, 0xe2	**EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.GetDriver** - Invokes **GetDriver()** with invalid *ControllerHandle.*	Call **GetDriver()** with invalid *ControllerHandle*. It should return **EFI_INVALID_PARAMETER**.
5.5.2.1.2	0xec346531, 0x5125, 0x4e5f, 0x93, 0xa9, 0x7a, 0x7a, 0xed, 0xc0, 0xe3, 0xb9	**EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.GetDriver** - Invokes **GetDriver()** with invalid *DriverImageHandle*	Call **GetDriver()** with invalid *DriverImagePath*. It should return **EFI_INVALID_PARAMETER**.
5.5.2.1.3	0xb6ce6934, 0xae1d, 0x41be, 0xba, 0x01, 0xac, 0x73, 0x49, 0x70, 0xe0, 0xb5	**EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.GetDriver** - Invokes **GetDriver()** and verify interface correctness within test case	Call **GetDriver()** with *DriverImageHandle* is **NULL**. If the return status is **EFI_SUCCESS**, get the next image handle till the end. The return status should be **EFI_SUCCESS**, except the last one. The last one should be **EFI_NOT_FOUND**.

Number	GUID	Assertion	Test Description
5.5.2.1.4	0xf8e30f06, 0x98b8, 0x4aba, 0xa0, 0x73, 0x67, 0x69, 0x33, 0xc0, 0xf8, 0x81	**EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.GetDriver** - Invokes **GetDriver()** and verify whether the image handle is installed.	Call **GetDriverPath()** to get the valid *DevicePath*.Call **LoadImage()** to get the *DriverImageHandle*. Use this *DevicePath* and *DriverImageHandle* to call **DriverLoaded()**.Call **GetDriver()**.The Image Handle got by the **GetDriver()** should be same as the former handle which is got by **LoadImage()**.The new *DriverImageHandle* should be same as the before one.

7.2.2 GetDriverPath()

Number	GUID	Assertion	Test Description
5.5.2.2.1	0x47008c31, 0xe877, 0x4acf, 0x88, 0x7a, 0xd5, 0x56, 0xd4, 0xb1, 0xd5, 0xe3	**EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.GetDriverPath** - Invokes **GetDriverPath()** with invalid *ControllerHandle.*	Call **GetDriverPath()** with invalid *ControllerHandle*. Return status should be **EFI_INVALID_PARAMETER**.
5.5.2.2.2	0xbb8d1b45, 0xe187, 0x4195, 0xa9, 0xdc, 0xdb, 0xc7, 0x5e, 0xef, 0x99, 0x92	**EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.GetDriverPath** - Invokes **GetDriverPath()** with invalid *DriverImageHandle*	Call **GetDriverPath()** with invalid *DriverImagePath*. Return status should be **EFI_INVALID_PARAMETER**.
5.5.2.2.3	0xe0434e5d, 0xa452, 0x4ef6, 0xb3, 0x90, 0xba, 0x12, 0x2a, 0xbb, 0xa8, 0xa8	**EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.GetDriverPath** - Invokes **GetDriverPath()** and verify interface correctness within test case	Call **GetDriverPath()** with *DriverImagePath* is **NULL**.If the return status is **EFI_SUCCESS**, get the next image handle till the end. The return status should be **EFI_SUCCESS**, except the last one. The last one should be **EFI_NOT_FOUND**.

7.2.3 DriverLoaded()

Number	GUID	Assertion	Test Description
5.5.2.3.1	0x7bad1b57, 0xc99c, 0x48c0, 0xb5, 0x28, 0x0b, 0x86, 0x0e, 0xfc, 0x27, 0xc3	`EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.DriverLoaded` - Invokes `DriverLoaded()` and verify interface correctness within test case	Call `GetDriverPath()` to get the valid *`DevicePath`*.Call `LoadImage()` to get the Driver Image Handle.Use this *`DevicePath`* and Driver Image Handle to call `DriverLoaded()`.The return status should be `EFI_SUCCESS`.
5.5.2.3.2	0x4d764ca3, 0x4d43, 0x4a89, 0x93, 0x4b, 0x8f, 0x60, 0x9e, 0xca, 0x82, 0x4d	`EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.DriverLoaded` - Invokes `DriverLoaded()` with *`DriverImagePath`* not gotten from the prior call to `GetDriverPath()`.	Call `DriverLoaded()` with *`DriverImagePath`* is not a device path that was returned on a prior call to `GetDriverPath()` for the controller specified by *`ControllerHandle`*. Return status should be `EFI_NOT_FOUND`.
5.5.2.3.3	0x745042f7, 0xa9e8, 0x436b, 0x8c, 0x44, 0x42, 0x49, 0x07, 0x90, 0x68, 0x50	`EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.DriverLoaded` - Invokes `DriverLoaded()` with invalid *`ControllerHandle`*	Call `DriverLoaded()` with invalid *`ControllerHandle`* .The return status should be `EFI_INVALID_PARAMETER`.
5.5.2.3.4	0xecc09588, 0xb786, 0x49b1, 0x93, 0x7f, 0x8e, 0xed, 0x89, 0xa7, 0x52, 0xd6	`EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.DriverLoaded` - Invokes `DriverLoaded()` with invalid *`DriverImagePath`*.	Call `DriverLoaded()` with invalid *`DriverImagePath`* .The return status should be `EFI_INVALID_PARAMETER`.
5.5.2.3.5	0xf5d05588, 0x0d6a, 0x40fa, 0xa9, 0x54, 0x4b, 0x40, 0xd7, 0x9b, 0x4e, 0x5b	`EFI_PLATFORM_DRIVER_OVERRIDE_PROTOCOL.DriverLoaded` - Invokes `DriverLoaded()` with invalid *`DriverImageHandle`*	Call `DriverLoaded()` with invalid *`DriverImageHandle`*. The return status should be `EFI_INVALID_PARAMETER`.

7.3 EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL Section.

7.3.1 GetDriver()

Number	GUID	Assertion	Test Description
5.5.3.1.1	0x18a52d36, 0xd149, 0x414c, 0xa8, 0xc9, 0x43, 0xc8, 0x55, 0x71, 0xc6, 0x5f	**EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL.GetDriver – GetDriver** returns **EFI_SUCCESS** with valid *DriverImageHandle*	1. Circularly call **GetDriver()** with *DriverImageHandle* retrieved by the last call of **GetDriver()**, until the end of the list of override drivers is reached. Expected Behavior: The return status of each valid *DriverImageHandle* must be **EFI_SUCCESS**.
5.5.3.1.2	0x841a7b86, 0xabf0, 0x40af, 0x92, 0x67, 0x3f, 0xb3, 0x69, 0x2f, 0xc0, 0x37	**EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL.GetDriver – GetDriver** returns **EFI_NOT_FOUND** with unsupported Parameters	1. Circularly call **GetDriver()** with *DriverImageHandle* retrieved by the last call of **GetDriver()**, until the end of the list of override drivers is reached. Expected Behavior: The last return status must be **EFI_NOT_FOUND**.
5.5.3.1.3	0x2f0b7eb4, 0xb6b4, 0x4a58, 0x87, 0x55, 0x93, 0x52, 0xd4, 0x7e, 0x27, 0xef	**EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL.GetDriver – GetDriver ()** returns **EFI_INVALID_PARAMETER** with invalid *DriverImageHandle*	1. Pass the invalid *DriverImageHandle* to the function Expected Behavior: The return status must be **EFI_INVALID_PARAMETER**.

7.4 EFI_DRIVER_CONFIGURATION_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_BUS_SPECIFIC_DRIVER_OVERRIDE_PROTOCOL Section.

7.4.1 SetOptions()

Number	GUID	Assertion	Test Description
5.5.4.1.1	0x82d78ef0, 0x0e7c, 0x4338, 0xb0, 0xe6, 0xef, 0x07, 0x01, 0x35, 0x18, 0xc7	**EFI_DRIVER_CONFIGURATION_PROTOCOL.SetOptions – SetOptions()** returns **EFI_INVALID_PARAMETER** with invalid *ControllerHandle*	1. Call **SetOptions()** with invalid *ControllerHandle*. Return status of **SetOptions()** is **EFI_INVALID_PARAMETER**.
5.5.4.1.2	0x159d6867, 0x6e6f, 0x4cb0, 0x99, 0xc1, 0xdf, 0x57, 0x86, 0xc0, 0x61, 0x3f	**EFI_DRIVER_CONFIGURATION_PROTOCOL.SetOptions – SetOptions()** returns **EFI_INVALID_PARAMETER** with invalid *ActionRequired*	1. Call **SetOptions()** with an *ActionRequired* value of **NULL**. Return status must be **EFI_INVALID_PARAMETER**.
5.5.4.1.3	0x97465a70, 0x7746, 0x4116, 0x93, 0xbc, 0x22, 0xb1, 0xaa, 0x9e, 0x14, 0xa2	**EFI_DRIVER_CONFIGURATION_PROTOCOL.SetOptions – SetOptions()** returns **EFI_INVALID_PARAMETER** with invalid *ControllerHandle* & *ChildHandle.*	1. Call **SetOptions()** with: (*ControllerHandle* == **NULL** && *ChildHandle* != **NULL**). Return status must be **EFI_INVALID_PARAMETER**.
5.5.4.1.4	0x976f0e0a, 0xa696, 0x4922, 0x8a, 0x44, 0xf3, 0x50, 0xf5, 0x0b, 0xd5, 0xe8	**EFI_DRIVER_CONFIGURATION_PROTOCOL.SetOptions – SetOptions()** returns **EFI_UNSUPPORTED** with unsupported *Language*.	1. Parse the **EFI_DRIVER_CONFIGURATION_PROTOCOL.SupportedLanguage**, compare with the language code repository. If could not find out an unsupported language, then skip this checkpoint. 2. Call **SetOptions()** with all unsupported *Language* codes. Each return status must be **EFI_UNSUPPORTED**.
5.5.4.1.5	0x12b263e5, 0xcb83, 0x4855, 0x94, 0x35, 0x6e, 0xfb, 0x53, 0x9d, 0x22, 0x51	**EFI_DRIVER_CONFIGURATION_PROTOCOL.SetOptions – SetOptions()** returns **EFI_UNSUPPORTED** with unsupported *ControllerHandle*.	1. Test case creates a virtual device handle that does not stand for any device controller. 2. Input this handle as the *ControllerHandle* input for the **SetOptions()**. The return code must be **EFI_UNSUPPORTED**.

7.4.2 OptionsValid()

Number	GUID	Assertion	Test Description
5.5.4.2.1	0x9a4ba394, 0xbf63, 0x4dba, 0xaf, 0x83, 0xc7, 0x50, 0xc9, 0xff, 0xaa, 0xf4	**EFI_DRIVER_CONFIGURATION_PROTOCOL.OptionsValid – OptionsValid()** returns **EFI_INVALID_PARAMETER** with invalid *ControllerHandle*.	1. Call **OptionsValid()** with invalid *ControllerHandle*. Return status must be **EFI_INVALID_PARAMETER**.
5.5.4.2.2	0x10a4cd4b, 0x0e42, 0x4bed, 0x9b, 0x3e, 0x53, 0x21, 0x50, 0x9c, 0xd0, 0xf6	**EFI_DRIVER_CONFIGURATION_PROTOCOL.OptionsValid – OptionsValid()** returns **EFI_UNSUPPORTED** with unsupported *ControllerHandle*.	1. Test case creates a virtual device handle that does not stand for any device controller. 2. Input this handle as the *ControllerHandle* input for the **OptionsValid()**. It should return **EFI_UNSUPPORTED**.

7.4.3 ForceDefaults()

Number	GUID	Assertion	Test Description
5.5.4.3.1	0x45b89573, 0xff7d, 0x4549, 0xbc, 0x5f, 0x7f, 0x23, 0x04, 0xa1, 0x1c, 0x43	**EFI_DRIVER_CONFIGURATION_PROTOCOL.ForceDefaults – ForceDefaults()** returns **EFI_INVALID_PARAMETER** with invalid *ControllerHandle*.	1. Call **ForceDefaults()** with invalid *ControllerHandle*. Return status must be **EFI_INVALID_PARAMETER**.
5.5.4.3.2	0x0ede4bce, 0x0456, 0x45e5, 0x86, 0x04, 0x88, 0xc4, 0xa2, 0xbb, 0x7c, 0xa1	**EFI_DRIVER_CONFIGURATION_PROTOCOL.ForceDefaults – ForceDefaults()** returns **EFI_INVALID_PARAMETER**. with an *ActionRequired* value of **NULL**	1. Call **ForceDefaults()** with an *ActionRequired* value of **NULL**. Return status must be **EFI_INVALID_PARAMETER**.
5.5.4.3.3	0x0e7dd3db, 0x072b, 0x45b6, 0xaa, 0xdf, 0xf3, 0xed, 0xed, 0x37, 0xe6, 0xae	**EFI_DRIVER_CONFIGURATION_PROTOCOL.ForceDefaults – ForceDefaults()** returns **EFI_UNSUPPORTED** with unsupported *ControllerHandle*.	1. Test case creates a virtual device handle that does not stand for any device controller. 2. Input this handle as the *ControllerHandle* input for the **ForceDefaults()**. It should return **EFI_UNSUPPORTED**.

7.5 EFI_DRIVER_DIAGNOSTICS_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_DRIVER_DIAGNOSTICS_PROTOCOL Section.

7.5.1 RunDiagnostic()

Number	GUID	Assertion	Test Description
5.5.5.1.1	0xe6351da7, 0x8e29, 0x451b, 0xb1, 0x16, 0xda, 0x93, 0x29, 0x97, 0x0f, 0x17	**EFI_DRIVER_DIAGNOSTIC_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_INVALID_PARAMETER** with invalid *ControllerHandle*.	1. Call **RunDiagnostics()** with invalid *ControllerHandle*. Return Status must be **EFI_INVALID_PARAMETER**
5.5.5.1.2	0xf98940fb, 0x1ae6, 0x42a8, 0x95, 0xb3, 0xd3, 0x90, 0x84, 0x17, 0x2e, 0xb7	**EFI_DRIVER_DIAGNOSTIC_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_INVALID_PARAMETER** with a *Language* value of **NULL**.	1. Call **RunDiagnostics()** with a *Language* value of **NULL**. Return Status must be **EFI_INVALID_PARAMETER**
5.5.5.1.3	0xe348a9ee, 0x10fc, 0x4487, 0x8c, 0x1a, 0xfc, 0xa8, 0x11, 0xd7, 0xbb, 0x24	**EFI_DRIVER_DIAGNOSTIC_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_INVALID_PARAMETER** with an *ErrorType* value of **NULL.**	1. Call **RunDiagnostics()** with an *ErrorType* value of **NULL**. Return Status must be **EFI_INVALID_PARAMETER**
5.5.5.1.4	0x1f03e17d, 0x3f3c, 0x45ab, 0x93, 0xf5, 0xd3, 0xde, 0x3e, 0xc3, 0xe3, 0xcc	**EFI_DRIVER_DIAGNOSTIC_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_INVALID_PARAMETER** with a *BufferSize* value of **NULL**.	1. Call **RunDiagnostics()** with a *BufferSize* value of **NULL**. Return status must be **EFI_INVALID_PARAMETER**
5.5.5.1.5	0x7a73befe, 0xb271, 0x486f, 0x9b, 0x0e, 0x97, 0x3c, 0x5e, 0x80, 0x64, 0xd9	**EFI_DRIVER_DIAGNOSTIC_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_INVALID_PARAMETER** with a *Buffer* value of **NULL**.	1. Call **RunDiagnostics()** with a *Buffer* value of **NULL**. Return status must be **EFI_INVALID_PARAMETER**

Number	GUID	Assertion	Test Description
5.5.5.1.6	0xaeab03a7, 0xfa56, 0x4e97, 0x8e, 0x1c, 0xc3, 0x35, 0xb4, 0xa4, 0xb4, 0x1c	**EFI_DRIVER_DIAGNOSTIC_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_UNSUPPORTED** with unsupported *Language*.	1. Parse the **EFI_DRIVER_DIAGNOSTICS_PROTOCOL.SupportedLanguage**, compare with the language code repository. If could not find out an unsupported language, then skip this checkpoint. 2. Call **RunDiagnostics()** with all unsupported Language codes. Each return status of **RunDiagnostics()** is **EFI_UNSUPPORTED**.
5.5.5.1.7	0xf8d9425c, 0x4bc8, 0x44a9, 0xa4, 0x33, 0x9a, 0x2c, 0x01, 0xec, 0x58, 0x27	**EFI_DRIVER_DIAGNOSTIC_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_UNSUPPORTED** with unsupported *ControllerHandle*.	1. Test case creates a virtual device handle that does not stand for any device controller. 2. Input this handle as the *ControllerHandle* input for the **RunDiagnostics()**. It should return **EFI_UNSUPPORTED**.

7.6 EFI_DRIVER_DIAGNOSTICS2_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_DRIVER_DIAGNOSTICS2_PROTOCOL Section.

7.6.1 RunDiagnostic()

Number	GUID	Assertion	Test Description
5.5.6.1.1	0x6c872dce, 0x787e, 0x44dc, 0xa8, 0x87, 0xea, 0x1b, 0x8d, 0x55, 0xfd, 0x59	**EFI_DRIVER_DIAGNOSTIC2_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_INVALID_PARAMETER** with **NULL** *ControllerHandle*.	1. Call **RunDiagnostics()** with **NULL** *ControllerHandle*. Return Status must be **EFI_INVALID_PARAMETER**
5.5.6.1.2	0xf3263eb0, 0x1630, 0x4749, 0x98, 0xe6, 0xc9, 0x50, 0x23, 0x15, 0xd3, 0xa2	**EFI_DRIVER_DIAGNOSTIC2_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_INVALID_PARAMETER** with invalid *ChildHandle*.	1. Call **RunDiagnostics()** with invalid *ChildHandle*. Return Status must be **EFI_INVALID_PARAMETER**

Number	GUID	Assertion	Test Description
5.5.6.1.3	0xc5b8e4ef, 0x2fa4, 0x4ae9, 0xa6, 0x5e, 0xdd, 0x47, 0x2d, 0xfd, 0x81, 0xe5	**EFI_DRIVER_DIAGNOSTIC2_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_INVALID_PARAMETER** with **Language** value of **NULL.**	1. Call **RunDiagnostics()** with **Language** value of **NULL**. Return Status must be **EFI_INVALID_PARAMETER**
5.5.6.1.4	0xe23426c8, 0x5fe2, 0x4e80, 0xa9, 0x40, 0xab, 0x66, 0x10, 0x63, 0x28, 0xf6	**EFI_DRIVER_DIAGNOSTIC2_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_INVALID_PARAMETER** with *ErrorType* value of **NULL**.	1. Call **RunDiagnostics()** with *ErrorType* value of **NULL**. Return status must be **EFI_INVALID_PARAMETER**
5.5.6.1.5	0x6e86ac1a, 0x0ce8, 0x4f83, 0x9d, 0xa2, 0x38, 0x79, 0x1e, 0xff, 0x0f, 0x8c	**EFI_DRIVER_DIAGNOSTIC2_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_INVALID_PARAMETER** with a *BufferSize* value of **NULL**.	1. Call **RunDiagnostics()** with a *BufferSize* value of **NULL**. Return status must be **EFI_INVALID_PARAMETER**
5.5.6.1.6	0x4c955e4c, 0x86b9, 0x4c6d, 0x83, 0xa0, 0x4e, 0xa3, 0x34, 0x67, 0xd0, 0x38	**EFI_DRIVER_DIAGNOSTIC2_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_INVALID_PARAMETER** with a *Buffer* value of **NULL**.	1. Call **RunDiagnostics()** with a *Buffer* value of **NULL**. Return status must be **EFI_INVALID_PARAMETER**
5.5.6.1.7	0x8b218e7b, 0x24a0, 0x400c, 0xa8, 0x69, 0x1a, 0xd1, 0x14, 0x8e, 0x7a, 0x07	**EFI_DRIVER_DIAGNOSTIC2_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_UNSUPPORTED** with unsupported *Language*.	1. Parse the **EFI_DRIVER_DIAGNOSTICS_PROTOCOL.SupportedLanguage**, compare with the language code repository. If could not find out an unsupported language, then skip this checkpoint. 2. Call **RunDiagnostics()** with all unsupported Language codes. Each return status of **RunDiagnostics()** is **EFI_UNSUPPORTED**.

Number	GUID	Assertion	Test Description
5.5.6.1.8	0xef071998, 0xeb8d, 0x488f, 0xa5, 0xd5, 0x9e, 0x44, 0x7a, 0x54, 0x20, 0x8b	**EFI_DRIVER_DIAGNOSTIC2_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_UNSUPPORTED with virtual device handle**	1. Test case creates a virtual device handle that does not stand for any device controller. 2. Input this handle as the *ControllerHandle* input for the **RunDiagnostics()**. It should return **EFI_UNSUPPORTED**.
5.5.6.1.9	0xc9da5237, 0x6ad0, 0x4c74, 0x88, 0xd0, 0x6e, 0x51, 0x7f, 0x6c, 0x4f, 0x63	**EFI_DRIVER_DIAGNOSTIC2_PROTOCOL.RunDiagnostics – RunDiagnostics()** return **EFI_UNSUPPORTED** with virtual child handle	1. Test case creates a virtual device handle that does not stand for any device controller. 2. Input this handle as the *ChildHandle* input for the **RunDiagnostics()**. It should return **EFI_UNSUPPORTED**.
5.5.6.1.10	0x2e31c21e, 0x1999, 0x42b7, 0x96, 0xe6, 0xda, 0x8e, 0xfc, 0xc1, 0xf1, 0x51	**EFI_DRIVER_DIAGNOSTIC2_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_SUCCESS** with supported *Language*.	1. Call **RunDiagnostics()** with supported *Language*. Return status must be **EFI_SUCCESS.**
5.5.6.1.11	0x04405fac, 0x1688, 0x4213, 0xa1, 0x1d, 0x4b, 0x64, 0x58, 0xff, 0xe7, 0x2c	**EFI_DRIVER_DIAGNOSTIC2_PROTOCOL.RunDiagnostics – RunDiagnostics()** returns **EFI_SUCCESS** with supported ***Language***.	1. Call **RunDiagnostics()** with supported ***Language***.. Return status must be **EFI_SUCCESS.**

7.7 EFI_COMPONENT_NAME_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_COMPONENT_NAME_PROTOCOL Section.

7.7.1 GetDriverName()

Number	GUID	Assertion	Test Description
5.5.7.1.1	0x628fcfba, 0xc74b, 0x4038, 0x91, 0x5a, 0x01, 0x1a, 0xb9, 0x0f, 0x67, 0x35	**EFI_COMPONENT_NAME_PROTOCOL.GetDriverName – GetDriverName()** returns its driver name in every supported language.	For each supported language: 1. Call **GetDriverName()** to retrieve current driver's name. 2. Dump the returned driver name. Each return code of **GetDriverName()** should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.5.7.1.2	0x59ed70e0, 0x9cc8, 0x48d5, 0x86, 0x75, 0xed, 0xcb, 0xb0, 0x88, 0xeb, 0xd9	`EFI_COMPONENT_NAME_PROTOCOL.GetDriverName` – `GetDriverName()` returns `EFI_INVALID_PARAMETER` with a *`Language`* value of `NULL`.	1. Call `GetDriverName()` with a *`Language`* value of `NULL`. THe return status of `GetDriverName()` is `EFI_INVALID_PARAMETER`.
5.5.7.1.3	0x9cffff0f, 0x65a7, 0x43a5, 0x9e, 0xf1, 0x74, 0x02, 0x27, 0x82, 0x3d, 0xfc	`EFI_COMPONENT_NAME_PROTOCOL.GetDriverName` – `GetDriverName()` returns `EFI_INVALID_PARAMETER` with a *`DriverName`* value of `NULL`.	1. Call `GetDriverName()` with a *`DriverName`* value of `NULL`. The return status of `GetDriverName()` is `EFI_INVALID_PARAMETER`.
5.5.7.1.4	0xcb089876, 0xe819, 0x4fd8, 0xac, 0xbe, 0x47, 0x56, 0x8c, 0x10, 0x93, 0xcc	`EFI_COMPONENT_NAME_PROTOCOL.GetDriverName` – `GetDriverName()` returns `EFI_UNSUPPORTED` with unsupported *`Language`*.	1. Parse the `EFI_COMPONENT_NAME_PROTOCOL.SupportedLanguage`, compare with the language code repository. If could not find out an unsupported language, then skip this checkpoint. 2. Call `GetDriverName()` with all unsupported *`Language`* codes. Each return status of `GetDriverName()` is `EFI_UNSUPPORTED`.

7.7.2 GetControllerName()

Number	GUID	Assertion	Test Description
5.5.7.2.1	0x961fabd3, 0x97ec, 0x4c97, 0xa0, 0x5a, 0xc2, 0xfd, 0xa6, 0x32, 0xf1, 0x3d	`EFI_COMPONENT_NAME_PROTOCOL.GetControllerName` – `GetGontrollerName()` must successfully retrieve *`ControllerName`* for all manageable *`ControllerHandle`*.	1. Retrieve all controller handles that are managed by the driver specified by the component protocol instance. 2. For each *`ControllerHandle`* Call `GetControllerName()` with the *`ControllerHandle`* and at the same time, with a *`ChildHandle`* value of `NULL` in every supported language. The `GetControllerName()` should return `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.5.7.2.2	0xa83cfe57, 0x8391, 0x472b, 0xbc, 0x0e, 0x12, 0x18, 0x95, 0x06, 0x86, 0x70	`EFI_COMPONENT_NAME_PROTOCOL.GetControllerName – GetGontrollerName()` must successfully retrieve *ControllerName* for *ChildHandle* of manageable *ControllerHandle*.	1. Retrieve all controllers that are managed by the driver specified by the component instance. 2. Retrieve all child controllers. (If the controller has no child controller, then skip this checkpoint). 3. For each controller and its child controller: Call `GetControllerName()` with every child controller of the bus controller. The `GetControllerName()` should return `EFI_SUCCESS`.
5.5.7.2.3	0x735f5c9b, 0x95c9, 0x4949, 0xa8, 0xf7, 0x0a, 0x61, 0x06, 0x2e, 0x28, 0x67	`EFI_COMPONENT_NAME_PROTOCOL.GetControllerName – GetGontrollerName()` returns `EFI_INVALID_PARAMETER` with invalid *ControllerHandle*.	1. Call `GetControllerName()` with invalid *ControllerHandle*. The return status of `GetControllerName()` is `EFI_INVALID_PARAMETER`.
5.5.7.2.4	0x6f51eca4, 0x1808, 0x4b5b, 0x96, 0x9b, 0x88, 0xd8, 0xc8, 0xa5, 0x00, 0x3e	`EFI_COMPONENT_NAME_PROTOCOL.GetControllerName – GetGontrollerName()` returns `EFI_INVALID_PARAMETER` with invalid *ChildHandle* when the driver is not a device driver.	Call `GetControllerName()` with invalid *ChildHandle* when the driver is not a device driver. The return status of `GetControllerName()` is `EFI_INVALID_PARAMETER`.
5.5.7.2.5	0x9d3dedbf, 0xa123, 0x475b, 0xb6, 0x3e, 0x15, 0x01, 0xbc, 0x99, 0x81, 0x83	`EFI_COMPONENT_NAME_PROTOCOL.GetControllerName – GetGontrollerName()` returns `EFI_INVALID_PARAMETER` with a *ControllerName* value of `NULL`.	1. Call `GetControllerName()` with a *ControllerName* value of `NULL`. The return status of `GetControllerName()` is `EFI_INVALID_PARAMETER`.
5.5.7.2.6	0xb436d551, 0xf2f4, 0x4fdc, 0xb0, 0x31, 0x07, 0x3d, 0xad, 0xec, 0xd7, 0x16	`EFI_COMPONENT_NAME_PROTOCOL.GetControllerName – GetGontrollerName()` returns `EFI_INVALID_PARAMETER` with a *Language* value of `NULL`	1. Call `GetControllerName()` with a *Language* value of `NULL` The return status of `GetControllerName()` is `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.5.7.2.7	0x27a4781a, 0xe85a, 0x4714, 0xab, 0x9a, 0x67, 0xc1, 0x01, 0x38, 0x5e, 0x83	EFI_COMPONENT_NAME_PROTOCOL.GetControllerName – GetControllerName() returns EFI_UNSUPPORTED with unsupported *Language*.	1. Parse the EFI_COMPONENT_NAME_PROTOCOL.SupportedLanguage, compare with the language code repository. If could not find out an unsupported language, then skip this checkpoint. 2. Find out all controller handles that will cause GetControllerHandle() return EFI_SUCCESS when with supported Language. 3. Call GetDriverName() with each *ControllerHandle* and at the same time with those unsupported *Language* codes. When input with unsupported *Language*, the return status of GetControllerName() should be EFI_UNSUPPORTED.
5.5.7.2.8	0xa1a56539, 0x8150, 0x483f, 0xa1, 0xb7, 0x23, 0xaf, 0x4f, 0x84, 0x64, 0xc7	EFI_COMPONENT_NAME_PROTOCOL.GetControllerName – GetControllerName() returns EFI_UNSUPPORTED with irrelevant *ControllerHandle*	1. Test case creates a virtual device handle that does not stand for any device controller. 2. Input this handle as the *ControllerHandle* input for the GetControllerName(). It should return EFI_UNSUPPORTED.
5.5.7.2.9	0x8a5321c3, 0x3e88, 0x4c62, 0xbf, 0xdd, 0xc7, 0xe4, 0xec, 0xf5, 0x1f, 0x9f	EFI_COMPONENT_NAME_PROTOCOL.GetControllerName – GetControllerName() returns EFI_UNSUPPORTED with irrelevant *ChildHandle*	1. Test case creates a virtual device handle that does not stand for any device controller. 2. Input this handle as the *ChildHandle* input for the GetControllerName() (at the same time, the *ControllerHandle* should be valid). It should also return EFI_UNSUPPORTED.
5.5.7.2.10	0xa5ecbbe1, 0x1795, 0x4798, 0xa8, 0x26, 0x20, 0x9c, 0x57, 0x8e, 0x1d, 0xe9	EFI_COMPONENT_NAME_PROTOCOL.GetControllerName – GetControllerName() returns EFI_UNSUPPORTED with device handle and not-NULL *ChildHandle*	1. Test case gets a valid device handle, and an invalid *ChildHandle*. 2. Input this device handle as *ControllerHandle* and the *ChildHandle*. It should return EFI_UNSUPPORTED.

Number	GUID	Assertion	Test Description
5.5.7.2.11	0xdb9e40a7, 0x8638, 0x4c0f, 0xb2, 0x94, 0xfe, 0x05, 0x23, 0xfa, 0x1e, 0x2f	**EFI_COMPONENT_NAME_PROTOCOL.GetControllerName – GetGontrollerName()** go through each of the handles	Call **GetGontrollerName()** with all of the handles. The return status should not be **EFI_INVALID_PARAMETER.**
5.5.7.2.12	0x79ab9a12, 0xe535, 0x4727, 0xa0, 0x4d, 0x20, 0xb7, 0x8f, 0x91, 0x8f, 0x85	**EFI_COMPONENT_NAME_PROTOCOL.GetControllerName – GetGontrollerName()** go through each of the handles and child handles	Call **GetGontrollerName()** with all of the handles and child handles. The return status should not be **EFI_INVALID_PARAMETER.**

7.8 EFI_COMPONENT_NAME2_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_COMPONENT_NAME2_PROTOCOL Section.

7.8.1 GetDriverName()

Number	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_NAME_PROTOCOL.GetDriverName – GetDriverName()** returns **EFI_UNSUPPORTED** with unsupported language	Call **GetDriverName()** with unsupported *Language*. The returned status should be **EFI_UNSUPPORTED**

Number	GUID	Assertion	Test Description
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

7.8.2 GetControllerName()

Number	GUID	Assertion	Test Description
5.5.8.2.1	0xc38a85af, 0x2d0a, 0x4bfa, 0x8f, 0x44, 0xa2, 0x47, 0xf1, 0xfd, 0x7b, 0x94	COMPONENT_NAME2_PROTOCOL.GetControllerName - GetControllerName() returns EFI_INVALID_PARAMETER with invalid ControllerHandle	Call GetControllerName() with invalid ControllerHandle. The returned status should be EFI_INVALID_PARAMETER
5.5.8.2.2	0xde8c8d23, 0x4aa6, 0x4dd7, 0x93, 0xbd, 0x35, 0x78, 0x40, 0x67, 0x6b, 0xff	COMPONENT_NAME2_PROTOCOL.GetControllerName - GetControllerName() returns EFI_INVALID_PARAMETER with invalid *ChildHandle* and non-device ControllerHandle	Call GetControllerName() with valid Bus Handle(non-device ControllerHandle) and invalid *ChildHandle*. The returned status should be EFI_INVALID_PARAMETER
5.5.8.2.3	0x8398d1d9, 0xdfb7, 0x47f1, 0xad, 0x65, 0x36, 0xf1, 0x2a, 0x6a, 0x47, 0xea	COMPONENT_NAME2_PROTOCOL.GetControllerName - GetControllerName() returns EFI_INVALID_PARAMETER with NULL ControllerName	Call GetControllerName() with valid device ControllerHandle and NULL ControllerName. The returned status should be EFI_INVALID_PARAMETER
5.5.8.2.4	0x8cf65e39, 0x125b, 0x4206, 0x99, 0x85, 0xca, 0xa5, 0x15, 0x68, 0x7b, 0x0a	COMPONENT_NAME2_PROTOCOL.GetControllerName - GetControllerName() returns EFI_INVALID_PARAMETER with NULL *Language*	Call GetControllerName() with valid device ControllerHandle and NULL *Language*. The returned status should be EFI_INVALID_PARAMETER

Number	GUID	Assertion	Test Description
5.5.8.2.5	0x064d252b, 0xbc7f, 0x4859, 0x86, 0x02, 0xaf, 0xa9, 0x7f, 0x8e, 0xa2, 0xbd	`COMPONENT_NAME2_PROTOCOL.GetControllerName - GetControllerName()` returns `EFI_UNSUPPORTED` with unsupported language	Call `GetControllerName()` with unsupported *`Language`*. The returned status should be `EFI_UNSUPPORTED`
5.5.8.2.6	0x95c8bfd8, 0xc67c, 0x411e, 0x93, 0x95, 0x43, 0x28, 0x01, 0x2c, 0x07, 0x66	`COMPONENT_NAME_PROTOCOL.GetControllerName - GetControllerName()` returns `EFI_UNSUPPORTED` with irrelevant ControllerHandle	Call `GetControllerName()` with irrelevant *`ControllerHandle`*. The returned status should be `EFI_UNSUPPORTED`
5.5.8.2.7	0x155c06f0, 0xe315, 0x4175, 0xa0, 0xe9, 0x4d, 0xe3, 0xc5, 0x16, 0x3c, 0xb2	`COMPONENT_NAME_PROTOCOL.GetControllerName - GetControllerName()` returns `EFI_UNSUPPORTED` with irrelevant *`ChildHandle`*	Call `GetControllerName()` with irrelevant *`ChildHandle`*. The returned status should be `EFI_UNSUPPORTED`
5.5.8.2.8	0xabf5cd96, 0xfb74, 0x489c, 0xae, 0x70, 0xeb, 0x31, 0xa0, 0xfd, 0xef, 0x25	`COMPONENT_NAME2_PROTOCOL.GetControllerName - GetControllerName()` returns `EFI_SUCCESS` with supported language	Call `GetControllerName()` with Supported *`Language`* and valid *`ControllerHandle`*. The returned status should be `EFI_SUCCESS`
5.5.8.2.9	0x38bd708a, 0xf1d7, 0x4b3b, 0xb2, 0x39, 0x06, 0xf6, 0xfd, 0xa2, 0x1c, 0xb8	`COMPONENT_NAME2_PROTOCOL.GetControllerName - GetControllerName()` returns `EFI_SUCCESS` support language	Call `GetControllerName()` with Supported *`Language`*, valid *`ControllerHandle`* and valid *`ChildHandle`*. The returned status should be `EFI_SUCCESS`

7.9 EFI_PLATFORM_TO_DRIVER_CONFIGURATION_PROTOCOL

Reference Document:

UEFI Specification, EFI_PLATFORM_TO_DRIVER_CONFIGURATION_PROTOCOL Section.

7.9.1 Query()

Number	GUID	Assertion	Test Description
5.5.9.1.1	0x6acc3f19, 0xe9b, 0x4ff7, 0xbd,0xd0, 0x7e,0x49, 0x19,0x6, 0xa8, 0xdd	EFI_PLATFORM_TO_DRIVER_CONFIGURATION_PROTOCOL.Query – Invoke Query() and verify interface correctness	1. Call Query() with valid ControllerHandle and Instance 2. if EFI_SUCCESS, get the next ControllerHandle till the end 3. The return status should be EFI_SUCCESS except the last one. The last one should be EFI_NOT_FOUND.
5.5.9.1.2	0x4cfb435, 0x4569, 0x48bb, 0x8c,0x8a, 0xba,0x2a, 0xa7,0x5f, 0x16,0xe2	EFI_PLATFORM_TO_DRIVER_CONFIGURATION_PROTOCOL.Query – Invoke Query() with invalid ControllerHandle	Call Query() with invalid ControllerHandle, it should return EFI_INVALID_PARAMETER
5.5.9.1.3	0x28730223, 0x508, 0x46c9, 0x83, 0xf7, 0x94, 0xec, 0x52, 0x4, 0x65, 0x2a	EFI_PLATFORM_TO_DRIVER_CONFIGURATION_PROTOCOL.Query – Invoke Query() with invalid Instance	Call Query() with Instance is NULL, it should return EFI_INVALID_PARAMETER

7.9.2 Response()

Number	GUID	Assertion	Test Description
5.5.9.2.1	0x15cd60c3, 0xb30, 0x44df, 0xbe,0x9, 0x0,0xfa, 0x9f,0xe6, 0xf8,0xc5	EFI_PLATFORM_TO_DRIVER_CONFIGURATION_PROTOCOL.Response – Invoke Response() and verify interface correctness	1. Call Query() with valid ControllerHandle and Instance, call Response() with the same ControllerHandle and the arguments returned from Query() 2. if EFI_SUCCESS, get the next ControllerHandle till the end 3. The return status should be EFI_SUCCESS except the last one. The last one should be EFI_NOT_FOUND.
5.5.9.2.2	0x88e2dc36, 0x4d7b, 0x467a, 0xbb,0x60, 0xc9,0x97, 0xb7,0x22, 0xb7,0x12	EFI_PLATFORM_TO_DRIVER_CONFIGURATION_PROTOCOL.Response – Invoke Response() with invalid ControllerHandle	Call Query() and Response() with invalid ControllerHandle, the return status should be EFI_INVALID_PARAMETER

7.9.3 DMTF SM CLP ParameterTypeGuid

Number	GUID	Assertion	Test Description
5.5.9.3.1	0x35a69b6e, 0x1755, 0x41ca, 0x97,0xd7, 0xab,0xc3, 0xb7,0xb7, 0x7c,0xd3	`EFI_CONFIGURE_CLP_PARAMETER_BLK.CLPCommand` - verify the DMTF CLP command line `NULL`-terminated string and return `EFI_SUCCESS`.	1. Invoke `Query()`, produce `EFI_CONFIGURE_CLP_PARAMETER_BLK`. 2. Verify ParameterTypeGuid. 3. Compare the CLPCommand string with Standard command verbs and options. The return code should be `EFI_SUCCESS`
5.5.9.3.2	0x77b6a0b3, 0x7efe, 0x42f8, 0x98,0xcf, 0xf5,0x49, 0x51,0xe7, 0x1c,0x2c	`EFI_CONFIGURE_CLP_PARAMETER_BLK.CLPReturnString` – verify the CLP return string is “format=keyword” format	1. Invoke `Query()` and `Response()`, produce `EFI_CONFIGURE_CLP_PARAMETER_BLK` 2. Verify the CLPReturnString format is “format=keyword” format. The return code should be `EFI_SUCCESS`
5.5.9.3.3	0xd7cacc21, 0x4e96, 0x444c, 0x91,0xcb, 0x70,0x4e, 0x3f,0xa8, 0x31,0x33	`EFI_CONFIGURE_CLP_PARAMETER_BLK.CLPCmdStatus` - with valid command and return the command status of CLP with `EFI_SUCCESS`.	1. Invoke `Query()` and `Response()`, produce `EFI_CONFIGURE_CLP_PARAMETER_BLK` 2. Compare the CLPCmdStatus string with Standard command return status table. The return code should be `EFI_SUCCESS`
5.5.9.3.4	0x69e16544, 0x23bd, 0x4b46, 0x9d,0xe5, 0xe0,0x6a, 0xb4,0x3d, 0x8b,0x12	`EFI_CONFIGURE_CLP_PARAMETER_BLK.CLPErrorValue` - compare this parameter with CLP Error Value and return code `EFI_SUCCESS`.	1. Invoke `Query()` and `Response()`, produce `EFI_CONFIGURE_CLP_PARAMETER_BLK` 2. Compare the CLPErrorValue with Error Values. The return code should be `EFI_SUCCESS`
5.5.9.3.5	0x78e97814, 0x4c3d, 0x42b3, 0xae,0x7c, 0x7b,0x16, 0x61,0x69, 0x32,0x4a	`EFI_CONFIGURE_CLP_PARAMETER_BLK. CLPMsgCode` - compare with CLP Message Code, return code `EFI_SUCCESS`.	1. Invoke `Query()` and `Response()`, produce `EFI_CONFIGURE_CLP_PARAMETER_BLK` 2. verify the CLPMsgCode is equal to the CLP Probable Cause Value, the return code should be `EFI_SUCCESS`

7.10 EFI_DRIVER_SUPPORTED_EFI_VERSION_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_DRIVER_SUPPORTED_EFI_VERSION_PROTOCOL Section.

Number	GUID	Assertion	Test Description
5.5.10.1.1	0x9b72180d, 0x155c, 0x4b7a, 0xbb, 0xa8, 0x99, 0x83, 0x7b, 0x2f, 0x9d, 0xf8	`EFI_DRIVER_SUPPORTED_EFI_VERSION_PROTOCOL.Length` - verify this value is the structure length, and return `EFI_SUCCESS`.	Verify the entire structure length is correct and return `EFI_SUCCESS` value.
5.5.10.1.2	0xac1951b1, 0x7243, 0x40a9, 0xa0, 0x1, 0x9d, 0x9d, 0x6e, 0x44, 0x8f, 0x5a	`EFI_DRIVER_SUPPORTED_EFI_VERSION_PROTOCOL.FirmwareVersion` - verify the parameter with `EFI_2_10_SYSTEM_REVISION`. Return `EFI_SUCCESS` or `EFI_INCOMPATIBLE_VERSION`..	Initialize the `EFI_VERSION_PTOTOCOL` and compare the version of the EFI Specification that driver conforms to with `EFI_2_10_SYSTEM_REVISION`. If equal, return `EFI_SUCCESS;` if not, return value should be `EFI_INCOMPATIBLE_VERSION.` .

7.11 EFI_ADAPTER_INFORMATION_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_ADAPTER_INFORMATION_PROTOCOL Section.

7.11.1 GetInformation()

Number	GUID	Assertion	Test Description
5.5.11.1.1	0x0d68257b, 0xf647, 0x452a,0x97, 0x44,0xa2, 0x23,0xe6, 0xee,0x3d, 0xf2	`EFI_ADAPTER_INFORMATION_PROTOCOL.GetInformation` - `GetInformation()` returns `EFI_SUCCESS` with valid information type.	Call `GetSupportedTypes()` to get the valid Information type. Call `GetInformation ()`, the return status should be `EFI_SUCCESS` and the InformationBlock != NULL.

Number	GUID	Assertion	Test Description
5.5.11.1.2	0x15a3a10d, 0xca48, 0x4d52,0x99, 0x89,0x51, 0x71,0xfc, 0x90,0x90, 0x54	EFI_ADAPTER_INFORMATION_PROTOCOL.GetInformation - GetInformation() returns correct InformationBlockSize.	Call GetSupportedTypes() to get the valid Information type. Call GetInformation () the return status should be EFI_SUCCESS and the InformationBlock != NULL. Compare the InformationBlockSize Received from step2 with the expected size. Their size should be equal.
5.5.11.1.3	0xeb7c1cc7, 0x5c94, 0x40c6,0xbe, 0xaf,0x53, 0x08,0xd7, 0xf6,0x35, 0x01	EFI_ADAPTER_INFORMATION_PROTOCOL.GetInformation - GetInformation() returns EFI_UNSUPPORTEDwith unknown InformationType.	Call GetInformation () with unknown InformationType, the return status should be EFI_UNSUPPORTED.
5.5.11.1.4	0xab0d01e7, 0x8f70, 0x4a76,0x87, 0x7e,0xa7, 0x13,0xce, 0x00,0x1b, 0x72	EFI_ADAPTER_INFORMATION_PROTOCOL.GetInformation - GetInformation() returns EFI_INVALID_PARAMETERS with NULL InformationBlock.	Call GetInformation ()with NULL InformationBlock, the return status should be EFI_INVALID_PARAMETERS.
5.5.11.1.5	0x5a831392, 0x7ee7, 0x4f3e,0xbc, 0xd6,0x32, 0x6d,0x64, 0xf9,0xc2, 0x1c	EFI_ADAPTER_INFORMATION_PROTOCOL.GetInformation - GetInformation() returns EFI_INVALID_PARAMETERS with NULL InformationBlockSize.	Call GetInformation ()with NULL InformationBlockSize, the return status should be EFI_INVALID_PARAMETERS.

7.11.2 SetInformation()

Number	GUID	Assertion	Test Description
5.5.11.2.1	0xeed7dbd9, 0x834c, 0x4dbf,0xa1, 0x8d,0x39, 0x9f,0xdf, 0x19,0xd3, 0xf0	EFI_ADAPTER_INFORMATION_PROTOCOL.SetInformation – SetInformation() returns EFI_SUCCESS with valid information type.	Call GetSupportedTypes() to get the valid Information type. Call GetInformation () the return status should be EFI_SUCCESS and the InformationBlock != NULL. Call SetInformation()the return status should be EFI_SUCCESS or EFI_WRITE_PROTECTED.
5.5.11.2.2	0x2e1eae6b, 0x95f1, 0x4189,0xac, 0x02,0xc8, 0x50,0x41, 0x02,0x3c, 0xca	EFI_ADAPTER_INFORMATION_PROTOCOL.SetInformation – SetInformation() returns EFI_SUCCESS with valid information type.	Call GetSupportedTypes() to get the valid Information type. Call GetInformation () the return status should be EFI_SUCCESS and the InformationBlock != NULL. Call SetInformation() the return status should be EFI_SUCCESS or EFI_WRITE_PROTECTED. Call GetInformation () and check the received information with the Information set by step3. They should be equal.
5.5.11.2.3	0xdb4d7a52, 0x608c, 0x46f7,0xaf, 0x23,0x0b, 0x10,0x1e, 0xc8,0xb8, 0xec	EFI_ADAPTER_INFORMATION_PROTOCOL.SetInformation – SetInformation() returns EFI_UNSUPPORTED with unknown InformationType.	Call SetInformation() with unknown InformationType, the return status should be EFI_UNSUPPORTED.

Number	GUID	Assertion	Test Description
5.5.11.2.4	0xd15882e0, 0xcb55, 0x42f4,0xbb, 0x30,0xcb, 0xa0,0x50, 0x3a,0xad, 0xc9	**EFI_ADAPTER_INFORMATION_PROTOCOL.SetInformation – SetInformation()** returns **EFI_INVALID_PARAMETER** or *EFI_WRITE_PROTECTED* with NULL InformationBlock.	Call **SetInformation()** with NULL InformationBlock, the return status should be **EFI_INVALID_PARAMETER** or **EFI_WRITE_PROTECTED.**

7.11.3 GetSupportedTypes()

Number	GUID	Assertion	Test Description
5.5.11.3.1	0x59a9f08d, 0xad58, 0x49e0,0x92, 0x7f,0x9b, 0x46,0xbb, 0x62,0x3b, 0x41	**EFI_ADAPTER_INFORMATION_PROTOCOL.GetSupportedTypes – GetSupportedTypes()** returns **EFI_SUCCESS.**	Call **GetSupportedTypes()** , the return status should be **EFI_SUCCESS.**
5.5.11.3.2	0xac9f6a14, 0xff26, 0x43d1,0x8c, 0x47,0x61, 0x56,0x00, 0xc4,0x12, 0xf4	EFI_ADAPTER_INFORMATION_PROTOCOL. GetSupportedTypes - GetSupportedTypes() returns EFI_SUCCESS.	Call **GetSupportedTypes()** to get the valid Information type. The Information type received from step1 should be one of the probable types.
5.5.11.3.3	0xd55b2936, 0x5f3f, 0x40a8,0xb8, 0xa1,0x40, 0x9f,0x59, 0x50,0xda, 0x61	**EFI_ADAPTER_INFORMATION_PROTOCOL.GetSupportedTypes – GetSupportedTypes()** returns **EFI_INVALID_PARAMETER** with NULL InfoTypesBuffer.	Call **GetSupportedTypes()** with NULL InfoTypesBuffer, the return status should be **EFI_INVALID_PARAMETER.**

5.5.11.3.4	0x890c711f, 0xce91, 0x4426,0xa5, 0xfd,0x01, 0x0a,0x1c, 0xa5,0x33, 0x5b	**`EFI_ADAPTER_INFORMATION_PROTOCOL.GetSupportedTypes()`** – **`GetSupportedTypes()`** returns **`EFI_INVALID_PARAMETER`** with NULL InfoTypesBufferCount.	Call **`GetSupportedTypes()`** with NULL InfoTypesBufferCount, the return status should be **`EFI_INVALID_PARAMETER`**.

8 Protocols Console Support Test

8.1 EFI_SIMPLE_ TEXT_INPUT_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_SIMPLE_TEXT_INPUT_PROTOCOL Section.

8.1.1 Reset()

Number	GUID	Assertion	Test Description
5.6.1.1.1	0x7cb5f8dd, 0x7346, 0x484b, 0xb1, 0xb3, 0xa6, 0x46, 0x69, 0x6d, 0xea, 0xe7	**EFI_SIMPLE_TEXT_INPUT_PROTOCOL.Reset – Reset()** returns **EFI_SUCCESS** with *ExtendedVerification* as **FALSE.**	1. Call **Reset()** with *ExtendedVerification* as **FALSE**. The return status should be **EFI_SUCCESS**.
5.6.1.1.2	0x6fc31add, 0xf34b, 0x4b56, 0x9b, 0xa6, 0x36, 0xb2, 0x7c, 0xbe, 0xf5, 0xa2	**EFI_SIMPLE_TEXT_INPUT_PROTOCOL.Reset – ReadKeyStroke()** returns **EFI_NOT_READY** when there is no key has been stroked.	1. Call **Reset()** with *ExtendedVerification* as **FALSE**. 2. After **Reset()**, do not stroke any key, and call **ReadKeyStroke()**. The return code should be **EFI_NOT_READY**
5.6.1.1.3	0x8da56db6, 0xd7df, 0x4029, 0xba, 0x98, 0x37, 0x46, 0x0b, 0x21, 0x0e, 0x3b	**EFI_SIMPLE_TEXT_INPUT_PROTOCOL.Reset – Reset()** returns **EFI_SUCCESS** with *ExtendedVerification* as **TRUE.**	1. Call **Reset()** with *ExtendedVerification* as **TRUE**. The return status should be **EFI_SUCCESS**.
5.6.1.1.4	0x3d51b174, 0x59f8, 0x44bc, 0xb7, 0xf7, 0x9a, 0x11, 0x2c, 0x51, 0x82, 0xa1	**EFI_SIMPLE_TEXT_INPUT_PROTOCOL.Reset – ReadKeyStroke()** returns **EFI_NOT_READY** when there is no key that has been stroked.	1. Call **Reset()** with *ExtendedVerification* as **TRUE**. 2. After **Reset()**, do not stroke any key, and call **ReadKeyStroke()**. The return code should be **EFI_NOT_READY**

8.1.2 ReadKeyStroke()

No automatic test case is designed to verify this function.

8.2 EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL Section.

UEFI 2.1 Specification, Section 11.3.

8.2.1 Reset()

Number	GUID	Assertion	Test Description
5.6.2.1.1	0xecaf43c6, 0x6b77, 0x413a, 0x89, 0x8f, 0x28, 0x0e, 0x92, 0x5f, 0xf9, 0x43	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.Reset – Reset()** without Extended Verification Mode	1. Call **Reset()** with *ExtendedVerification* as **FALSE**. 2. Check cursor position. It should be (0,0).
5.6.2.1.2	0xc40bba44, 0xcfa3, 0x4494, 0xaf, 0xa5, 0xfa, 0x2f, 0x78, 0xcb, 0x20, 0x20	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.Reset – Reset()** without Extended Verification Mode returns **EFI_SUCCESS**	1. Call **Reset()** with *ExtendedVerification* as **FALSE**. The return code should be **EFI_SUCCESS**
5.6.2.1.3	0x51267bf4, 0x7b3e, 0x46fd, 0xac, 0x6c, 0xff, 0x8e, 0x54, 0x61, 0xd1, 0x7f	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.Reset – Reset()** with Extended Verification Mode	1. Call **Reset()** with *ExtendedVerification* as **TRUE**. 2. Check cursor position. It should be (0,0).
5.6.2.1.4	0x1771a342, 0xbbc3, 0x43da, 0x91, 0x4d, 0x7d, 0x59, 0xb7, 0xd8, 0x86, 0x2e	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.Reset – Reset()** with Extended Verification Mode returns **EFI_SUCCESS**	1. Call **Reset()** with *ExtendedVerification* as **TRUE**. The return code should be **EFI_SUCCESS**

8.2.2 OutputString()

Number	GUID	Assertion	Test Description
5.6.2.2.1	0x3e706c2f, 0xc7ee, 0x43de, 0x8f, 0xe7, 0x39, 0x81, 0x33, 0x11, 0x7d, 0x9b	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.OutputString – OutputString()` with normal Unicode String. Cursor value in Mode should be assigned to the right position.	1. Call `OutputString()` with normal Unicode String. 2. Check cursor position. It should be at the end of the string. In addition, other attributes of output mode remain unchanged.
5.6.2.2.2	0xb7c77060, 0xbd1e, 0x4607, 0x85, 0x41, 0xdc, 0xf5, 0x08, 0xe3, 0xff, 0xd4	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.OutputString – OutputString()` with normal Unicode string returns `EFI_SUCCESS`	1. Call `OutputString()` with normal Unicode String. The return code should be `EFI_SUCCESS`
5.6.2.2.3	0xf3f07bdb, 0x683d, 0x448f, 0xa5, 0x4a, 0xb5, 0x61, 0xf9, 0x86, 0x95, 0xb5	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.OutputString – OutputString()` with very long Unicode String. Cursor value in Mode should be assigned to the right position.	1. Call `OutputString()` with very long Unicode String. 2. Check cursor position. It should be at the end of the string. In addition, other attributes of output mode remain unchanged.
5.6.2.2.4	0xcefd060c, 0x9ed5, 0x4862, 0x96, 0x75, 0xda, 0x26, 0x3b, 0xdc, 0x35, 0x3a	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.OutputString – OutputString()` with very long Unicode String returns `EFI_SUCCESS`	1. Call `OutputString()` with very long Unicode String. The return code should be `EFI_SUCCESS`
5.6.2.2.5	0x722925c0, 0xf84a, 0x4aa0, 0x9d, 0xe8, 0x04, 0x03, 0x70, 0xe0, 0x69, 0x0f	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.OutputString – OutputString()` with cursor control Unicode String. Cursor value in Mode should be assigned to the right position.	1. Call `OutputString()` with Drawing Unicode String. 2. Check cursor position. It should be at the end of the string. In addition, other attributes of output mode remain unchanged.
5.6.2.2.6	0x6fce5c66, 0xd273, 0x446d, 0x88, 0x54, 0x94, 0x7b, 0x6c, 0xd4, 0xa3, 0x96	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.OutputString – OutputString()` with Drawing Unicode String returns `EFI_SUCCESS`	1. Call `OutputString()` with Drawing Unicode String. The return code should be `EFI_SUCCESS`
5.6.2.2.7	0xae266668, 0xa3ef, 0x4930, 0x85, 0x64, 0x55, 0x9f, 0x9e, 0x96, 0x14, 0x6b	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.OutputString – OutputString()` with cursor control Unicode String. Cursor value in Mode should be assigned to the right position.	1. Call `OutputString()` with cursor control Unicode String. 2. Check cursor position. It should be at the appointed postion. In addition, other attributes of output mode remain unchanged.

Number	GUID	Assertion	Test Description
5.6.2.2.8	0x2e40bcfe, 0x7713, 0x4ab1, 0x99, 0x5c, 0xe0, 0x8b, 0x2d, 0xdc, 0x2b, 0x60	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.OutputString – OutputString() with cursor control Unicode String returns EFI_SUCCESS	1. Call OutputString() with cursor control Unicode String. The return code should be EFI_SUCCESS

8.2.3 TestString()

Number	GUID	Assertion	Test Description
5.6.2.3.1	0x0317202b, 0x4c09, 0x4f09, 0xa8, 0x9e, 0x17, 0x91, 0x7d, 0x0b, 0xb5, 0x6c	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.TestString – TestString() with cursor control Unicode String. Cursor value in Mode should be assigned to the right position.	1. Call TestString() with normal Unicode String. 2. Check cursor position. It should be at the end of the string. In addition, other attributes of output mode remain unchanged.
5.6.2.3.2	0x92609750, 0x7965, 0x4e08, 0xae, 0xaf, 0xb1, 0xec, 0xa3, 0x61, 0x63, 0x66	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.TestString – TestString() with normal Unicode string returns EFI_SUCCESS	1. Call TestString() with normal Unicode String Mode value. The return code should be EFI_SUCCESS

8.2.4 QueryMode()

Number	GUID	Assertion	Test Description
5.6.2.4.1	0x26d95327, 0x008c, 0x4ca1, 0xb6, 0x75, 0x9d, 0x86, 0x20, 0xdf, 0x73, 0x19	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.QueryMode – QueryMode() with supported *ModeNumber* value remains other attributes unchanged	1. Call QueryMode() with supported *ModeNumber* value. Other attributes should remain unchanged.
5.6.2.4.2	0xf2b8054e, 0xcfa7, 0x4fcd, 0x9e, 0x6c, 0xc6, 0x07, 0xbe, 0x62, 0xff, 0x27	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.QueryMode – QueryMode() with supported *ModeNumber* value returns EFI_SUCCESS	1. Call QueryMode() with supported *ModeNumber* value. The return code should be EFI_SUCCESS

Number	GUID	Assertion	Test Description
5.6.2.4.3	0x4b5c620e, 0x0e2f, 0x4c19, 0xa2, 0x41, 0x25, 0xbd, 0x47, 0x67, 0xbf, 0x3e	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.QueryMode – QueryMode() with unsupported *ModeNumber* value returns EFI_UNSUPPORTED	1. Call QueryMode() with each *ModeNumber* value less than MaxMode. If *ModeNumber* #1 (80*50) is unsupported, the return code should be EFI_UNSUPPORTED
5.6.2.4.4	0x5c444cd8, 0x3dce, 0x4be7, 0xb5, 0xcd, 0x39, 0x38, 0xd5, 0x04, 0xac, 0x95	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.QueryMode – *ModeNumber* #0 is supported and the dimension is 80 * 25	1. Call QueryMode() with each Mode value less than MaxMode. *ModeNumber* #0 should be supported and the dimension is 80 * 25
5.6.2.4.5	0x3b069c23, 0xde80, 0x4eb9, 0x86, 0x57, 0x48, 0x0f, 0x63, 0x81, 0x6c, 0x53	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.QueryMode – If *ModeNumber* #1 is supported, the dimension is 80 * 50	1. Call QueryMode() with each *ModeNumber* value less than MaxMode. If *ModeNumber* #1 is supported, the dimension should be 80 * 50
5.6.2.4.6	0x891cb899, 0xc05e, 0x4160, 0xa9, 0x8c, 0x06, 0x04, 0xc4, 0x0a, 0x44, 0x48	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.QueryMode – QueryMode() with Invalid *ModeNumber* Fields, EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.Mode does not change before and after	1. Call QueryMode() with Invalid Mode Fields beyond MaxMode. EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.Mode should not change before and after
5.6.2.4.7	0x8f0b6ebe, 0xaa65, 0x4aa4, 0x8c, 0xfc, 0x22, 0x08, 0x74, 0xe7, 0x95, 0x63	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.QueryMode – QueryMode() with Invalid Mode Fields returns EFI_UNSUPPORTED	1. Call QueryMode() with Invalid Mode Fields beyond MaxMode. The return code should be EFI_UNSUPPORTED

8.2.5 SetMode()

Number	GUID	Assertion	Test Description
5.6.2.5.1	0x3680c8c3, 0x8fc6, 0x4fe2, 0xa2, 0xdb, 0x4f, 0xcb, 0xe1, 0x0a, 0x14, 0x87	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetMode – SetMode()** with supported *ModeNumber* value sets mode and cursor postion	1. Call **SetMode()** with supported *ModeNumber* value. Cursor position should be set to (0,0). Current mode should be the appointed mode. Other attributes should remain unchanged.
5.6.2.5.2	0xcb1c6bc5, 0x6c12, 0x4d3a, 0x91, 0xc4, 0x2e, 0xdb, 0x09, 0xa3, 0x5d, 0x5f	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetMode – SetMode()** with supported *ModeNumber* value returns **EFI_SUCCESS**	1. Call **SetMode()** with supported *ModeNumber* value. The return code should be **EFI_SUCCESS**
5.6.2.5.3	0xab044f50, 0xd0d3, 0x44f5, 0x92, 0x34, 0xe0, 0x52, 0xcc, 0x26, 0x47, 0x89	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetMode – SetMode()** with Invalid Mode Fields, **EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.Mode** does not change before and after	1. Call **SetMode()** with Invalid Mode Fields beyond **MaxMode**. **EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.Mode** should not change before and after
5.6.2.5.4	0x6ce26a46, 0xab4a, 0x44df, 0x86, 0xc0, 0x3a, 0x97, 0xc3, 0xa3, 0x93, 0x0f	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetMode – SetMode()** with Invalid Mode Fields returns **EFI_UNSUPPORTED**	1. **SetMode()** with Invalid Mode Fields beyond **MaxMode**. The return code should be **EFI_UNSUPPORTED**.

8.2.6 SetAttribute()

Number	GUID	Assertion	Test Description
5.6.2.6.1	0xb401e101, 0x5386, 0x49fc, 0x89, 0x64, 0x54, 0x3b, 0xad, 0x90, 0x7b, 0x58	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetAttribute – SetAttribute()** with supported attributes returns **EFI_SUCCESS**	1. Call **SetAttribute()** with supported attributes. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.6.2.6.2	0x49b1f9ea, 0x085c, 0x4b2b, 0xa8, 0x98, 0x75, 0x6a, 0xa7, 0x61, 0x2f, 0x4a	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetAttribute – SetAttribute() should return EFI_SUCCESS with valid attributes	1. Check return status of SetAttribute() with valid attribute to set foreground color
5.6.2.6.3	0xefa8f25f, 0x60fe, 0x4707, 0x9f,0x2b, 0x66, 0x12, 0xf6, 0x4d, 0x3f, 0x6e	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetAttribute – SetAttribute() changes output color and remains other mode fields unchanged	1. Check all the fields of output mode. The background color and foreground color should be set as appointed value, and other fields should not be changed.
5.6.2.6.4	0x3af1e31e, 0x1523, 0x4ad3, 0xa0, 0x77, 0x51, 0xd2, 0x32, 0x8e, 0xdf, 0x80	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetAttribute – SetAttribute() with supported attributes returns EFI_SUCCESS and output color is set as expected	1. After the multiple calls of SetAttribute(), check all the return codes and changes in output mode fields.
5.6.2.6.5	0x42c6876b, 0x46e7, 0x47a5, 0xb4, 0x27, 0x25, 0x06, 0x1e, 0x25, 0xe8, 0xbf	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetAttribute – SetAttribute() with Invalid Attribute values, does not change EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.Mode.	1. Call SetAttribute() with Invalid Attribute values. EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.Mode should not be changed
5.6.2.6.6	0x300a1814, 0xd2c8, 0x4a51, 0xa9, 0x37, 0x0b, 0x8c, 0xe9, 0x3f, 0xb4, 0x45	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetAttribute – SetAttribute() with Invalid Attribute values returns EFI_UNSUPPORTED	1. Call SetAttribute() with Invalid Attribute values. The return code should be EFI_UNSUPPORTED.

8.2.7 ClearScreen()

Number	GUID	Assertion	Test Description

5.6.2.7.1	0xa92ce5f8, 0x89a8, 0x4695, 0xbc, 0xb1, 0x59, 0x3e, 0x0e, 0x88, 0xe2, 0x41	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.ClearScreen – ClearScreen()` sets cursor position to (0,0) and remain other attributes unchanged.	1. Call `ClearScreen()` in all supported modes. The cursor position should be set to (0,0), and other attributes of output should not be changed.
5.6.2.7.2	0xb3a0092f, 0xe768, 0x4359, 0xa9, 0xeb, 0x3d, 0x85, 0x27, 0x78, 0xc4, 0xcb	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.ClearScreen – ClearScreen()` returns `EFI_SUCCESS`.	1. Call `ClearScreen()` in all supported modes. The return code should be `EFI_SUCCESS`.

8.2.8 SetCursorPosition()

Number	GUID	Assertion	Test Description
5.6.2.8.1	0xe4f9fd56, 0x1e72, 0x44ee, 0xb0, 0x31, 0xae, 0xc6, 0xb4, 0xda, 0xb2, 0x0d	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetCursorPosition – SetCursorPosition()` moves cursor to appointed position and remain other attributes unchanged.	1. Call `SetCursorPosition()` in all supported modes to move cursor to every valid position within dimension boundary. The cursor position should be set to appointed value, and other attributes of output should not be changed.
5.6.2.8.2	0xbe56dc0d, 0x8779, 0x4700, 0xb4, 0x4c, 0x6d, 0xf4, 0x39, 0xfb, 0xf6, 0xaa	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetCursorPosition – SetCursorPosition()` returns `EFI_SUCCESS`.	1. Call `SetCursorPosition()` in all supported modes to move cursor to every valid position within dimension boundary. The return code should be `EFI_SUCCESS`.
5.6.2.8.3	0xa125b94f, 0xcbc6, 0x4e25, 0x80, 0x33, 0xfb, 0xf0, 0xde, 0x73, 0x14, 0x65	`EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetCursorPosition – SetCursorPosition()` returns `EFI_SUCCESS` and moves cursor to appointed position.	1. Call `SetCursorPosition()` in all supported modes to move cursor to every valid position within dimension boundary. 2. Check return code and behavior of each call.

Number	GUID	Assertion	Test Description
5.6.2.8.4	0x85e9aabd, 0x1376, 0x4e67, 0xb6, 0x14, 0xce, 0xcf, 0x63, 0x36, 0x9b, 0x31	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetCursorPosition – SetCursorPosition() with Invalid Row/ Column Numbers does not change cursor position.	1. Call SetCursorPosition() with Invalid Row/Column Numbers. Mode->CursorColumn / CursorRow should remain unchanged.
5.6.2.8.5	0xbeff2f08, 0xbc3e, 0x4e4f, 0xb8, 0x6f, 0x05, 0xb0, 0xe9, 0xd1, 0x0b, 0xa3	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.SetCursorPosition – SetCursorPosition() with Invalid Row/ Column Numbers returns EFI_UNSUPPORTED.	1. Call SetCursorPosition() with Invalid Row/Column Numbers. The return code should be EFI_UNSUPPORTED.

8.2.9 EnableCursor()

Number	GUID	Assertion	Test Description
5.6.2.9.1	0xdf85a087, 0xd1c9, 0x4739, 0x97, 0x2c, 0x4e, 0xd8, 0x61, 0x5f, 0x56, 0xd4	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.EnableCursor – EnableCursor() with TRUE returns EFI_SUCCESS or EFI_UNSUPPORTED	1. Call EnableCursor() with TRUE. If EnableCursor() is unsupported, the return code should be EFI_UNSUPPORTED
5.6.2.9.2	0x318fe413, 0xd07d, 0x4aad, 0x9c, 0x62, 0xf8, 0xfe, 0x7f, 0x77, 0xbe, 0xb2	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.EnableCursor – EnableCursor() with TRUE changes cursor status to visible.	1. Call EnableCursor() with TRUE. 2. If EnableCursor() success, CursorVisible should be TRUE
5.6.2.9.3	0x07394e57, 0xf2f5, 0x4045, 0x8b, 0x2c, 0x91, 0xbb, 0x2b, 0xe4, 0x3c, 0x4e	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.EnableCursor – EnableCursor() with TRUE returns EFI_SUCCESS or EFI_UNSUPPORTED	1. Call EnableCursor() with TRUE. If EnableCursor() is supported. The return code should be EFI_SUCCESS

Number	GUID	Assertion	Test Description
5.6.2.9.4	0xb3121d1b, 0xbd25, 0x477d, 0xad, 0xc3, 0x5d, 0xe3, 0x1b, 0x19, 0x43, 0x25	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.EnableCursor – EnableCursor()** with **FALSE** returns **EFI_SUCCESS** or **EFI_UNSUPPORTED**	1. Call **EnableCursor()** with **FALSE**. If **EnableCursor()** is unsupported, the return code should be **EFI_UNSUPPORTED**
5.6.2.9.5	0xcfd7fe8d, 0x1674, 0x4205, 0xb6, 0x3a, 0xe6, 0x4e, 0x86, 0x15, 0x66, 0x0c	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.EnableCursor – EnableCursor()** with **FALSE** changes cursor status to invisible.	1. Call **EnableCursor()** with **FALSE**. 2. If **EnableCursor()** success, *CursorVisible* should be **FALSE**
5.6.2.9.6	0x3f2b2512, 0x91cf, 0x44d9, 0xae, 0xbd, 0x89, 0x76, 0x40, 0xf1, 0xb4, 0x1f	**EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL.EnableCursor – EnableCursor()** with **FALSE** returns **EFI_SUCCESS** or **EFI_UNSUPPORTED**	1. Call **EnableCursor()** with **FALSE**. If **EnableCursor()** is supported. The return code should be **EFI_SUCCESS**

8.3 EFI_SIMPLE_POINTER_PROTOCOL_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_SIMPLE_POINTER_PROTOCOL Section.

8.3.1 Reset()

Number	GUID	Assertion	Test Description
5.6.4.1.1	0x3fcb89c6, 0xe504, 0x4669, 0xbf, 0x31, 0xba, 0x03, 0xb7, 0x66, 0xc8, 0xc2	**EFI_SIMPLE_POINTER_PROTOCOL.Reset – Reset()** with an *ExtendedVerification* value of **FALSE** returns **EFI_SUCCESS**	1. Call **Reset()** with an *ExtendedVerification* value of **FALSE**. The return code should be **EFI_SUCCESS**.
5.6.4.1.2	0xd752813f, 0x32dc, 0x4820, 0xb7, 0x59, 0xe8, 0x97, 0x0c, 0xf3, 0x33, 0x89	**EFI_SIMPLE_POINTER_PROTOCOL.Reset – GetState()** after **Reset()** returns 0 for all related movement.	1. Call **Reset()** with an *ExtendedVerification* value of **FALSE**. 2. Call **GetState()**. If success, *RelativeMovementX*, *RelativeMovementY* and *RelativeMovementZ* should be 0.

Number	GUID	Assertion	Test Description
5.6.4.1.3	0x716eefc7, 0x8c0a, 0x4636, 0xa0, 0xdb, 0x7e, 0x70, 0x20, 0xce, 0xe8, 0x5d	**EFI_SIMPLE_POINTER_PROTOCOL.Reset - GetState()** after **Reset()** returns **EFI_UNSUPPORTED**.	1. Call **Reset()** with an *ExtendedVerification* value of **FALSE**. 2. Call **GetState()**. The return code maybe **EFI_UNSUPPORTED**.
5.6.4.1.4	0xce6806f5, 0xe186, 0x4c24, 0x83, 0xaa, 0x00, 0x4f, 0xac, 0xf0, 0x28, 0x65	**EFI_SIMPLE_POINTER_PROTOCOL.Reset - Reset()** with an *ExtendedVerification* value of **TRUE** returns **EFI_SUCCESS**	1. Call **Reset()** with an *ExtendedVerification* value of **TRUE**. The return code should be **EFI_SUCCESS**
5.6.4.1.5	0xd3e54374, 0x17b6, 0x417b, 0xae, 0xc7, 0xcc, 0x55, 0xcc, 0x42, 0x35, 0xa2	**EFI_SIMPLE_POINTER_PROTOCOL.Reset - GetState()** after **Reset()** returns 0 for all related movement.	1. Call **Reset()** with an *ExtendedVerification* value of **TRUE**. 2. Call **GetState()**. If success, *RelativeMovementX, RelativeMovementY* and *RelativeMovementZ* should be 0.
5.6.4.1.6	0xd8a03978, 0x7023, 0x4d61, 0x92, 0xbd, 0x15, 0xd3, 0x9b, 0x3f, 0x5d, 0x11	**EFI_SIMPLE_POINTER_PROTOCOL.Reset - GetState()** after **Reset()** returns **EFI_UNSUPPORTED**.	1. Call **Reset()** with an *ExtendedVerification* value of **TRUE**. 2. Call **GetState()**. The return code maybe **EFI_UNSUPPORTED**.

8.3.2 GetState()

Number	GUID	Assertion	Test Description
5.6.4.2.1	0x5271062e, 0xdef9, 0x4d30, 0x84, 0x3b, 0x8d, 0x6e, 0x41, 0x33, 0x13, 0xf3	**EFI_SIMPLE_POINTER_PROTOCOL.GetState - GetState()** after **Reset()** returns 0 for all related movement.	1. Call **Reset()** with an *ExtendedVerification* value of **FALSE**. 2. Call **GetState()**. If success, *RelativeMovementX, RelativeMovementY* and *RelativeMovementZ* should be 0.

Number	GUID	Assertion	Test Description
5.6.4.2.2	0x7614c447, 0x12a0, 0x403d, 0x8a, 0xde, 0x98, 0x97, 0x51, 0x7d, 0xd8, 0x49	EFI_SIMPLE_POINTER_PROTOCOL.GetState - GetState() returns EFI_NOT_READY when there is no move since last call of GetState().	1. Call Reset() with an *ExtendedVerification* value of FALSE. 2. Call GetState(). 3. Call GetState() again, the return code should be EFI_NOT_READY.
5.6.4.2.3	0x2f8f8710, 0x02dd, 0x411f, 0xaa, 0xb5, 0x27, 0xe1, 0x3a, 0x6a, 0xb2, 0x79	EFI_SIMPLE_POINTER_PROTOCOL.GetState - GetState() after Reset() returns 0 for all related movement.	1. Call Reset() with an *ExtendedVerification* value of TRUE. 2. Call GetState(). If success, *RelativeMovementX, RelativeMovementY* and *RelativeMovementZ* should be 0.
5.6.4.2.4	0x3db7ea18 ,0xda9d, 0x4760, 0xa7,0x43, 0x04,0xb4, 0x8d,0x14, 0x4e,0x90	EFI_SIMPLE_POINTER_PROTOCOL.GetState - GetState() returns EFI_NOT_READY when there is no move since last call of GetState().	1. Call Reset() with an *ExtendedVerification* value of TRUE. 2. Call GetState(). 3. Call GetState() again, the return code should be EFI_NOT_READY.

8.4 EFI_SERIAL_IO_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_SERIAL_IO_PROTOCOL Section.

8.4.1 Reset()

Number	GUID	Assertion	Test Description
5.6.5.1.1	0x2e369256, 0x6c78, 0x49e9, 0x9e, 0xd5, 0xe3, 0xd2, 0x88, 0x34, 0x33, 0xa0	EFI_SERIAL_IO_PROTOCOL .Reset - Reset() returns EFI_SUCCESS.	1. Call Reset(). The return code should be EFI_SUCCESS.

8.4.2 SetAttributes()

Number	GUID	Assertion	Test Description
5.6.5.2.1	0x34260cb2, 0x43ae, 0x4853, 0x87, 0x4b, 0x47, 0x7c, 0xeb, 0x14, 0x42, 0x02	**EFI_SERIAL_IO_PROTOCOL .SetAttributes - SetAttributes()** with valid *BaudRate* returns **EFI_SUCCESS**.	1. Call **SetAttributes()** with various valid *BaudRate*. The return code should be **EFI_SUCCESS** and the *BaudRate* field of **Mode** should be equal to the set value.
5.6.5.2.2	0x3fd35bee, 0x5013, 0x472f, 0xa0, 0x08, 0xbd, 0xdf, 0x31, 0x9c, 0xe6, 0x6b	**EFI_SERIAL_IO_PROTOCOL .SetAttributes - SetAttributes()** with valid *ReceiveFifoDepth* returns **EFI_SUCCESS**.	1. Call **SetAttributes()** with various valid *ReceiveFifoDepth*. The return code should be **EFI_SUCCESS** and the *ReceiveFifoDepth* field of *Mode* should be equal to the set value.
5.6.5.2.3	0x8cf74222, 0x7134, 0x47b6, 0xa5, 0x82, 0xf4, 0xd9, 0xad, 0xa7, 0xa3, 0xf4	**EFI_SERIAL_IO_PROTOCOL .SetAttributes - SetAttributes()** with valid *Timeout* returns **EFI_SUCCESS**.	1. Call **SetAttributes()** with various valid *Timeout*. The return code should be **EFI_SUCCESS** and the *Timeout* field of *Mode* should be equal to the set value.
5.6.5.2.4	0x68f91273, 0x0078, 0x4e6c, 0xb9, 0xdb, 0x62, 0x59, 0xb5, 0x39, 0xf7, 0x4a	**EFI_SERIAL_IO_PROTOCOL .SetAttributes - SetAttributes()** with valid *Parity* returns **EFI_SUCCESS**.	1. Call **SetAttributes()** with various valid *Parity*. The return code should be **EFI_SUCCESS** and the *Parity* field of *Mode* should be equal to the set value.
5.6.5.2.5	0xdf6038c2, 0x3752, 0x4e22, 0xab, 0x4c, 0xfe, 0x66, 0x67, 0x0c, 0xa3, 0xdf	**EFI_SERIAL_IO_PROTOCOL .SetAttributes - SetAttributes()** with valid *DataBits* returns **EFI_SUCCESS**.	1. Call **SetAttributes()** with various valid *DataBits*. The return code should be **EFI_SUCCESS** and the *DataBits* field of *Mode* should be equal to the set value.
5.6.5.2.6	0xdf6f2692, 0x9a0d, 0x4b0f, 0xbc, 0x8e, 0x36, 0x8b, 0x6a, 0x03, 0xe0, 0xb1	**EFI_SERIAL_IO_PROTOCOL .SetAttributes - SetAttributes()** with valid *StopBits* returns **EFI_SUCCESS**.	1. Call **SetAttributes()** with various valid *StopBits*. The return code should be **EFI_SUCCESS** and the *StopBits* field of *Mode* should be equal to the set value.

Number	GUID	Assertion	Test Description
5.6.5.2.7	0xb199d5d2, 0x1143, 0x499e, 0xa5, 0xf8, 0xf0, 0xa7, 0x6f, 0x79, 0xfe, 0xe5	EFI_SERIAL_IO_PROTOCOL.SetAttributes - SetAttributes() with default attributes returns EFI_SUCCESS.	1. Call SetAttributes() with default attributes: BaudRate=115200; FifoDepth=1; Timeout=1000000; Parity=NoParity; DataBits=8; StopBits=OneStopBit; The return code should be EFI_SUCCESS
5.6.5.2.8	0x3041ec45, 0x00af, 0x4787, 0xb1, 0xe9, 0x15, 0xb8, 0x7a, 0xc5, 0xdd, 0xc8	EFI_SERIAL_IO_PROTOCOL.SetAttributes - SetAttributes() with nonstandard BaudRate values returns EFI_SUCCESS and set BaudRate as the nearest standard baud rate value.	1. Call SetAttributes() with nonstandard BaudRate values. The return code should be EFI_SUCCESS and the BaudRate field of Mode should be equal to the nearest standard baud rate value.
5.6.5.2.9	0x7a5cca70, 0x46c7, 0x4488, 0x87, 0x65, 0x84, 0x33, 0x66, 0x78, 0xa5, 0x01	EFI_SERIAL_IO_PROTOCOL.SetAttributes - SetAttributes() with unsupported BaudRate returns EFI_INVALID_PARAMETER.	1. Call SetAttributes() with unsupported BaudRate. The return code should be EFI_INVALID_PARAMETER.
5.6.5.2.10	0x190ca14d, 0xa6c2, 0x4a42, 0x86, 0x29, 0xa5, 0x14, 0x96, 0xc8, 0xe0, 0x52	EFI_SERIAL_IO_PROTOCOL.SetAttributes - SetAttributes() with unsupported ReceiveFifoDepth returns EFI_INVALID_PARAMETER.	1. Call SetAttributes() with unsupported ReceiveFifoDepth. The return code should be EFI_INVALID_PARAMETER.
5.6.5.2.11	0xd40c796b, 0xb654, 0x4fb5, 0x88, 0xb0, 0x1e, 0xc8, 0x2a, 0x27, 0x13, 0x50	EFI_SERIAL_IO_PROTOCOL.SetAttributes - SetAttributes() with unsupported Timeout returns EFI_INVALID_PARAMETER.	1. Call SetAttributes() with unsupported Timeout. The return code should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.6.5.2.12	0x15dc5ee1, 0x9871, 0x4e25, 0xb2, 0x22, 0xc5, 0x38, 0x5c, 0x9b, 0xf3, 0x6b	**EFI_SERIAL_IO_PROTOCOL .SetAttributes - SetAttributes()** with unsupported *Parity* returns **EFI_INVALID_PARAMETER**.	1. Call **SetAttributes()** with unsupported *Parity*. The return code should be **EFI_INVALID_PARAMETER**.
5.6.5.2.13	0x0aa15e38 , 0xb05c, 0x46cf, 0xb1, 0xf3, 0x1e, 0xb7, 0x41, 0x37, 0xb8, 0xbf	**EFI_SERIAL_IO_PROTOCOL .SetAttributes - SetAttributes()** with unsupported *DataBits* returns **EFI_INVALID_PARAMETER**.	1. Call **SetAttributes()** with unsupported *DataBits*. The return code should be **EFI_INVALID_PARAMETER**.
5.6.5.2.14	0x174a5c87, 0x74cf, 0x4e88, 0x84, 0x04, 0x68, 0x3e, 0xcb, 0x40, 0xf3, 0x2f	**EFI_SERIAL_IO_PROTOCOL .SetAttributes - SetAttributes()** with unsupported *StopBits* returns **EFI_INVALID_PARAMETER**.	1. Call **SetAttributes()** with unsupported *StopBits*. The return code should be **EFI_INVALID_PARAMETER**.

8.4.3 SetControl()

Number	GUID	Assertion	Test Description
5.6.5.3.1	0xac56dfb5, 0xce1c, 0x42a6, 0x98, 0xc9, 0xc6, 0xf5, 0xc8, 0xad, 0x83, 0xda	**EFI_SERIAL_IO_PROTOCOL .SetControl - SetControl()** with valid bits returns **EFI_SUCCESS** and **GetControl()** returns the set bits.	1. Call **SetControl()** with valid control bits. The return code should be **EFI_SUCCESS**. 2. Call **GetControl()**. The valid control bits should be set.
5.6.5.3.2	0x00605cbc, 0x3965, 0x4b61, 0xa2, 0x54, 0x2b, 0x2b, 0x72, 0x31, 0x72, 0xea	**EFI_SERIAL_IO_PROTOCOL .SetControl - SetControl()** with unsupported control bits returns **EFI_UNSUPPORTED**.	1. Call **SetControl()** with unsupported control bits. The return code should be **EFI_UNSUPPORTED**.

8.4.4 GetControl()

Number	GUID	Assertion	Test Description
5.6.5.4.1	0x131f5894, 0x1613, 0x4f3e, 0xbd, 0x45, 0x2b, 0xdd, 0xb7, 0xed, 0x22, 0xb0	`EFI_SERIAL_IO_PROTOCOL .GetControl - GetControl()` returns `EFI_SUCCESS` and gets the bits set by `SetControl()`.	1. Call `SetControl()` with valid control bits. 2. Call `GetControl()`.The return code should be `EFI_SUCCESS` and the valid control bits should be returned.
5.6.5.4.2	0xdd059dc5, 0x6558, 0x4d43, 0xac, 0x65, 0x58, 0xa6, 0x1d, 0x64, 0x8d, 0xb0	`EFI_SERIAL_IO_PROTOCOL .GetControl - GetControl()` returns `EFI_SUCCESS` and gets the bit of `EFI_SERIAL_INPUT_BUFFER_EMPTY` after buffer contents are read out.	1. Call `Read()` to read out buffer contents. 2. Call `GetControl()`.The return code should be `EFI_SUCCESS` and `EFI_SERIAL_INPUT_BUFFER_EMPTY` is set.

8.4.5 Write()

Number	GUID	Assertion	Test Description
5.6.5.5.1	0x72c50358, 0xc760, 0x4200, 0x8d, 0xb2, 0x09, 0x4d, 0x96, 0x84, 0x6f, 0x1a	`EFI_SERIAL_IO_PROTOCOL .Write - Write()` in software-loopback mode returns `EFI_SUCCESS` and `Read()` gets the same contents.	1. Call `Write()` in software-loopback mode. The return code should be `EFI_SUCCESS`. 2. Call `Read()` to get buffer. It should return the written contents.
5.6.5.5.2	0x688bf990, 0xfd8f, 0x430e, 0x8e, 0x1c, 0x78, 0x07, 0x2d, 0x74, 0xbd, 0x08	`EFI_SERIAL_IO_PROTOCOL .Write - Write()` in hardware-loopback mode returns `EFI_SUCCESS` and `Read()` gets the same contents.	1. Call `Write()` in hardware-loopback mode. The return code should be `EFI_SUCCESS`. 2. Call `Read()` to get buffer. It should return the written contents.
5.6.5.5.3	0x198873b8, 0xe8f2, 0x4bfd, 0xa0, 0x20, 0x36, 0xff, 0xb4, 0x93, 0x72, 0x02	`EFI_SERIAL_IO_PROTOCOL .Write - Write()` in non-loopback mode returns `EFI_SUCCESS`.	1. Call `Write()` in non-loopback mode. The return code should be `EFI_SUCCESS`.

8.4.6 Read()

Number	GUID	Assertion	Test Description
5.6.5.6.1	0x8ad0312f, 0x4cfc, 0x4611, 0xb7, 0x62, 0x85, 0x3a, 0xa3, 0x9d, 0x2f, 0xd9	**EFI_SERIAL_IO_PROTOCOL .Read - Read()** in software-loopback mode returns **EFI_SUCCESS** and gets the same contents written.	1. Call **Write()** in software-loopback mode. 2. Call **Read()** in software-loopback to get buffer. It should return the written contents. The return code should be **EFI_SUCCESS**.
5.6.5.6.2	0x76cb227f, 0x312d, 0x4476, 0x8c, 0x59, 0x6a, 0x98, 0x27, 0x5b, 0x62, 0x3d	**EFI_SERIAL_IO_PROTOCOL .Read - Read()** in hardware-loopback mode returns **EFI_SUCCESS** and gets the same contents written.	1. Call **Write()** in hardware-loopback mode. 2. Call **Read()** in hardware-loopback to get buffer. It should return the written contents. The return code should be **EFI_SUCCESS**.
5.6.5.6.3	0x3faefba1, 0x4049, 0x4868, 0x8f, 0x34, 0x59, 0xaf, 0x3e, 0x62, 0xdf, 0xb0	**EFI_SERIAL_IO_PROTOCOL .Read - Read()** in hardware-loopback mode without any characters in buffer returns **EFI_TIME_OUT** and set buffer size to 0.	1. Call **Read()** to read out all contents in buffer. 2. Call **Read()** again, the return code should be **EFI_TIME_OUT** and *BufferSize* should be 0.
5.6.5.6.4	0xc96db50e, 0xd269, 0x4fb0, 0x88, 0xbd, 0x6a, 0x02, 0x06, 0x66, 0x53, 0xa7	**EFI_SERIAL_IO_PROTOCOL .Read - Read()** in hardware-loopback mode with *BufferSize*=2 returns **EFI_TIME_OUT** when there is only 1 byte contents in serial buffer.	1. Call **Read()** to read out all contents in buffer. 2. Call **Write()** to write 1 byte into serial buffer. 3. Call **Read()** again with *BufferSize* = 2. The return code should be **EFI_TIME_OUT**, *BufferSize* should be 1 and 1 byte should be read.
5.6.5.6.5	0xb636572b, 0x7aaa, 0x4146, 0x8d, 0xd4, 0x18, 0xef, 0xac, 0xb4, 0x8a, 0x1a	**EFI_SERIAL_IO_PROTOCOL .Read - Read()** in software-loopback mode without any characters in buffer returns **EFI_TIME_OUT** and set buffer size to 0.	1. Call **Read()** to read out all contents in buffer. 2. Call **Read()** again, the return code should be **EFI_TIME_OUT** and *BufferSize* should be 0.
5.6.5.6.6	0x48050436, 0xc835, 0x4a24, 0x87, 0x75, 0x4d, 0x2e, 0x47, 0x88, 0xb5, 0x97	**EFI_SERIAL_IO_PROTOCOL .Read - Read()** in software-loopback mode with *BufferSize*=2 returns **EFI_TIME_OUT** when there is only 1 byte contents in serial buffer.	1. Call **Read()** to read out all contents in buffer. 2. Call **Write()** to write 1 byte into serial buffer. 3. Call **Read()** again with *BufferSize* = 2. The return code should be **EFI_TIME_OUT**, *BufferSize* should be 1 and 1 byte should be read.

8.5 EFI_GRAPHICS_OUTPUT_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_GRAPHICS_OUTPUT_PROTOCOL Section.

8.5.1 QueryMode()

Number	GUID	Assertion	Test Description
5.6.6.1.1	0xd1824539, 0x92cd, 0x434c, 0x81, 0x65, 0x87, 0x2c, 0xc2, 0x1a, 0x5f, 0x9e	**EFI_GRAPHICS_OUTPUT_PROTOCOL.QueryMode** – returns **EFI_SUCCESS** with valid parameter.	For valid graphics mode number from 0 to *MaxMode*-1: •Call **SetMode()** to switch the video device to the specified mode. •Call **QueryMode()** with the current mode to get the current info structure and check the content of Info. The returned status should be **EFI_SUCCESS**. For *PixelBlueGreenRedReserved8BitPerColor* or *PixelRedGreenBlueReserved8BitPerColor*, the *FrameBufferSize* should be *PixelsPerScanLine* * *VerticalResolution* * *PixelElementSize*.
5.6.6.1.2	0x82dfd41e, 0x49db, 0x4c86, 0x99, 0xbb, 0xc5, 0x74, 0x33, 0x4b, 0xa0, 0xc3	**EFI_GRAPHICS_OUTPUT_PROTOCOL.QueryMode** – Call **QueryMode()** with *MaxMode*.	1. Call **QueryMode()** with *MaxMode*. The returned status must be **EFI_INVALID_PARAMETER**. 2. Call **QueryMode()** with a *SizeOfInfo* value of **NULL**. The returned status must be **EFI_INVALID_PARAMETER**. 3. Call **QueryMode()** with an Info value of **NULL**. The returned status must be **EFI_INVALID_PARAMETER**. 4. For valid graphics mode number from 0 to *MaxMode*-1: •Call **QueryMode()** with the specified mode number. The returned status should be any value except **EFI_INVALID_PARAMETER**. •The called allocated buffer that the Info points to should not be **NULL.**

Number	GUID	Assertion	Test Description
5.6.6.1.3	0x8ebcd9ab, 0x69a9, 0x48a2, 0x9b, 0xbc, 0x8c, 0x47, 0x9e, 0x68, 0x91, 0x56	`EFI_GRAPHICS_OUTPUT_PROTOCOL.QueryMode` – Call `QueryMode()` with a *`SizeOfInfo`* value of `NULL`.	1. Call `QueryMode()` with *`MaxMode`*. The returned status must be `EFI_INVALID_PARAMETER`. 2. Call `QueryMode()` with a *`SizeOfInfo`* value of `NULL`. The returned status must be `EFI_INVALID_PARAMETER`. 3. Call `QueryMode()` with an Info *`value`* of `NULL`. The returned status must be `EFI_INVALID_PARAMETER`. 4. For valid graphics mode number from 0 to *`MaxMode`*-1: •Call `QueryMode()` with the specified mode number. The returned status should be any value except `EFI_INVALID_PARAMETER`. •The called allocated buffer that the Info points to should not be `NULL`
5.6.6.1.4	0x394e306b, 0x652a, 0x403a, 0xbd, 0x15, 0xdb, 0x9b, 0x46, 0xc3, 0x44, 0x3b	`ConsoleContro.QueryMode` – Call `QueryMode()` with an *`Info`* value of `NULL`.	1. Call `QueryMode()` with *`MaxMode`*. The returned status must be `EFI_INVALID_PARAMETER`. 2. Call `QueryMode()` with a *`SizeOfInfo`* value of `NULL`. The returned status must be `EFI_INVALID_PARAMETER`. 3. Call `QueryMode()` with an *`Info`* value of `NULL`. The returned status must be `EFI_INVALID_PARAMETER`. 4. For valid graphics mode number from 0 to *`MaxMode`*-1: •Call `QueryMode()` with the specified mode number. The returned status should be any value but `EFI_INVALID_PARAMETER`. •The called allocated buffer that the Info points to should not be `NULL`

Number	GUID	Assertion	Test Description
5.6.6.1.5	0xe7782dc5, 0x2b78, 0x460f, 0xb1, 0x02, 0x88, 0xd5, 0x12, 0x06, 0x45, 0x1f	**EFI_GRAPHICS_OUTPUT_PROTOCOL.QueryMode** – Call **QueryMode()** with a valid *ModeNumber*. The returned status should be **EFI_SUCCESS**.	1. Call **QueryMode()** with *MaxMode*. The returned status must be **EFI_INVALID_PARAMETER**. 2. Call **QueryMode()** with a *SizeOfInfo* value of **NULL**. The returned status must be **EFI_INVALID_PARAMETER**. 3. Call **QueryMode()** with an *Info* value of **NULL**. The returned status must be **EFI_INVALID_PARAMETER**. 4. For valid graphics mode number from 0 to *MaxMode*-1: •Call **QueryMode()** with the specified mode number. The returned status should be any value except **EFI_INVALID_PARAMETER**. •The called allocated buffer that the Info points to should not be **NULL**
5.6.6.1.6	0x486360f1, 0x6b8e, 0x48b5, 0x8b, 0xa8, 0xae, 0x40, 0xeb, 0x3b, 0x07, 0xa2	**EFI_GRAPHICS_OUTPUT_PROTOCOL.QueryMode** – Call **QueryMode()** with with valid parameters.	1. Call **QueryMode()** with *MaxMode*. The returned status must be **EFI_INVALID_PARAMETER**. 2. Call **QueryMode()** with a *SizeOfInfo* value of **NULL**. The returned status must be **EFI_INVALID_PARAMETER**. 3. Call **QueryMode()** with an *Info* value of **NULL**. The returned status must be **EFI_INVALID_PARAMETER**. 4. For valid graphics mode number from 0 to *MaxMode*-1: •Call **QueryMode()** with the specified mode number. The returned status should be any value except **EFI_INVALID_PARAMETER**. •The called allocated buffer that the Info points to should not be **NULL**

Number	GUID	Assertion	Test Description
5.6.6.1.7	0xdc19ab69, 0x764e, 0x429b, 0xa5, 0x3f, 0xb8, 0x1e, 0xd6, 0x3c, 0xd6, 0xc0	**EFI_GRAPHICS_OUTPUT_PROTOCOL.QueryMode** – Call **QueryMode()** to Check the mode structure and dump it.	1. Call **QueryMode()** with *MaxMode*. The returned status must be **EFI_INVALID_PARAMETER**. 2. Call **QueryMode()** with a *SizeOfInfo* value of **NULL**. The returned status must be **EFI_INVALID_PARAMETER**. 3. Call **QueryMode()** with an *Info* value of **NULL**. The returned status must be **EFI_INVALID_PARAMETER**. 4. For valid graphics mode number from 0 to *MaxMode*-1: •Call **QueryMode()** with the specified mode number. The returned status should be any value except **EFI_INVALID_PARAMETER**. •The called allocated buffer that the Info points to should not be **NULL**.

8.5.2 SetMode()

Number	GUID	Assertion	Test Description
5.6.6.2.1	0xb3a4939b, 0xd00a, 0x4da7, 0xaf, 0x6d, 0xf3, 0xee, 0xcb, 0xf9, 0x99, 0x0c	**EFI_GRAPHICS_OUTPUT_PROTOCOL.SetMode** – **SetMode()** returns **EFI_SUCCESS** when setting the graphics device and the set of active video output devices to the video mode specified by *ModeNumber*.	Call **SetMode()** with valid mode numbers from 0 to *MaxMode*-1.The returned status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.6.6.2.2	0x128e953b, 0xe6ec, 0x4f93, 0xa8, 0xec, 0x72, 0xc5, 0x9b, 0x8a, 0x40, 0x43	`EFI_GRAPHICS_OUTPUT_PROTOCOL.SetMode` – Call `SetMode()` with valid *ModeNumber*.	1. For valid graphics mode number from 0 to *MaxMode*-1: •Call `SetMode()` with valid mode, the returned status should not be `EFI_UNSUPPORTED` and should be `EFI_SUCCESS`. •Call `QueryMode()` with the same mode number as `SetMode()`. •The Info structure the `QueryMode()` returns and current mode of graphic device should be the same. 4. Call `SetMode()` with *MaxMode*. The returned status should be `EFI_UNSUPPORTED`.
5.6.6.2.3	0x4f13e7ba, 0xb35a, 0x4bf7, 0xb1, 0xc0, 0xfe, 0x39, 0x9c, 0x49, 0x97, 0xfe	`EFI_GRAPHICS_OUTPUT_PROTOCOL.SetMode` –Call `QueryMode()` with the *ModeNumber* the `SetMode()` set , then compare the Info structure `QueryMode()` returns with current mode of graphic device in order to verify whether they are same, and at last dump the Info structure.	1. For valid graphics mode number from 0 to *MaxMode*-1: •Call `SetMode()` with valid mode, the returned status should not be `EFI_UNSUPPORTED` and should be `EFI_SUCCESS`. •Call `QueryMode()` with the same mode number as `SetMode()`. •The Info structure the `QueryMode()` returns and current mode of graphic device should be the same. 2. Call `SetMode()` with *MaxMode*. The returned status should be `EFI_UNSUPPORTED`.
5.6.6.2.4	0x8776b9dc, 0x711e, 0x4e36, 0x99, 0x21, 0x7e, 0xa7, 0xc4, 0xc7, 0xee, 0x6d	`EFI_GRAPHICS_OUTPUT_PROTOCOL.SetMode` – Call `SetMode()` with valid *MaxMode*.	1. For valid graphics mode number from 0 to *MaxMode*-1: •Call `SetMode()` with valid mode, the returned status should not be `EFI_UNSUPPORTED` and should be `EFI_SUCCESS`. •Call `QueryMode()` with the same mode number as `SetMode()`. •The Info structure the `QueryMode()` returns and current mode of graphic device should be the same. 2. Call `SetMode()` with *MaxMode*. The returned status should be `EFI_UNSUPPORTED`.

8.5.3 Blt()

Number	GUID	Assertion	Test Description
5.6.6.3.1	0x95a44702, 0xcea0, 0x480f, 0x9f, 0x84, 0xe2, 0x4c, 0x17, 0xbf, 0x47, 0x79	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoFill** operation should fill graphic screen with pixels.	For valid graphics mode number from 0 to *MaxMode*-1: •Call **Blt()** with a BltOperation value of *EfiBltVideoFill* to write data from the *BltBuffer* pixel (0, 0) directly to every pixel of the video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of *EfiBltVideoToBltBuffer* to retrieve the rectangles drawn by the last *EfiBltVideoFill* operation. All the retrieved rectangles should be the same pixel used in the last *EfiBltVideoFill* operation. If pixels verification passes, the return status should be **EFI_SUCCESS**.
5.6.6.3.2	0x699c30b0, 0xab3f, 0x45d9, 0xbd, 0x69, 0x6b, 0x93, 0x96, 0xb7, 0x7e, 0x66	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoFill** operation should fill graphic screen with pixels.	For valid graphics mode number from 0 to *MaxMode*-1: •Call **Blt()** with a *BltOperation* value of *EfiBltVideoFill* to write data from the BltBuffer pixel (0, 0) directly to every pixel of the video display rectangle (*DestinationX*, *DestinationY*) (*DestinationX* +*Width*, *DestionationY* +*Height*). The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of *EfiBltVideoToBltBuffer* to retrieve the rectangles drawn by the last *EfiBltVideoFill* operation. All the retrieved rectangles should be the same pixel used in the last *EfiBltVideoFill* operation. If pixels verification passes, the return status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.6.6.3.3	0xc34c3fa4, 0xa61e, 0x4598, 0x9f, 0x80, 0x2d, 0xee, 0x8e, 0x2c, 0x9b, 0x57	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo** operation should write data to video screen and **EfiBltVideoToBltBuffer** operation should read data from video display rectangle.	For valid graphics mode number from 0 to *MaxMode*-1: •Call **Blt()** with a *BltOperation* value of *EfiBltVideoToBltBuffer* to read data from the video display rectangle (*SourceX*, *SourceY*) (SourceX+Width, SourceY+Height) and place it in the BltBuffer rectangle (DestinationX, DestinationY) (DestinationX+Width, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to write data from the *BltBuffer* rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) directly to the video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). – Source and Destination should reverse with that of **EfiBltVideoToBltBuffer** operation. The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0.The returned status should be **EFI_SUCCESS**. •The *BltBuffer* and BltBuffer2 should be the same.

Number	GUID	Assertion	Test Description
5.6.6.3.4	0x33a341ea, 0xc6a2, 0x4037, 0x8a, 0x2d, 0x19, 0xea, 0x1f, 0xe2, 0xf2, 0xa6	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo** operation should write data to video screen and **EfiBltVideoToBltBuffer** operation should read data from video display rectangle.	For valid graphics mode number from 0 to *MaxMode*-1: •Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (SourceX+Width, SourceY+Height) and place it in the BltBuffer rectangle (DestinationX, DestionationY) (DestinationX+Width, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to write data from the *BltBuffer* rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) directly to the video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). – Source and Destination should reverse with that of **EfiBltVideoToBltBuffer** operation. The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0.The returned status should be **EFI_SUCCESS**. •The *BltBuffer* and BltBuffer2 should be the same.

Number	GUID	Assertion	Test Description
5.6.6.3.5	0x13f113dc, 0xafd0, 0x4658, 0xb7, 0xfb, 0x83, 0xd5, 0xae, 0x6f, 0x10, 0x58	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVide** operation should write data to video screen and **EfiBltVideoToBltBuffer** operation should read data from video display rectangle.	For valid graphics mode number from 0 to *MaxMode*-1: •Call **Blt()** with a *BltOperation* value of *EfiBltVideoToBltBuffer* to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the *BltBuffer* rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to write data from the *BltBuffer* rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) directly to video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). – Source and Destination should reverse with that of **EfiVideoToBltBuffer** operation. The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0.The returned status should be **EFI_SUCCESS**. •The *BltBuffer* and BltBuffer2 should be the same.

Number	GUID	Assertion	Test Description
5.6.6.3.6	0x5ca291cc, 0x84a0, 0x489d, 0x9b, 0x2a, 0x0f, 0x2f, 0xcc, 0xc6, 0x0b, 0x29	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVide** operation should write data to video screen and **EfiBltVideoToBltBuffer** operation should read data from video display rectangle.	For valid graphics mode number from 0 to *MaxMode*-1: •Call **Blt()** with a *BltOperation* value of *EfiBltVideoToBltBuffer* to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the BltBuffer rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to write data from the *BltBuffer* rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) directly to video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). – Source and Destination should reverse with that of **EfiBltVideoToBltBuffer** operation. The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of *EfiBltVideoToBltBuffer* to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0.The returned status should be **EFI_SUCCESS**. •The BltBuffer and BltBuffer2 should be the same.

Number	GUID	Assertion	Test Description
5.6.6.3.7	0x6c2632c0, 0xe3de, 0x4afc, 0xb3, 0xa1, 0xbe, 0x50, 0x75, 0xab, 0x2d, 0x7a	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToVideo** operation should copy data from one video display rectangle to another.	For valid graphics mode number from 0 to *MaxMode*-1: •Call **Blt()** with a *BltOperation* value of *EfiBltVideoToBltBuffer* to save original screen and read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in *BltBuffer* rectangle (0, 0) (*Width*, *Height*) with a delta value of 0. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) to another display rectangle (*DestinationX*, *DestinationY*)(*DestinationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of *EfiBltVideoToBltBuffer* again to retrieve the area from the video display and read data from the video display rectangle (*DestinationX*, *DestinationY*) (*DestinationX*+*Width*, *DestinationY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0. •The BltBuffer and BltBuffer2 should be the same.

Number	GUID	Assertion	Test Description
5.6.6.3.8	0x07d1d0c1, 0x3884, 0x4310, 0x97, 0xbc, 0x16, 0xd6, 0xaa, 0x1a, 0x21, 0x80	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToVideo** operation should copy data from video display rectangle to another.	For valid graphics mode number from 0 to *MaxMode*-1: •Call **Blt()** with a *BltOperation* value of *EfiBltVideoToBltBuffer* to save original screen and read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in *BltBuffer* rectangle (0, 0) (*Width*, *Height*) with a delta value of 0. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) to another display rectangle (*DestinationX*, *DestinationY*)(*DestinationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. •Call **Blt()** with a *BltOperation* value of *EfiBltVideoToBltBuffer* again to retrieve the area from the video display and read data from the video display rectangle (*DestinationX*, *DestinationY*) (*DestinationX*+*Width*, *DestinationY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0. •The BltBuffer and BltBuffer2 should be the same.
5.6.6.3.9	0x11af616a, 0xbef5, 0x4590, 0xbe, 0x85, 0x19, 0x52, 0xa0, 0x0d, 0xe1, 0xaf	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt –**Call **Blt()** with invalid *BltOperation*.	Repeat the following step 6 times: •Call **Blt()** with a *BltOperation* value other than EfiBltVideoFill/ **EfiBltVideoToBltBuffer**/ **EfiBltBufferToVideo**/ **EfiBltVideoToVideo**.The returned status should be **EFI_INVALID_PARAMETER**. 3. Restore the screen mode.

Number	GUID	Assertion	Test Description
5.6.6.3.11	0xe967bdc7, 0xa0ea, 0x4fd7, 0xab, 0xba, 0x52, 0xf3, 0xef, 0x53, 0x22, 0x3e	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt** –Call **Blt()** to verify that the pixels **EfiBltVideoToBltBuffer** retrieves are the same as the pixels **EfiBltVideoFill** fills.	For valid graphics mode number from 0 to *MaxMode*-1: •To select a different valid parameter (*SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*, *Delta*): Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to write data from the *BltBuffer* pixel (0, 0) directly to every pixel of the video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to retrieve the rectangles drawn by the last **EfiBltVideoFill** operation. The returned status should be **EFI_SUCCESS**. All of the retrieved rectangles should be the same pixel used in the last **EfiBltVideoFill** operation.
5.6.6.3.12	0x1fc521b0, 0x63c1, 0x4f42, 0xb8, 0x14, 0x06, 0x8a, 0x6c, 0x9c, 0x3e, 0x29	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt** – **EfiBltVideoFill** operation should fill the graphic screen with pixels.	For valid graphics mode number from 0 to *MaxMode*-1: •To select a different valid parameter (*SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*, *Delta*): Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to write data from the *BltBuffer* pixel (0, 0) directly to every pixel of the video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to retrieve the rectangles drawn by the last **EfiBltVideoFill** operation. The returned status should be **EFI_SUCCESS**. All of the retrieved rectangles should be the same pixel used in the last **EfiBltVideoFill** operation.

Number	GUID	Assertion	Test Description
5.6.6.3.13	0x04fd0571, 0xf3eb, 0x4d69, 0xb2, 0xd2, 0x5c, 0x4f, 0xfb, 0x10, 0x5a, 0xc3	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToBltBuffer** operation should retrieve the pixels from the video memory to buffer.	For valid graphics mode number from 0 to *MaxMode*-1: •To select a different valid parameter (*SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*, *Delta*): Call **Blt()** with a *BltOperation* value **EfiBltVideoFill** to write data from the *BltBuffer* pixel (0, 0) directly to every pixel of the video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to retrieve the rectangles drawn by the last **EfiBltVideoFill** operation. The returned status should be **EFI_SUCCESS**. All of the retrieved rectangles should be the same pixel used in the last **EfiBltVideoFill** operation.
5.6.6.3.14	0x5bee154c, 0xe519, 0x4be4, 0xaf, 0x8c, 0xb4, 0x18, 0x8e, 0x79, 0xb4, 0xbf	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo** operation should draw the bitmap from the specified buffer to the video screen.	For valid graphics mode number from 0 to *MaxMode*-1: • To select a different valid parameter (*SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*, *Delta*): Load a bitmap from the prepared buffer and call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display it in the video. The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the BltBuffer rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
			Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to write data from the BltBuffer rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) directly to video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). – Source and Destination should reverse with that of **EfiBltVideoToBltBuffer** operation. The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0.The returned status should be **EFI_SUCCESS**. The BltBuffer and BltBuffer2 should be the same.
5.6.6.3.15	0xf9e726c1, 0x1346, 0x419e, 0x90, 0x8a, 0x66, 0xc4, 0x49, 0x8c, 0xfd, 0x71	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToBltBuffer** operation should retrieve the pixels from the video to the buffer *BltBuffer*.	For valid graphics mode number from 0 to *MaxMode*-1: •To select a different valid parameter (*SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*, *Delta*): Load a bitmap from the prepared buffer and call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display it in the video. The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the *BltBuffer* rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
			Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to write data from the *BltBuffer* rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) directly to video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). – Source and Destination should be the reverse of the **EfiBltVideoToBltBuffer** operation. The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0.The returned status should be **EFI_SUCCESS**. The BltBuffer and BltBuffer2 should be the same.
5.6.6.3.16	0x00f74a1b, 0x4599, 0x45b7, 0xb6, 0xf7, 0x13, 0xf2, 0xcb, 0xd8, 0x6c, 0xe6	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo** operation should write data to the video screen.	For valid graphics mode number from 0 to *MaxMode*-1: •To select a different valid parameter (*SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*, *Delta*): Load a bitmap from the prepared buffer and call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display it in the video. The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the *BltBuffer* rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
			Call `Blt()` with a *`BltOperation`* value of **`EfiBltBufferToVideo`** to write data from the *`BltBuffer`* rectangle (*`SourceX`*, *`SourceY`*) (*`SourceX`*+*`Width`*, *`SourceY`*+*`Height`*) directly to the video display rectangle (*`DestinationX`*, *`DestionationY`*) (*`DestionationX`*+*`Width`*, *`DestionationY`*+*`Height`*). – Source and Destination should be the reverse of the **`FeiBltVideoToBltBuffer`** operation. The returned status should be **`EFI_SUCCESS`**. Call `Blt()` with a *`BltOperation`* value of **`EfiBltVideoToBltBuffer`** to read data from the video display rectangle (*`SourceX`*, *`SourceY`*) (*`SourceX`*+*`Width`*, *`SourceY`*+*`Height`*) and place it in the BltBuffer2 rectangle (0, 0) (*`Width`*, *`Height`*) with a *`Delta`* value of 0.The returned status should be **`EFI_SUCCESS`**. The BltBuffer and BltBuffer2 should be the same.
5.6.6.3.17	0x26da6582, 0x8b82, 0x4bd2, 0xac, 0x3a, 0x6e, 0x37, 0x85, 0x4f, 0xd8, 0x21	**`EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToBltBuffer`** operation should retrieve the pixels from the video to another buffer BltBuffer2.	For valid graphics mode number from 0 to *`MaxMode`*-1: •To select a different valid parameter (*`SourceX`*, *`SourceY`*, *`DestinationX`*, *`DestinationY`*, *`Width`*, *`Height`*, *`Delta`*): Load a bitmap from the prepared buffer and call `Blt()` with a *`BltOperation`* value of **`EfiBltBufferToVideo`** to display it in the video. The returned status should be **`EFI_SUCCESS`**. Call `Blt()` with a *`BltOperation`* value of **`EfiBltVideoToBltBuffer`** to read data from the video display rectangle (*`SourceX`*, *`SourceY`*) (*`SourceX`*+*`Width`*, *`SourceY`*+*`Height`*) and place it in the *`BltBuffer`* rectangle (*`DestinationX`*, *`DestionationY`*) (*`DestionationX`*+*`Width`*, *`DestionationY`*+*`Height`*). The returned status should be **`EFI_SUCCESS`**.

Number	GUID	Assertion	Test Description
			Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to write data from the BltBuffer rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) directly to the video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). – Source and Destination should be the reverse of the **EfiBltVideoToBltBuffer** operation. The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a *Delta* value of 0.The returned status should be **EFI_SUCCESS**. The BltBuffer and BltBuffer2 should be the same.
5.6.6.3.18	0x0aaf7f4e, 0x1794, 0x403c, 0xb3, 0xb0, 0x18, 0xf5, 0xe4, 0xd3, 0xc4, 0xea	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt –**Verify if the pixels retrieved from the first operation of **EfiBltVideoToBltBuffer** are the same as the pixels retrieved from the second operation.	For valid graphics mode number from 0 to *MaxMode*-1: •To select a different valid parameter (*SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*, *Delta*): Load a bitmap from the prepared buffer and call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display it in the video. The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the *BltBuffer* rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
			Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to write data from the BltBuffer rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) directly to the video display rectangle (*DestinationX*, *DestionationY*) (*DestionationX*+*Width*, *DestionationY*+*Height*). – Source and Destination should be the reverse of the **EfiBltVideoToBltBuffer** operation. The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0.The returned status should be **EFI_SUCCESS**. The BltBuffer and BltBuffer2 should be the same.
5.6.6.3.19	0x2a79335b, 0xafc3, 0x4ccf, 0x9b, 0xa4, 0x91, 0x9b, 0xe4, 0xb8, 0xbe, 0xfc	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo** operation should draw the bitmap from the specified buffer to the video screen.	For valid graphics mode number from 0 to *MaxMode*-1: •To select a different valid parameter (*SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*, *Delta*): Load a bitmap from the prepared buffer and call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display it in the video. The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to save the original screen and read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the *BltBuffer* rectangle (0, 0) (*Width*, *Height*) with a delta value of 0.The returned status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
			Call `Blt()` with a *`BltOperation`* value of EfiBltVideoToVideo to copy data from the video display rectangle (*`SourceX`*, *`SourceY`*) (*`SourceX`*+*`Width`*, *`SourceY`*+*`Height`*) to another display rectangle (*`DestinationX`*, *`DestinationY`*)(*`DestinationX`*+*`Width`*, *`DestionationY`*+*`Height`*). The returned status should be **`EFI_SUCCESS`**. Call `Blt()` with a *`BltOperation`* value of **`EfiBltVideoToBltBuffer`** again to retrieve the area from the video display and read data from the video display rectangle (*`DestinationX`*, *`DestinationY`*) (*`DestinationX`*+*`Width`*, *`DestinationY`*+*`Height`*) and place it in BltBuffer2 rectangle (0, 0) (*`Width`*, *`Height`*) with a delta value of 0. The returned status should be **`EFI_SUCCESS`**. The BltBuffer and BltBuffer2 should be the same.
5.6.6.3.20	0x3f4c2c88, 0xa1f8, 0x46f5, 0x9e, 0x5e, 0x67, 0x50, 0xb4, 0xae, 0x2b, 0x6f	**`EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToBltBuffer`** operation should retrieve the pixels from the video display rectangle to the buffer *`BltBuffer`*.	For valid graphics mode number from 0 to *`MaxMode`*-1: •To select a different valid parameter (*`SourceX`*, *`SourceY`*, *`DestinationX`*, *`DestinationY`*, *`Width`*, *`Height`*, *`Delta`*): Load a bitmap from the prepared buffer and call `Blt()` with a *`BltOperation`* value of **`EfiBltBufferToVideo`** to display it in the video. The returned status should be **`EFI_SUCCESS`**. Call `Blt()` with a *`BltOperation`* value of **`EfiBltVideoToBltBuffer`** to save the original screen and read data from the video display rectangle (*`SourceX`*, *`SourceY`*) (*`SourceX`*+*`Width`*, *`SourceY`*+*`Height`*) and place it in the *`BltBuffer`* rectangle (0, 0) (*`Width`*, *`Height`*) with a delta value of 0.The returned status should be **`EFI_SUCCESS`**.

Number	GUID	Assertion	Test Description
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) to another display rectangle (*DestinationX*, *DestinationY*)(*DestinationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** again to retrieve the area from the video display and read data from the video display rectangle (*DestinationX*, *DestinationY*) (*DestinationX*+*Width*, *DestinationY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0. The returned status should be **EFI_SUCCESS**. The BltBuffer and BltBuffer2 should be the same.
5.6.6.3.21	0xa11dd47e, 0xf144, 0x460c, 0x9e, 0x18, 0x7e, 0xb7, 0xed, 0xda, 0xc0, 0x18	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToVideo** operation should copy data from one video display rectangle to another.	For valid graphics mode number from 0 to *MaxMode*-1: •To select a different valid parameter (*SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*, *Delta*): Load a bitmap from the prepared buffer and call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display it in the video. The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to save the original screen and read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the *BltBuffer* rectangle (0, 0) (*Width*, *Height*) with a delta value of 0.The returned status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
			Call **Blt()** with a *BltOperation* value of EfiBltVideoToVideo to copy data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) to another display rectangle (*DestinationX*, Dest*i*nationY) (*DestinationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** again to retrieve the area from the video display and read data from the video display rectangle (*DestinationX*, *DestinationY*) (*DestinationX*+*Width*, *DestinationY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0. The returned status should be **EFI_SUCCESS**. The BltBuffer and BltBuffer2 should be the same.
5.6.6.3.22	0xbe3e3046, 0x5aea, 0x48d0, 0x91, 0xc4, 0x62, 0xce, 0xff, 0x61, 0x3c, 0xec	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToBltBuffer** operation should retrieve the pixels from the video display rectangle to another buffer BltBuffer2.	For valid graphics mode number from 0 to *MaxMode*-1: •To select a different valid parameter (*SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*, *Delta*): Load a bitmap from the prepared buffer and call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display it in the video. The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to save the original screen and read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the *BltBuffer* rectangle (0, 0) (*Width*, *Height*) with a delta value of 0.The returned status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
			Call `Blt()` with a *BltOperation* value of **EfiBltVideoToVideo** to copy data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) to another display rectangle (*DestinationX*, *DestinationY*)(*DestinationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. Call `Blt()` with a *BltOperation* value of **EfiBltVideoToBltBuffer** again to retrieve the area from the video display and read data from the video display rectangle (*DestinationX*, *DestinationY*) (*DestinationX*+*Width*, *DestinationY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0. The returned status should be **EFI_SUCCESS**. The BltBuffer and BltBuffer2 should be the same.
5.6.6.3.23	0xed4e402a, 0x403c, 0x4071, 0x86, 0x93, 0x9d, 0x8d, 0x28, 0xf7, 0x83, 0xd9	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt** –Verify that the pixels **EfiBltVideoToBltBuffer** retrieves are the same as the ones the second operation retrieves.	For valid graphics mode number from 0 to *MaxMode*-1: •To select a different valid parameter (*SourceX*, *SourceY*, *DestinationX*, *DestinationY*, *Width*, *Height*, *Delta*): Load a bitmap from the prepared buffer and call `Blt()` with a *BltOperation* value of **EfiBltBufferToVideo** to display it in the video. The returned status should be **EFI_SUCCESS**. Call `Blt()` with a *BltOperation* value **EfiBltVideoToBltBuffer** to save the original screen and read data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) and place it in the *BltBuffer* rectangle (0, 0) (*Width*, *Height*) with a delta value of 0.The returned status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy data from the video display rectangle (*SourceX*, *SourceY*) (*SourceX*+*Width*, *SourceY*+*Height*) to another display rectangle (*DestinationX*, *DestinationY*)(*DestinationX*+*Width*, *DestionationY*+*Height*). The returned status should be **EFI_SUCCESS**. Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** again to retrieve the area from the video display and read data from the video display rectangle (*DestinationX*, *DestinationY*) (*DestinationX*+*Width*, *DestinationY*+*Height*) and place it in the BltBuffer2 rectangle (0, 0) (*Width*, *Height*) with a delta value of 0. The returned status should be **EFI_SUCCESS**. The BltBuffer and BltBuffer2 should be the same.
5.6.6.3.24	0x3b54894e, 0x6383, 0x4dd5, 0x9e, 0x53, 0xbe, 0x6b, 0xc1, 0x1b, 0xd8, 0x94	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo** operation should draw the bitmap from the specified buffer to the video screen.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. •Repeat the following actions with 2 times: change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise. Call **Blt()** with a *BltOperation* value **EfiBltBufferToVideo** to display the BMP file stored in *BltBuffer*. The returned status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
			Call `Blt()` with a *BltOperation* value `EfiBltVideoFill` to clear the rectangle in the last EfiBltBufferToVideo operation. The returned status should be `EFI_SUCCESS`. **Note:** The rotation of (*DestinationX*, *DestinationY*) may have some distance between the two coordinates. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should not exceed the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.
5.6.6.3.25	0xd0869ac8, 0x1d16, 0x4657, 0xae, 0xf2, 0x06, 0xc3, 0x49, 0x82, 0x1d, 0x55	`EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoFill` operation should fill the graphic screen with pixels.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare *BltBuffer* from a small BMP file. •Call `Blt()` with a *BltOperation* value of `EfiBltVideoFill` to clear the display. •Repeat following actions with 2 times: change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise. Call `Blt()` with a *BltOperation* value of `EfiBltBufferToVideo` to display the BMP file stored in *BltBuffer*. The returned status should be `EFI_SUCCESS`.
			Call `Blt()` with a *BltOperation* value of `EfiBltVideoFill` to clear the rectangle in the last `EfiBltBufferToVideo` operation. The returned status should be `EFI_SUCCESS`. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.26	0x1f026b26, 0x36fd, 0x4f1c, 0x95, 0x4c, 0x16, 0x0f, 0x9f, 0x98, 0x49, 0xd1	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo** operation should draw the bitmap from the specified buffer to the video screen.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. •Repeat following actions with 2 times: change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise. Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file stored in *BltBuffer*. The returned status should be **EFI_SUCCESS**.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the rectangle in the last **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.27	0xd0bfb3c3, 0x54df, 0x4c07, 0x8e, 0x5c, 0x7a, 0x19, 0xa3, 0x5b, 0x5c, 0x0c	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoFill** operation should fill the graphic screen with pixels.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. •Repeat following actions with 2 times: change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise. Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file stored in *BltBuffer*. The returned status should be **EFI_SUCCESS**.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the rectangle in the last **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP files rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.28	0xfde7edd9, 0x1486, 0x45e9, 0xae, 0x06, 0x31, 0xe8, 0xcb, 0x3f, 0xf3, 0x46	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo** operation should draw the bitmap from the specified buffer to the video screen.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* Value of **EfiBltVideoFill** to clear the display. •Repeat following actions with 2 times: change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise. Call **Blt()** with a *BltOperation* value **EfiBltBufferToVideo** to display the BMP file stored in *BltBuffer*. The returned status should be **EFI_SUCCESS**.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the rectangle in the last **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.29	0x538471f3, 0x8828, 0x4d1b, 0x8c, 0x2b, 0x01, 0x37, 0xe9, 0x4f, 0xae, 0xc9	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoFill** operation should fill the graphic screen with pixels.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare BltBuffer from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. •Repeat following actions with 2 times: change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise. Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file stored in *BltBuffer*. The returned status should be **EFI_SUCCESS**.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the rectangle in the last **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.30	0x30ef55c6, 0x62a2, 0x4f90, 0xb3, 0xf8, 0xf4, 0xf9, 0x1b, 0x94, 0xbf, 0x91	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo** operation should draw the bitmap from the specified buffer to the video screen.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare BltBuffer from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. •Repeat following actions with 2 times: change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise. Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file stored in *BltBuffer*. The returned status should be **EFI_SUCCESS**.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the rectangle in the last **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.31	0x2bb7feeb, 0x9b15, 0x4b27, 0x92, 0x61, 0xff, 0xa6, 0x9e, 0xcf, 0x0a, 0x00	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoFill** operation should fill the graphic screen with pixels.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. •Repeat following actions, change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise: Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file stored in *BltBuffer*. The returned status should be **EFI_SUCCESS**.
			Call **Blt()** with a *BltOperation* value **EfiBltVideoFill** to clear the rectangle in the last call of **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** The rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.32	0x3bb9ebcc, 0x370a, 0x4c02, 0xb2, 0x0d, 0x1f, 0x86, 0x5a, 0x98, 0xaa, 0x15	**`EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo`** operation should draw the bitmap from the specified buffer to the video screen.	For valid graphics mode number from 0 to *`MaxMode`*-1: •Prepare some *`BltBuffer`* from a small BMP file. •Call **`Blt()`** with a *`BltOperation`* value of **`EfiBltVideoFill`** to clear the display. •Repeat following actions, change the (*`DestinationX`*,*`DestinationY`*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*`DestinationX`*,*`DestinationY`*) clockwise: Call **`Blt()`** with a *`BltOperation`* value of **`EfiBltBufferToVideo`** to display the BMP file stored in *`BltBuffer`*. The returned status should be **`EFI_SUCCESS`**.
			Call **`Blt()`** with a *`BltOperation`* value of **`EfiBltVideoFill`** to clear the rectangle in the last **`EfiBltBufferToVideo`** operation. The returned status should be **`EFI_SUCCESS`**. **Note:** The rotation of (*`DestinationX`*, *`DestinationY`*) may have some distance between two coordinate. In addition, the (*`DestinationX`*, *`DestinationY`*,*`Width`*, *`Height`*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.33	0xb904f2be, 0x720e, 0x4d9b, 0x86, 0x72, 0xd7, 0x84, 0x6b, 0xbc, 0x53, 0xea	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo** operation should draw the bitmap from the specified buffer to the video screen.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. The returned status should be **EFI_SUCCESS**, and the display should be cleaned. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file. •Repeat following actions, change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise: Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy the video drawn by the last **EfiBltVideoToVideo** or **fiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** The two rectangles from the EfiBltVideoToVideo operation should not overlap.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the source rectangle in the last **EfiBltVideoToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** The rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.34	0x53748ffc, 0xaff8, 0x4cc9, 0x83, 0xab, 0xc7, 0x09, 0xe1, 0x59, 0x1c, 0xed	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToVideo** operation should copy data from one video display rectangle to another.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. The returned status should be **EFI_SUCCESS**, and the display should be cleaned. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file. •Repeat following actions, change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise: Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy the video drawn by the last **EfiBltVideoToVideo** or **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** The two rectangles from the **EfiBltVideoToVideo** operation should not overlap.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the source rectangle in the last call of EfiBltVideoToVideo operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.35	0x4acd2d08, 0x01dd, 0x411f, 0xa6, 0xe2, 0xf3, 0x6f, 0x9f, 0x4b, 0x03, 0xb0	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoFill** operation should fill the graphic screen with pixels.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. The returned status should be **EFI_SUCCESS**, and the display should be cleaned. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file. •Repeat following actions, change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise: Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy the Video drawn by the last **EfiBltVideoToVideo** or **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** The two rectangles from the **EfiBltVideoToVideo** operation should not overlap.
			Call **Blt()** with a *BltOperation* value of *EfiBltVideoFill* to clear the source rectangle in the last EfiBltVideoToVideo operation. The returned status should be **EFI_SUCCESS**. **Note:** The rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.36	0xb11e8ade, 0x0c54, 0x4963, 0x89, 0x66, 0xa0, 0x4a, 0x50, 0x40, 0x1c, 0x7b	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToVideo** operation should copy data from one video display rectangle to another.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. The returned status should be **EFI_SUCCESS**, and the display should be cleaned. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file. •Repeat following actions, change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to lower the left-hand corner, then to upper the left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise: Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy the Video drawn by the last **EfiBltVideoToVideo** or **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** The two rectangles from the **EfiBltVideoToVideo** operation should not overlap.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the source rectangle in the last **EfiBltVideoToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.37	0xfa43d810, 0x7501, 0x481f, 0xbd, 0xcd, 0xc1, 0x06, 0x57, 0x94, 0x84, 0x9a	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoFill** operation should fill the graphic screen with pixels.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. The returned status should be **EFI_SUCCESS**, and the display should be cleaned. •Call **Blt()** with a *BltOperation* vlue of **EfiBltBufferToVideo** to display the BMP file. •Repeat following actions, change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to upper the left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise: Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy the Video drawn by the last **EfiBltVideoToVideo** or **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** The two rectangles from the **EfiBltVideoToVideo** operation should not overlap.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the source rectangle in the last **EfiBltVideoToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.38	0x94989a37, 0x3941, 0x4cd8, 0x97, 0x0b, 0x14, 0xfa, 0x46, 0xb6, 0x07, 0x16	**`EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToVideo`** operation should copy data from one video display rectangle to another.	For valid graphics mode number from 0 to *`MaxMode`*-1: •Prepare some *`BltBuffer`* from a small BMP file. •Call **`Blt()`** with a *`BltOperation`* value of **`EfiBltVideoFill`** to clear the display. The returned status should be **`EFI_SUCCESS`**, and the display should be cleaned. •Call **`Blt()`** with a *`BltOperation`* value of **`EfiBltBufferToVideo`** to display the BMP file. •Repeat following actions, change the (*`DestinationX`*,*`DestinationY`*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*`DestinationX`*,*`DestinationY`*) clockwise: Call **`Blt()`** with a *`BltOperation`* value of **`EfiBltVideoToVideo`** to copy the Video drawn by the last **`EfiBltVideoToVideo`** or **`EfiBltBufferToVideo`** operation. The returned status should be **`EFI_SUCCESS`**. **Note:** The two rectangles from the **`EfiBltVideoToVideo`** operation should not overlap.
			Call **`Blt()`** with a *`BltOperation`* value of **`EfiBltVideoFill`** to clear the source rectangle in the last call of **`EfiBltVideoToVideo`** operation. The returned status should be **`EFI_SUCCESS`**. **Note:** the rotation of (*`DestinationX`*, *`DestinationY`*) may have some distance between two coordinate. In addition, the (*`DestinationX`*, *`DestinationY`*, *`Width`*, *`Height`*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.39	0x4dde309d, 0xaf32, 0x4a35, 0x91, 0x5a, 0x41, 0xcb, 0xb0, 0x18, 0x7c, 0x29	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoFill** operation should fill the graphic screen with pixels.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. The returned status should be **EFI_SUCCESS**, and the display shoul be cleaned. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file. •Repeat following actions, change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise: Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy the Video drawn by the last **EfiBltVideoToVideo** or **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** The two rectangles from the **EfiBltVideoToVideo** operation should not overlap.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the source rectangle in the last **EfiBltVideoToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.40	0xaa6b7386, 0x0537, 0x4762, 0xa1, 0x43, 0xca, 0xde, 0xb7, 0x55, 0x15, 0xc7	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToVideo** operation should copy data from one video display rectangle to another.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. The returned status should be **EFI_SUCCESS**, and the display should be cleaned. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file. •Repeat following actions, change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise: Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy the Video drawn by the last **EfiBltVideoToVideo** or **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** The two rectangles from the **EfiBltVideoToVideo** operation should not overlap.
			Call **Blt()** with a *BltOperation* value **EfiBltVideoFill** to clear the source rectangle in the last **EfiBltVideoToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.41	0xb751208f, 0x10eb, 0x47eb, 0x9c, 0x73, 0x15, 0x08, 0xb8, 0xc9, 0xcd, 0xbe	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoFill** operation should fill the graphic screen with pixels.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. The returned status should be **EFI_SUCCESS**, and the display should be cleaned. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file. •Repeat following actions, change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise: Call **Blt()** with a *BltOperation* value **EfiBltVideoToVideo** to copy the Video drawn by the last **EfiBltVideoToVideo** or **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** The two rectangles from **theEfiBltVideoToVideo** operation should not overlap.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the source rectangle in the last **EfiBltVideoToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt the user to judge whether the BMP file rotates correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.42	0x57b7debf, 0xb831, 0x40d1, 0x8b, 0xa0, 0xa6, 0x57, 0x7b, 0x92, 0xe2, 0x53	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt** –check logo rotatation correctly from user's view.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the display. The returned status should be **EFI_SUCCESS**, and the display should be cleaned. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display the BMP file. •Repeat following actions, change the (*DestinationX*,*DestinationY*) from the upper left-hand corner to the upper right-hand corner, then down to the lower right-hand corner, then to the lower left-hand corner, then to the upper left-hand corner, i.e., rotate the (*DestinationX*,*DestinationY*) clockwise: Call **Blt()** with a *BltOperation* value of **EfiBltVideoToVideo** to copy the Video drawn by the last **EfiBltVideoToVideo** or **EfiBltBufferToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** The two rectangles from the **EfiBltVideoToVideo** operation should not overlap.
			Call **Blt()** with a *BltOperation* value of **EfiBltVideoFill** to clear the source rectangle in the last call of **EfiBltVideoToVideo** operation. The returned status should be **EFI_SUCCESS**. **Note:** the rotation of (*DestinationX*, *DestinationY*) may have some distance between two coordinate. In addition, the (*DestinationX*, *DestinationY*, *Width*, *Height*) should avoid exceeding the boundary of the current screen mode. •Prompt user to judge whether the rotation of the BMP file correctly.

Number	GUID	Assertion	Test Description
5.6.6.3.43	0x8971c5fe, 0x02c6, 0x4ada, 0xab, 0x30, 0x36, 0xc5, 0xa7, 0xd9, 0xdc, 0x01	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltBufferToVideo** operation should draw the bitmap from the specified buffer to the video screen.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display BMP file to some position of the screen. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to retrieve the whole screen to a large *BltBuffer*. The return status should be **EFI_SUCCESS**. •Change *SourceX* from 0 to HorizontalResolution step by step, and change *SourceY* from 0 to VerticalResolution step by step, carry out following action: For small *Width*, *Height*, and *BltBuffer*, call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer**. The returned status should be **EFI_SUCCESS**. The small *BltBuffer* retrieved should be the same as the corresponding segment in the large *BltBuffer* standing for the whole screen buffer.

Number	GUID	Assertion	Test Description
5.6.6.3.44	0x03093b96, 0x2b15, 0x4008, 0xb7, 0xbf, 0x9f, 0x8c, 0x17, 0x41, 0x2d, 0xb3	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt – EfiBltVideoToBltBuffer** operation should retrieve the pixels from the video display rectangle to the specified buffer.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display BMP file to some position of the screen. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to retrieve the whole screen to a large *BltBuffer*. The return status should be **EFI_SUCCESS** •Change *SourceX* from 0 to HorizontalResolution step by step, and change *SourceY* from 0 to VerticalResolution step by step, carry out following action: For small *Width*, *Height*,and *BltBuffer*, Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer**. The returned status should be **EFI_SUCCESS**. The small *BltBuffer* retrieved should be the same as the corresponding segment in the large *BltBuffer* standing for the whole screen buffer.

Number	GUID	Assertion	Test Description
5.6.6.3.45	0x1ef36d93, 0x8591, 0x4172, 0x94, 0xfd, 0x93, 0x08, 0x54, 0x6e, 0x73, 0x11	**EFI_GRAPHICS_OUTPUT_PROTOCOL.Blt** –Blt/ **EfiBltVideoToBltBuffe**, Pixel verification.	For valid graphics mode number from 0 to *MaxMode*-1: •Prepare some *BltBuffer* from a small BMP file. •Call **Blt()** with a *BltOperation* value of **EfiBltBufferToVideo** to display BMP file to some position of the screen. •Call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer** to retrieve the whole screen to a large *BltBuffer*. The return status should be **EFI_SUCCESS** •Change *SourceX* from 0 to HorizontalResolution step by step, and change *SourceY* from 0 to VerticalResolution step by step, carry out following action: For mall *Width*, *Height*, and *BltBuffer*, call **Blt()** with a *BltOperation* value of **EfiBltVideoToBltBuffer**. The returned status should be **EFI_SUCCESS**. The small *BltBuffer* retrieved should be the same as the corresponding segment in the large *BltBuffer* standing for the whole screen buffer.

8.6 EFI_SIMPLE_ TEXT_INPUT_EX_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL Section.

8.6.1 Reset()

Number	GUID	Assertion	Test Description
5.6.7.1.1	0xc969bba7, 0xed63, 0x4235, 0x80, 0x46, 0xa1, 0x8c, 0xa2, 0x8a, 0x3f, 0x6a	**SIMPLE_TEXT_INPUT_EX_PROTOCOL.Reset - Reset()** returns **EFI_SUCCESS** and **ReadKeyStrokeEx** return **EFI_NOT_READY**	It is a auto test. Call **Reset()**. **EFI_SUCCESS** should be returned. Call **ReadKeyStrokeEx()**. **EFI_NOT_READY** should be returned.

5.6.7.1.2	0x35381b6c, 0x1035, 0x4241, 0x95, 0x80, 0x21, 0x25, 0x3b, 0x78, 0x60, 0x8d	`SIMPLE_TEXT_INPUT_EX_PROTOCOL.Reset - Reset()` returns `EFI_SUCCESS` and `ReadKeyStrokeEx` return `EFI_NOT_READY`	It is a manual test. Press a key. Call `Reset()`. `EFI_SUCCESS` should be returned. Call `ReadKeyStrokeEx()`. `EFI_NOT_READY` should be returned.

8.6.2 ReadKeyStrokeEx ()

Number	GUID	Assertion	Test Description
5.6.7.2.1	0x89854ccd, 0xa672, 0x4856, 0xb7, 0x6c, 0xb1, 0x66, 0xc5, 0x64, 0x2f, 0x9a	`SIMPLE_TEXT_INPUT_EX_PROTOCOL.ReadKeyStorkeEx - ReadKeyStorkeEx()` returns `EFI_INVALID_PARAMETER` with KeyData being `NULL`.	Call `Reset()` first, and `EFI_SUCCESS` should be returned. Call `ReadKeyStrokeEx()` with `KeyData` being `NULL`. `EFI_INVALID_PARAMETER` should be returned.
5.6.7.2.2	0x5d141dc0, 0xded6, 0x4e01, 0xa9, 0x8b, 0x55, 0x1f, 0x3e, 0xe3, 0x59, 0x4d	`SIMPLE_TEXT_INPUT_EX_PROTOCOL.ReadKeyStorkeEx - ReadKeyStorkeEx()` returns `EFI_NOT_READY` with console just been reseted.	Call `Reset()` first, and `EFI_SUCCESS` should be returned. Call `ReadKeyStrokeEx()` with valid parameter. `EFI_NOT_READY` should be returned.
5.6.7.2.3	0x5eed7df1, 0x4630, 0x44e1, 0x97, 0xaa, 0xd3, 0x26, 0x82, 0x24, 0xc4, 0x30	`EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL.ReadKeyStroke - ReadKeyStrokeEx()` return `EFI_SUCCESS` with key input	It is a manual test. Part 1: Call `Reset()`. `EFI_SUCCESS` should be returned. Press a key. Call `ReadKeyStrokeEx()` with valid parameter. `EFI_SUCCESS` should be returned.
5.6.7.2.4	0x3032721e, 0x8089, 0x49d4, 0x94, 0x5a, 0x46, 0x07, 0xdc, 0x05, 0xcf, 0x8d	`EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL.ReadKeyStroke - ReadKeyStroke()` with key input, user's view	Part 2: Echo the key which is pressed. Tester decides the SUCCESS or Failure.

8.6.3 SetState ()

Number	GUID	Assertion	Test Description
5.6.7.3.1	0x6647a0e7, 0x483c, 0x4777, 0xa9, 0x4b, 0xc8, 0xbc, 0xa3, 0xdf, 0xc7, 0x9c	`SIMPLE_TEXT_INPUT_EX_PROTOCOL.SetState - SetState()` returns `EFI_INVALID_PARAMETER` with KeyToggleState being `NULL`.	Call `SetState()` with *`KeyToggleState`* being `NULL`. Return status should be `EFI_INVALID_PARAMETER`.
5.6.7.3.2	0x4c766c77, 0xdbf3, 0x4b3d, 0x82, 0x59, 0x81, 0xf8, 0xb8, 0xaa, 0x17, 0x75	`SIMPLE_TEXT_INPUT_EX_PROTOCOL.SetState - SetState()` returns `EFI_UNSUPPORTED` with KeyToggleState being a unsupported bit set.	Call `SetState()` with unsupported *`KeyToggleState`*. Return status should be `EFI_UNSUPPORTED`.
5.6.7.3.3	0x44bf142c, 0x72a9, 0x445e, 0xaf, 0x84, 0xaa, 0xc5, 0x96, 0xc6, 0x3f, 0xc8	`SIMPLE_TEXT_INPUT_EX_PROTOCOL.SetState - SetState()` returns `EFI_UNSUPPORTED` or `EFI_SUCCESS` with a valid bit set	Call `SetState()` with valid `KeyToogleState.` The return status should be `EFI_UNSUPPORTED` or `EFI_SUCCESS`. Press a key and call `ReadKeyStrokeEx()`. The *`KeyToggleState`* should be same as the State which be set.

8.6.4 RegisterKeyNotify ()

Number	GUID	Assertion	Test Description
5.6.7.4.1	0x27a40c7e, 0x119e, 0x451d, 0x84, 0x70, 0x1d, 0xc4, 0x52, 0x09, 0x2b, 0x0a	`SIMPLE_TEXT_INPUT_EX_PROTOCOL.RegisterKeyNotify - RegisterKeyNotify()` returns `EFI_INVALID_PARAMETER` with *`KeyData`* being `NULL`	Call `RefisterKeyNotify()` with `NULL` *`KeyData`*. The return status should be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.6.7.4.2	0xb03a561d, 0x6339, 0x4035, 0xaf, 0xd5, 0xfa, 0x2e, 0xce, 0x16, 0x4b, 0xf9	**SIMPLE_TEXT_INPUT_EX_PROTOCOL.RegisterKeyNotify - RegisterKeyNotify()** returns **EFI_INVALID_PARAMETER** with *KeyNotificationFunction* being **NULL**	Call **RefisterKeyNotify()** with **NULL** *KeyNotificationFunction*. The return status should be **EFI_INVALID_PARAMETER**.
5.6.7.4.3	0x5b22932e, 0xc24d, 0x45fe, 0x8b, 0xbd, 0x2e, 0x0e, 0x56, 0xfa, 0xc3, 0x16	**SIMPLE_TEXT_INPUT_EX_PROTOCOL.RegisterKeyNotify - RegisterKeyNotify()** returns **EFI_INVALID_PARAMETER** with *NotifyHandle* being **NULL**.	Call **RefisterKeyNotify()** with **NULL** *NotifyHandle*. The return status should be **EFI_INVALID_PARAMETER**.
5.6.7.4.4	0x69a1c06c, 0x516e, 0x4595, 0xbe, 0x4f, 0x6b, 0x18, 0x58, 0xcc, 0x82, 0x3d	**SIMPLE_TEXT_INPUT_EX_PROTOCOL.RegisterKeyNotify - RegisterKeyNotify()** returns **EFI_SUCCESS**.	It is a manual test. Part 1: Call **Reset()**. The return status should be **EFI_SUCCESS**. Press a key, call **ReadKeyStrokeEx()** to get the key value. The return status should be **EFI_SUCCESS**. Call **RegisterKeyNotify()** with the key. The return status should **EFI_SUCCESS**.
5.6.7.4.5	0x6f509a8c, 0x0df2, 0x499d, 0x97, 0x56, 0x35, 0xbe, 0x3c, 0xcb, 0x21, 0xc4	**SIMPLE_TEXT_INPUT_EX_PROTOCOL.RegisterKeyNotify - RegisterKeyNotify()** returns **EFI_SUCCESS** and notify function has been invoked.	Part 2: Call **Reset()**. The return status should be **EFI_SUCCESS**. Press the same key and check the the result of the notify function.. Call **UnregisterKeyNotify()**.

8.6.5 UnregisterKeyNotify ()

Number	GUID	Assertion	Test Description
5.6.7.5.1	0xa5244802, 0xf4bf, 0x4e8a, 0xaa, 0x76, 0xe4, 0x87, 0x37, 0x43, 0xd8, 0xd1	**SIMPLE_TEXT_INPUT_EX_PROTOCOL.UnregisterKeyNotify - UnregisterKeyNotify()** returns **EFI_INVALID_PARAMETER** with *NotifyHandle* being **NULL**	Call **UnrefisterKeyNotify** with **NULL** *NotifyHandle*. The return status should be **EFI_INVALID_PARAMETER**.
5.6.7.5.2	0x5d1c8b11, 0x326f, 0x4cf5, 0xb0, 0x3d, 0x89, 0xaa, 0x2f, 0xaf, 0x66, 0x42	**SIMPLE_TEXT_INPUT_EX_PROTOCOL.UnregisterKeyNotify - UnregisterKeyNotify()** returns **EFI_INVALID_PARAMETER** with *NotifyHandle* not refer to a register notify function anymore.	Call **RegisterKeyNotify()** with valid parameter. Return status should be **EFI_SUCCESS**. Call **UnregisterKeyNotify()** twice. Return status should be **EFI_INVALID_PARAMETER**.
5.6.7.5.3	0x5fe62478, 0x4614, 0x4430, 0xb9, 0xe9, 0x30, 0xe2, 0x12, 0x19, 0xeb, 0x35	**SIMPLE_TEXT_INPUT_EX_PROTOCOL.UnregisterKeyNotify - UnregisterKeyNotify()** returns **EFI_INVALID_PARAMETER** with NotifyHandle being illegal format.	Call **UnrefisterKeyNotify()** with invalid *NotifyHandle*. The return status should be **EFI_INVALID_PARAMETER**.
5.6.7.5.4	0xe305a4b5, 0x03c3, 0x43c4, 0x93, 0x16, 0x7d, 0x7a, 0xb3, 0x6a, 0x13, 0xa5	**SIMPLE_TEXT_INPUT_EX_PROTOCOL.UnregisterKeyNotify - UnregisterKeyNotify()** returns **EFI_SUCCESS**.	It is a manual test. Part 1: Call **Reset()**. The return status should be **EFI_SUCCESS**. Press a key, call **ReadKeyStrokeEx()** to get the key value. The return status should be **EFI_SUCCESS**. Call **RegisterKeyNotify()** with the key. The return status should **EFI_SUCCESS**. Call **UnregisterKeyNotify()** with the valid *NotifyHandle*. The return status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.6.7.5.5	0x534369f7, 0x8399, 0x4353, 0x94, 0xad, 0xc4, 0x48, 0xfa, 0xda, 0xeb, 0x84	**SIMPLE_TEXT_INPUT_EX_PROTOCOL.UnregisterKeyNotify - UnregisterKeyNotify()** returns **EFI_SUCCESS** and notify function has not been invoked.	Part 2: Call **Reset()**. The return status should be **EFI_SUCCESS**. Press the same key and check the the result of the notify function.

8.7 EFI_ABSOLUTE_POINTER_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_ABSOLUTE_POINTER_PROTOCOL Section.

8.7.1 Reset()

Number	GUID	Assertion	Test Description
5.6.8.1.1	0xa4e0b129, 0x4bd4, 0x4446, 0x8d, 0x32, 0xaa, 0x45, 0x64, 0xb2, 0x74, 0x6e	**EFI_ABSOLUTE_POINTER_PROTOCOL.Reset - Reset()** with an ExtendedVerification value of **FALSE** returns **EFI_SUCCESS**. If device error, should return **EFI_DEVICE_ERROR**	1. Call **Reset()** with an *ExtendedVerification* value of **FALSE**. The return code should be **EFI_SUCCESS** or **EFI_DEVICE_ERROR**.
5.6.8.1.2	0xc246b3ff, 0xc1d5, 0x499b, 0x92, 0x87, 0x73, 0xf5, 0x88, 0xf6, 0xa9, 0x9f	**EFI_ABSOLUTE_POINTER_PROTOCOL.Reset - Reset()** with an ExtendedVerification value of **TRUE** returns **EFI_SUCCESS**. If device error, should return **EFI_DEVICE_ERROR**.	1. Call **Reset()** with an *ExtendedVerification* value of **TRUE**. The return code should be **EFI_SUCCESS** or **EFI_DEVICE_ERROR**.
5.6.8.1.3	0xab689092, 0xc9e2, 0x4618, 0x90, 0xa8, 0x4, 0x74, 0xde, 0x94, 0x7c, 0x4e	**EFI_ABSOLUTE_POINTER_PROTOCOL.Reset - GetState()** after **Reset ()** return **EFI_UNSUPPORTED**	1. Call **Reset()** with an *ExtendedVerification* value of **FALSE**. 2. Call **GetState()**,the return code maybe **EFI_UNSUPPORTED**.
5.6.8.1.4	0x5c250202, 0xe791, 0x4cee, 0x86, 0x74, 0x4e, 0x3a, 0x43, 0xbc, 0x18, 0x15	**EFI_ABSOLUTE_POINTER_PROTOCOL.Reset - Reset()** with an ExtendedVerification value of **TRUE** returns **EFI_SUCCESS**.	1. Call **Reset()** with an *ExtendedVerification* value of **TRUE**. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.6.8.1.5	0x39c1417d, 0xf6a2, 0x4a77, 0xbb, 0xcd, 0xe, 0xb8, 0xeb, 0x41, 0xe7, 0x52	**EFI_ABSOLUTE_POINTER_PROTOCOL.Reset - GetState()** after **Reset ()** return 0 for all current movement.	1. Call **Reset()** with an *ExtendedVerification* value of **TRUE**. 2. Call **GetState()**. if success, Current movement in X,Y and Z should be 0.
5.6.8.1.6	0xd909148a, 0xd05a, 0x4694, 0xb4, 0xc4, 0xfc, 0x27, 0x87, 0x40, 0xce, 0x78	**EFI_ABSOLUTE_POINTER_PROTOCOL.Reset - GetState()** after **Reset ()** return **EFI_UNSUPPORTED**	1. Call **Reset()** with an *ExtendedVerification* value of **TRUE**. 2. Call **GetState(),**the return code maybe **EFI_UNSUPPORTED**.

8.7.2 GetState()

Number	GUID	Assertion	Test Description
5.6.8.2.1	0x5271062f, 0xdef9, 0x4d30, 0x84,0x3b, 0x8d,0x6e, 0x41,0x33, 0x13,0xf3	**EFI_ABSOLUTE_POINTER_PROTOCOL.GetState - GetState()** after **Reset()** returns 0 for all current movement.	1. Call **Reset()** with an *ExtendedVerification* value of **FALSE**. 2. Call **GetState()**. if success, Current movement in X,Y and Z should be 0.
5.6.8.2.2	0x7614c448, 0x12a0, 0x403d, 0x8a,0xde, 0x98,0x97, 0x51,0x7d, 0xd8,0x49	**EFI_ABSOLUTE_POINTER_PROTOCOL.GetState - GetState()** returns **EFI_NOT_READY** when there is no move since last call of **GetState()**or returns **EFI_DEVICE_ERROR** while a device error occurred..	1. Call **Reset()** with an *ExtendedVerification* value of FALSE. 2. Call **GetState()** . 3. Call **GetState()** again, the return code should be **EFI_NOT_READY** or **EFI_DEVICE_ERROR**.
5.6.8.2.3	0x2f8f8711, 0x02dd, 0x411f, 0xaa,0xb5, 0x27,0xe1, 0x3a,0x6a, 0xb2,0x79	**EFI_ABSOLUTE_POINTER_PROTOCOL.GetState - GetState()** after Reset() returns 0 for all current movement.	1. Call **Reset()** with an *ExtendedVerification* value of **TRUE**. 2. Call **GetState()**. if success, Current movement in X,Y and Z should be 0.

Number	GUID	Assertion	Test Description
5.6.8.2.4	0x3db7ea19, 0xda9d, 0x4760, 0xa7,0x43, 0x04,0xb4, 0x8d,0x14, 0x4e,0x90	**`EFI_ABSOLUTE_POINTER_PROTOCOL.GetState - GetState()`** returns **`EFI_NOT_READY`** when there is no move since last call of **`GetState()`** or returns **`EFI_DEVICE_ERROR`** while a device error occurred.	1. Call **`Reset()`** with an *`ExtendedVerification`* value of **`TRUE`**. 2. Call **`GetState()`** . 3. Call **`GetState()`** again, the return code should be **`EFI_NOT_READY`** or **`EFI_DEVICE_ERROR`**.

9 Protocols Bootable Image Support Test

9.1 EFI_LOAD_FILE_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_LOAD_FILE_PROTOCOL Section.

No automatic test is designed to test this protocol.

9.2 EFI_SIMPLE_FILE_SYSTEM_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_SIMPLE_FILE_SYSTEM_PROTOCOL Section.

9.2.1 OpenVolume()

Number	GUID	Assertion	Test Description
5.7.2.1.1	0xe1bbbe46, 0x1fe6, 0x4f0b, 0x8d, 0x2e, 0x1b, 0x94, 0x5c, 0x16, 0xf4, 0x87	`EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – OpenVolume()` returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `OpenVolume()` to open root directory at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.
5.7.2.1.2	0xdf0cc997, 0x16b5, 0x4f26, 0x9f, 0x95, 0xb5, 0x53, 0x5c, 0x73, 0xe6, 0x86	`EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – OpenVolume()` returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `OpenVolume()` to open root directory at `TPL_CALLBACK`. The return code should be `EFI_SUCCESS`.
5.7.2.1.3	0xe4d6498c, 0xc4d5, 0x4dd2, 0x93, 0x88, 0x3c, 0x7b, 0xd2, 0x94, 0x9b, 0x4c	`EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – GetInfo()` to retrieve `EFI_FILE_INFO` on root directory at `TPL_APPLICATION` returns `EFI_SUCCESS`.	1. Call `OpenVolume()` to open root directory at `TPL_APPLICATION`. 2. Call `GetInfo()` to retrieve `EFI_FILE_INFO` on root directory. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.7.2.1.4	0xeca437ce, 0xcca2, 0x4f7d, 0xb0, 0x55, 0x42, 0x99, 0x78, 0x46, 0xe5, 0x5c	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – GetInfo() to retrieve EFI_FILE_INFO on root directory at TPL_CALLBACK returns EFI_SUCCESS.	1. Call OpenVolume() to open root directory at TPL_CALLBACK. 2. Call GetInfo() to retrieve EFI_FILE_INFO on root directory. The return code should be EFI_SUCCESS.
5.7.2.1.5	0xacf4bb1e, 0x292b, 0x46a5, 0x9d, 0x98, 0xac, 0xa1, 0x04, 0x02, 0x43, 0x1c	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – EFI_FILE_INFO.Attribute & EFI_FILE_DIRECTORY != 0 at TPL_APPLICATION.	1. Call OpenVolume() to open root directory at TPL_APPLICATION. 2. Call GetInfo() to retrieve EFI_FILE_INFO on root directory. 3. EFI_FILE_INFO.Attribute & EFI_FILE_DIRECTORY != 0.
5.7.2.1.6	0x7639775e, 0xb879, 0x4c64, 0x87, 0xde, 0x96, 0x6b, 0xb7, 0x76, 0xb8, 0x6b	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – EFI_FILE_INFO.Attribute & EFI_FILE_DIRECTORY != 0 at TPL_CALLBACK.	1. Call OpenVolume() to open root directory at TPL_CALLBACK. 2. Call GetInfo() to retrieve EFI_FILE_INFO on root directory. The return code should be EFI_SUCCESS. 3. EFI_FILE_INFO.Attribute & EFI_FILE_DIRECTORY != 0.
5.7.2.1.7	0x21746222, 0x29c8, 0x4b78, 0x87, 0x3e, 0x35, 0x4e, 0x58, 0x26, 0x79, 0xde	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO on root directory at TPL_APPLICATION returns EFI_SUCCESS.	1. Call OpenVolume() to open root directory at TPL_APPLICATION. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO on root directory. The return code should be EFI_SUCCESS.
5.7.2.1.8	0x454082d8, 0x05b5, 0x48df, 0xb0, 0x91, 0x99, 0xb7, 0xdd, 0x87, 0x05, 0x10	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO on root directory at TPL_CALLBACK returns EFI_SUCCESS.	1. Call OpenVolume() to open root directory at TPL_CALLBACK. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO on root directory. The return code should be EFI_SUCCESS.
5.7.2.1.9	0x31b71760, 0xbe9c, 0x47aa, 0x8c, 0x49, 0x4c, 0xcf, 0x33, 0x44, 0x57, 0x9f	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL on root directory at TPL_APPLICATION returns EFI_SUCCESS.	1. Call OpenVolume() to open root directory at TPL_APPLICATION. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL on root directory. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.2.1.10	0x7853b6a4, 0x66ba, 0x4d50, 0xa9, 0x06, 0xd7, 0x9a, 0x12, 0xa9, 0x21, 0x8e	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL on root directory at TPL_CALLBACK returns EFI_SUCCESS.	1. Call OpenVolume() to open root directory at TPL_CALLBACK. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL on root directory. The return code should be EFI_SUCCESS.
5.7.2.1.11	0x943883d4, 0xb2c6, 0x4041, 0x98, 0xab, 0x34, 0x2f, 0x9c, 0x24, 0x8c, 0x0c	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – Volume labels gotten from EFI_FILE_SYSTEM_INFO and EFI_FILE_SYSTEM_VOLUME_LABEL should be the same at TPL_APPLICATION.	1. Call OpenVolume() to open root directory at TPL_APPLICATION. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO on root directory. 3. Call GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL on root directory. 4. Volume labels gotten from EFI_FILE_SYSTEM_INFO and EFI_FILE_SYSTEM_VOLUME_LABEL should be the same.
5.7.2.1.12	0x6fdeb4e4, 0xe12d, 0x4c6b, 0x8e, 0x8c, 0xcd, 0x83, 0x34, 0x0b, 0x1f, 0xe6	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – Volume labels gotten from EFI_FILE_SYSTEM_INFO and EFI_FILE_SYSTEM_VOLUME_LABEL should be the same at TPL_CALLBACK.	1. Call OpenVolume() to open root directory at TPL_CALLBACK. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO on root directory. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL on root directory. 4. Volume labels gotten from EFI_FILE_SYSTEM_INFO and EFI_FILE_SYSTEM_VOLUME_LABEL should be the same
5.7.2.1.13	0x2b9fe6a3, 0xd6b0, 0x4ab9, 0x9e, 0x92, 0xbe, 0x93, 0xba, 0x4f, 0xcd, 0xe1	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – Delete() root directory returns EFI_WARN_DELETE_FAILURE at TPL_APPLICATION.	1. Call OpenVolume() to open root directory at TPL_APPLICATION. 2. Call Delete() to delete root directory. The return code should be EFI_WARN_DELETE_FAILURE.
5.7.2.1.14	0xf958f344, 0xa399, 0x437e, 0xa8, 0x85, 0x29, 0xab, 0x58, 0xe6, 0x88, 0x91	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume – Delete() root directory returns EFI_WARN_DELETE_FAILURE at TPL_CALLBACK.	1. Call OpenVolume() to open root directory at TPL_CALLBACK. 2. Call Delete() to delete root directory. The return code should be EFI_WARN_DELETE_FAILURE.

9.3 EFI_FILE_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_FILE_ PROTOCOL Section.

9.3.1 Open()

Number	GUID	Assertion	Test Description
5.7.3.1.1	0x9c974f8c, 0x9e6a, 0x4188, 0x81, 0xc5, 0x7f, 0x1a, 0x12, 0x33, 0x60, 0x94	`EFI_FILE_PROTOCOL.Open - Open()` to create file under root directory with pure filename returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Open()` to create file under root directory with pure filename at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.
5.7.3.1.2	0x02e9e015, 0x3ed6, 0x4c43, 0x91, 0xec, 0x6e, 0x70, 0x05, 0xe1, 0xfd, 0xc0	`EFI_FILE_PROTOCOL.Open - Open()` to create file under root directory with pure filename returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Open()` to create file under root directory with pure filename at `TPL_CALLBACK`. The return code should be `EFI_SUCCESS`.
5.7.3.1.3	0x76e95e01, 0xf92b, 0x4068, 0xab, 0x80, 0x06, 0x25, 0x30, 0x8b, 0x8a, 0x06	`EFI_FILE_PROTOCOL.Open - Open()` to create directory under root directory returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Open()` to create directory under root directory at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.
5.7.3.1.4	0xa5073db1, 0x277c, 0x4714, 0xb2, 0x67, 0x24, 0xd2, 0x2f, 0xbc, 0x4b, 0x96	`EFI_FILE_PROTOCOL.Open - Open()` to create directory under root directory returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Open()` to create directory under root directory at `TPL_CALLBACK`. The return code should be `EFI_SUCCESS`
5.7.3.1.5	0xecc31b62, 0x9297, 0x454d, 0xbd, 0x50, 0x9c, 0x63, 0xd4, 0x8a, 0x01, 0xe9	`EFI_FILE_PROTOCOL.Open - Open()` to create file under root directory with filename containing sub directory name returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Open()` to create file under root directory with filename containing sub directory name at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.7.3.1.6	0x99a1fb48, 0xe279, 0x4b2f, 0x9c, 0x74, 0x42, 0xa1, 0x35, 0x7b, 0x83, 0x93	EFI_FILE_PROTOCOL.Open - Open() to create file under root directory with filename containing sub directory name returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Open() to create file under root directory with filename containing sub directory name at TPL_CALLBACK. The return code should be EFI_SUCCESS.
5.7.3.1.7	0xc2a6e394, 0x4e56, 0x41a0, 0x84, 0xce, 0xf2, 0xd3, 0x30, 0x74, 0xda, 0xa3	EFI_FILE_PROTOCOL.Open - Open() to create directory under root directory returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Open() to create directory under root directory at TPL_APPLICATION. The return code should be EFI_SUCCESS.
5.7.3.1.8	0x6e444a6d , 0x6eb0, 0x42cc, 0x9b, 0xcb, 0x26, 0x79, 0x29, 0xc1, 0x69, 0x25	EFI_FILE_PROTOCOL.Open - Open() to create directory under root directory returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Open() to create directory under root directory at TPL_CALLBACK. The return code should be EFI_SUCCESS.
5.7.3.1.9	0x355911f3, 0x0f0e, 0x4deb, 0x9e, 0x8b, 0x70, 0xa2, 0x04, 0x6b, 0x77, 0x38	EFI_FILE_PROTOCOL.Open - Open() to create file under sub directory with pure filename returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Open() to create file under sub directory with pure filename at TPL_APPLICATION. The return code should be EFI_SUCCESS.
5.7.3.1.10	0xc2c3a263, 0x7b56, 0x4845, 0x8f, 0x50, 0x8c, 0xf4, 0x61, 0xdb, 0x7f, 0x53	EFI_FILE_PROTOCOL.Open - Open() to create file under sub directory with pure filename returns EFI_SUCCESS at TPL_CALLBACK	1. Call Open() to create file under sub directory with pure filename at TPL_CALLBACK. The return code should be EFI_SUCCESS.
5.7.3.1.11	0xaffa623a, 0x30f8, 0x44e3, 0xad, 0x85, 0x36, 0xa3, 0xa3, 0xdb, 0xfd, 0x9f	EFI_FILE_PROTOCOL.Open - Open() to create directory under root directory returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Open() to create directory under root directory at TPL_APPLICATION. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.1.12	0xed784eaf, 0x75db, 0x4bde, 0x8d, 0x5e, 0xeb, 0x5d, 0x22, 0x9a, 0x59, 0x39	EFI_FILE_PROTOCOL.Open - Open() to create directory under root directory returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Open() to create directory under root directory at TPL_CALLBACK. The return code should be EFI_SUCCESS.
5.7.3.1.13	0xf9517e49, 0x4aea, 0x4b7b, 0xa3, 0x92, 0xd9, 0x37, 0x55, 0x3e, 0x3a, 0x6c	EFI_FILE_PROTOCOL.Open - Open() to create directory under sub directory returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Open() to create directory under sub directory at TPL_APPLICATION. The return code should be EFI_SUCCESS.
5.7.3.1.14	0x9294bf3e, 0x589f, 0x498b, 0x97, 0xca, 0xf3, 0xb4, 0xda, 0x1a, 0xb8, 0x4c	EFI_FILE_PROTOCOL.Open - Open() to create directory under sub directory returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Open() to create directory under sub directory at TPL_CALLBACK. The return code should be EFI_SUCCESS.
5.7.3.1.15	0xe01a3218, 0x4f72, 0x4c8f, 0x9d, 0x13, 0x41, 0xbf, 0x02, 0xc4, 0x39, 0xb3	EFI_FILE_PROTOCOL.Open - Open() to create file with sub directory handle and filename containing sub directory name returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Open() to create directory(dir1) under root directory at TPL_APPLICATION. 2. Call Open() to create directory(dir2) under sub directory(dir1) . 3. Call Open() to create file with sub directory handle(dir1) and filename containing sub directory name(dir2\filename). The return code should be EFI_SUCCESS.
5.7.3.1.16	0x6b32a44f, 0x5670, 0x4ce6, 0xbb, 0xb5, 0x36, 0xd8, 0x29, 0x0e, 0x2c, 0x50	EFI_FILE_PROTOCOL.Open - Open() to create file with sub directory handle and filename containing sub directory name returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Open() to create directory(dir1) under root directory at TPL_CALLBACK. 2. Call Open() to create directory(dir2) under sub directory(dir1) . 3. Call Open() to create file with sub directory handle(dir1) and filename containing sub directory name(dir2\filename). The return code should be EFI_SUCCESS.
5.7.3.1.17	0x36c16a36, 0x0891, 0x4108, 0x84, 0x86, 0x9f, 0x24, 0x3d, 0xde, 0x25, 0xe2	EFI_FILE_PROTOCOL.Open - Open() to create directory under root directory returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Open() to create directory under root directory at TPL_APPLICATION. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.1.18	0xf9f0d04b, 0x1409, 0x4157, 0x9b, 0x51,0x80, 0xf0, 0xb5, 0xea, 0xa4, 0x92	`EFI_FILE_PROTOCOL.Open` - `Open()` to create directory under root directory returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Open()` to create directory under root directory at `TPL_CALLBACK`. The return code should be `EFI_SUCCESS`.
5.7.3.1.19	0xda80d9df, 0xa96b, 0x44d1, 0xac, 0xba, 0x0b, 0x92, 0x1d, 0xe1, 0xf2, 0x72	`EFI_FILE_PROTOCOL.Open` - `Open()` to create directory under sub directory returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Open()` to create directory under sub directory at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.
5.7.3.1.20	0xdf2be803, 0x6ae8, 0x477f, 0x99, 0xf3, 0xd4, 0x90, 0x80, 0x90, 0x15, 0x2c	`EFI_FILE_PROTOCOL.Open` - `Open()` to create directory under sub directory returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Open()` to create directory under sub directory at `TPL_CALLBACK`. The return code should be `EFI_SUCCESS`
5.7.3.1.21	0xc48ebac5, 0xc94a, 0x434d, 0x8a, 0x35, 0xb6, 0x40, 0x61, 0x98, 0xf9, 0xa3	`EFI_FILE_PROTOCOL.Open` - `Open()` to create file with root handle and filename containing absolute file path returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Open()` to create directory(dir1) under root directory at `TPL_APPLICATION`. 2. Call `Open()` to create directory(dir2) under sub directory(dir1) . 3. Call `Open()` to create file with root handle and filename containing absolute file path (dir1\dir2\filename). The return code should be `EFI_SUCCESS`.
5.7.3.1.22	0x9ae5e6ce, 0x1e6e, 0x42b7, 0x93, 0x6b, 0x66, 0xd6, 0x94, 0x2f, 0x9b, 0x98	`EFI_FILE_PROTOCOL.Open` - `Open()` to create file with root handle and filename containing absolute file path returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Open()` to create directory(dir1) under root directory at `TPL_CALLBACK`. 2. Call `Open()` to create directory(dir2) under sub directory(dir1) . 3. Call `Open()` to create file with root handle and filename containing absolute file path (dir1\dir2\filename). The return code should be `EFI_SUCCESS`.
5.7.3.1.23	0x533f1869, 0xebc8, 0x444c, 0x8c, 0xf6, 0x44, 0x09, 0x80, 0x31, 0xce, 0xa4	`EFI_FILE_PROTOCOL.Open` - `Open()` to create file under root directory with pure filename returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Open()` to create file under root directory with pure filename at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.7.3.1.24	0xe2ba78af, 0xa282, 0x45f4, 0xaa, 0x92, 0xe6, 0xa2, 0xc7, 0xad, 0x8c, 0x70	`EFI_FILE_PROTOCOL.Open` - `Open()` to create file under root directory with pure filename returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Open()` to create file under root directory with pure filename at `TPL_CALLBACK`. The return code should be `EFI_SUCCESS`.
5.7.3.1.25	0x9eb5fd5d, 0x3d06, 0x4e49, 0x98, 0xb4, 0x41, 0x77, 0x61, 0xca, 0x3a, 0x75	`EFI_FILE_PROTOCOL.Open` - `Open()` to create an existing file opens the existing file at `TPL_APPLICATION`.	1. Call `Open()` to create an existing file under root directory with pure filename at `TPL_APPLICATION`. The existing file should be opened and the return code should be `EFI_SUCCESS`.
5.7.3.1.26	0x6ed38ac8, 0x0f4a, 0x4294, 0x9d, 0x9c, 0xb6, 0x6f, 0xa6, 0x00, 0xcf, 0xd1	`EFI_FILE_PROTOCOL.Open` - `Open()` to create an existing file opens the existing file at `TPL_CALLBACK`.	1. Call `Open()` to create an existing file under root directory with pure filename at `TPL_CALLBACK`. The existing file should be opened and the return code should be `EFI_SUCCESS`.
5.7.3.1.27	0x1a6ec46e, 0x5a2b, 0x43e0, 0x98, 0x59, 0x36, 0x6a, 0xbb, 0xdf, 0xf4, 0x86	`EFI_FILE_PROTOCOL.Open` – `Write()` and `SetInfo()` to existing file returns `EFI_SUCCESS` except read-only mode at `TPL_APPLICATION`.	1. Call `Open()` to create a file. 2. Call `Write()` and `SetInfo()` to the new file. The return code should be `EFI_SUCCESS` and the file size should be equal to the set value.
5.7.3.1.28	0x5e81beb3, 0x3cee, 0x4724, 0xa9, 0xa4, 0x6d, 0x64, 0xd4, 0x80, 0x87, 0x5d	`EFI_FILE_PROTOCOL.Open` – `Write()` and `SetInfo()` to existing file returns `EFI_SUCCESS` except read-only mode at `TPL_CALLBACK`	1. Call `Open()` to create a file. 2. Call `Write()` and `SetInfo()` to the new file. The return code should be `EFI_SUCCESS` and the file size should be equal to the set value.
5.7.3.1.29	0x249b05c8, 0x931f, 0x4d21, 0xb3, 0x09, 0xbc, 0x0f, 0x32, 0x8f, 0x17, 0xc6	`EFI_FILE_PROTOCOL.Open` - `Open()` to create file under root directory with filename containing sub directory name returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Open()` to create file under root directory with filename containing sub directory name at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.7.3.1.30	0x2c6a8296, 0x3fd8, 0x4e72, 0x80, 0x84, 0x8f, 0x3e, 0x61, 0x59, 0x10, 0xe2	EFI_FILE_PROTOCOL.Open - Open() to create file under root directory with filename containing sub directory name returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Open() to create file under root directory with filename containing sub directory name at TPL_CALLBACK. The return code should be EFI_SUCCESS
5.7.3.1.31	0xf2ec0ec0, 0xc79b, 0x4035, 0xbf, 0x70, 0x3a, 0x0f, 0x02, 0xf7, 0xe9, 0x4f	EFI_FILE_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL_APPLICATION.	1. Call Open() to create an existing file under root directory with pure filename at TPL_APPLICATION. The existing file should be opened and the return code should be EFI_SUCCESS.
5.7.3.1.32	0x18b07457, 0x2108, 0x4c00, 0xb4, 0xb6, 0x88, 0xcb, 0xb1, 0x0e, 0x79, 0xfe	EFI_FILE_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL_CALLBACK	1. Call Open() to create an existing file under root directory with pure filename at TPL_CALLBACK. The existing file should be opened and the return code should be EFI_SUCCESS.
5.7.3.1.33	0x4ee79e47, 0x1530, 0x42ee, 0xbb, 0x70, 0x1a, 0xf3, 0xad, 0x6a, 0xef, 0xda	EFI_FILE_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI_SUCCESS except read-only mode at TPL_APPLICATION.	1. Call Open() to create a file. 2. Call Write() and SetInfo() to the new file. The return code should be EFI_SUCCESS and the file size should be equal to the set value.
5.7.3.1.34	0x0e5ad766, 0x5368, 0x4465, 0xa5, 0x15, 0x1d, 0x5e, 0x65, 0xc9, 0x28, 0x9b	EFI_FILE_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI_SUCCESS except read-only mode at TPL_CALLBACK.	1. Call Open() to create a file. 2. Call Write() and SetInfo() to the new file. The return code should be EFI_SUCCESS and the file size should be equal to the set value.
5.7.3.1.35	0x5f2d183d, 0x748e, 0x4b7b, 0x82, 0xd8, 0xa9, 0xf3, 0x27, 0xa1, 0x96, 0xe3	EFI_FILE_PROTOCOL.Open - Open() to create file under directory handle with pure filename returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Open() to create file under directory handle with pure filename at TPL_APPLICATION. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.1.36	0x953e3193 , 0x444c, 0x47d0, 0x8f, 0x79, 0xb8, 0xa3, 0x02, 0xd2, 0x31, 0x85	`EFI_FILE_PROTOCOL.Open` - `Open()` to create file under directory handle with pure filename returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Open()` to create file under directory handle with pure filename at `TPL_CALLBACK`. The return code should be `EFI_SUCCESS`.
5.7.3.1.37	0xbf004c75, 0x42b1, 0x4038, 0xac, 0x38, 0x1c, 0xfb, 0x6d, 0x24, 0xa9, 0xfa	`EFI_FILE_PROTOCOL.Open` - `Open()` to create an existing file opens the existing file at `TPL_APPLICATION`.	1. Call `Open()` to create an existing file under directory handle with pure filename at `TPL_APPLICATION`. The existing file should be opened and the return code should be `EFI_SUCCESS`.
5.7.3.1.38	0xe5bae1ec, 0x1ce7, 0x4fde, 0xad, 0x6d, 0xfd, 0x0c, 0x85, 0xe8, 0x51, 0x44	`EFI_FILE_PROTOCOL.Open` - `Open()` to create an existing file opens the existing file at `TPL_CALLBACK`.	1. Call `Open()` to create an existing file under directory handle with pure filename at `TPL_CALLBACK`. The existing file should be opened and the return code should be `EFI_SUCCESS`.
5.7.3.1.39	0x16fb933e, 0x1f91, 0x46e6, 0x9e, 0xff, 0xfc, 0x4f, 0x6c, 0x91, 0xde, 0xc4	`EFI_FILE_PROTOCOL.Open` – `Write()` and `SetInfo()` to existing file returns `EFI_SUCCESS` except read-only mode at `TPL_APPLICATION`.	1. Call `Open()` to create a file. 2. Call `Write()` and `SetInfo()` to the new file. The return code should be `EFI_SUCCESS` and the file size should be equal to the set value.
5.7.3.1.40	0x1225566e , 0xb893, 0x4059, 0xa2, 0x55, 0xfc, 0x4c, 0x0c, 0xb5, 0x72, 0x9c	`EFI_FILE_PROTOCOL.Open` – `Write()` and `SetInfo()` to existing file returns `EFI_SUCCESS` except read-only mode at `TPL_CALLBACK`.	1. Call `Open()` to create a file. 2. Call `Write()` and `SetInfo()` to the new file. The return code should be `EFI_SUCCESS` and the file size should be equal to the set value.
5.7.3.1.41	0x62066bfd, 0x6a13, 0x43db, 0x99, 0x85, 0x3f, 0xeb, 0x92, 0xd3, 0x00, 0x7d	`EFI_FILE_PROTOCOL.Open` - `Open()` to create file with directory handle and filename containing sub directory name returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Open()` to create file with directory handle and filename containing sub directory name at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.7.3.1.42	0x2c61c286, 0xd23a, 0x414e, 0x9a, 0x1a, 0x5a, 0xe0, 0xf6, 0xb5, 0x40, 0x33	EFI_FILE_PROTOCOL. Open - Open() to create file with directory handle and filename containing sub directory name returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Open() to create file with directory handle and filename containing sub directory name at TPL_CALLBACK. The return code should be EFI_SUCCESS.
5.7.3.1.43	0x0f0c6895, 0x5d36, 0x4ec9, 0xa4, 0xd8, 0x10, 0x32, 0x94, 0xf2, 0x8a, 0x91	EFI_FILE_PROTOCOL. Open - Open() to create an existing file opens the existing file at TPL_APPLICATION	1. Call Open() to create an existing file with directory handle and filename containing sub directory name at TPL_APPLICATION. The existing file should be opened and the return code should be EFI_SUCCESS.
5.7.3.1.44	0x961d3514 , 0x0fa1, 0x46d0, 0x83, 0x63, 0x0e, 0xa9, 0x96, 0x45, 0xe4, 0xec	EFI_FILE_PROTOCOL. Open - Open() to create an existing file opens the existing file at TPL_CALLBACK.	1. Call Open() to create an existing file with directory handle and filename containing sub directory name at TPL_CALLBACK. The existing file should be opened and the return code should be EFI_SUCCESS.
5.7.3.1.45	0x84c25d8c, 0xc15c, 0x4b18, 0x8c, 0xf8, 0x75, 0x82, 0x73, 0xb1, 0x3f, 0x3e	EFI_FILE_PROTOCOL. Open – Write() and SetInfo() to existing file returns EFI_SUCCESS except read-only mode at TPL_APPLICATION.	1. Call Open() to create a file. 2. Call Write() and SetInfo() to the new file. The return code should be EFI_SUCCESS and the file size should be equal to the set value.
5.7.3.1.46	0x59f0b532, 0x9f66, 0x44f7, 0xa4, 0x7b, 0x0e, 0xb3, 0x65, 0xe8, 0x47, 0x06	EFI_FILE_PROTOCOL. Open – Write() and SetInfo() to existing file returns EFI_SUCCESS except read-only mode at TPL_CALLBACK.	1. Call Open() to create a file. 2. Call Write() and SetInfo() to the new file. The return code should be EFI_SUCCESS and the file size should be equal to the set value.
5.7.3.1.47	0xcf2f3c12, 0x0608, 0x4661, 0xa3, 0x86, 0xf7, 0xc3, 0x13, 0x93, 0xef, 0x7a	EFI_FILE_PROTOCOL. Open - Open() to create file with directory handle and filename containing absolute directory name returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Open() to create file with directory handle and filename containing absolute directory name at TPL_APPLICATION. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.1.48	0x4b3cdcfd, 0xf479, 0x43c2, 0xbb, 0x48, 0xd6, 0xa4, 0x06, 0x06, 0xc5, 0xc2	`EFI_FILE_PROTOCOL.Open – Open()` to create file with directory handle and filename containing absolute directory name returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Open()` to create file with directory handle and filename containing absolute directory name at `TPL_CALLBACK`. The return code should be `EFI_SUCCESS`.
5.7.3.1.49	0x7c4c7717, 0x89cf, 0x46b0, 0x84, 0x89, 0xd6, 0x18, 0x54, 0xa3, 0xc3, 0x2b	`EFI_FILE_PROTOCOL.Open – Open()` to create an existing file opens the existing file at `TPL_APPLICATION`.	1. Call `Open()` to create an existing file with directory handle and filename containing absolute directory name at `TPL_APPLICATION`. The existing file should be opened and the return code should be `EFI_SUCCESS`.
5.7.3.1.50	0xc4849d07, 0x41e8, 0x4636, 0xa8, 0x3e, 0xb1, 0x7c, 0xc0, 0xe9, 0x4f, 0x26	`EFI_FILE_PROTOCOL.Open – Open()` to create an existing file opens the existing file at `TPL_CALLBACK`	1. Call `Open()` to create an existing file with directory handle and filename containing absolute directory name at `TPL_CALLBACK`. The existing file should be opened and the return code should be `EFI_SUCCESS`.
5.7.3.1.51	0xddd23c97, 0xecc8, 0x434d, 0xb0, 0x69, 0x22, 0xcb, 0x26, 0xb5, 0x88, 0xfe	`EFI_FILE_PROTOCOL.Open – Write()` and `SetInfo()` to existing file returns `EFI_SUCCESS` except read-only mode at `TPL_APPLICATION`	1. Call `Open()` to create a file. 2. Call `Write()` and `SetInfo()` to the new file. The return code should be `EFI_SUCCESS` and the file size should be equal to the set value.
5.7.3.1.52	0x0e86769e, 0xe067, 0x4593, 0x82, 0x4c, 0xc9, 0x85, 0x97, 0x51, 0xac, 0x61	`EFI_FILE_PROTOCOL.Open – Write()` and `SetInfo()` to existing file returns `EFI_SUCCESS` except read-only mode at `TPL_CALLBACK`	1. Call `Open()` to create a file. 2. Call `Write()` and `SetInfo()` to the new file. The return code should be `EFI_SUCCESS` and the file size should be equal to the set value.
5.7.3.1.53	0x77240620, 0xcee3, 0x481d, 0xa6, 0xb4, 0x8d, 0x68, 0x50, 0x83, 0x91, 0xd1	`EFI_FILE_PROTOCOL.Open – Open()` with non-existent file name returns `EFI_NOT_FOUND`.	1. Call `Open()` to open a non-existent file. The return code should be `EFI_NOT_FOUND`.

Number	GUID	Assertion	Test Description
5.7.3.1.54	0xa08a7f58, 0xbedb, 0x467b, 0x99, 0x43, 0xe7, 0xe9, 0x6e, 0x62, 0x0d, 0x11	EFI_FILE_PROTOCOL.Open – Open() with non-existent file path returns EFI_NOT_FOUND.	1. Call Open() to create a file handle with non-existent file path. The return code should be EFI_NOT_FOUND.
5.7.3.1.55	0xe2310546, 0xf1ac, 0x47ce, 0xa5, 0x65, 0x88, 0xd0, 0x03, 0x2a, 0x50, 0xa7	EFI_FILE_PROTOCOL.Open – Open() with invalid open-mode returns EFI_INVALID_PARAMETER.	1. Call Open() to open file handle with invalid open-mode. The return code should be EFI_INVALID_PARAMETER.

9.3.2 Close()

Number	GUID	Assertion	Test Description
5.7.3.2.1	0x8f787cb1, 0xe4d7, 0x4d58, 0x97, 0x5b, 0xb6, 0xf1, 0x42, 0xc8, 0xcb, 0xc8	EFI_FILE_PROTOCOL.Close – Close() file handle or directory handle returns EFI_SUCCESS at TPL_APPLICATION	1. Call Open() to create file or directory handles at TPL_APPLICATION. 2. Call Close() to close file handle and directory handles. The return code should be EFI_SUCCESS.
5.7.3.2.2	0x301114f7, 0x1f9d, 0x4dcb, 0xb2, 0xc7, 0x24, 0x17, 0x24, 0x66, 0xc6, 0xd9	EFI_FILE_PROTOCOL.Close – Close() file handle or directory handle returns EFI_SUCCESS at TPL_CALLBACK	1. Call Open() to create file or directory handles at TPL_CALLBACK. 2. Call Close() to close file handle and directory handles. The return code should be EFI_SUCCESS.
5.7.3.2.3	0x134343f0, 0xee4d, 0x4c3d, 0xa5, 0x5d, 0xa2, 0x3c, 0x48, 0x75, 0x51, 0x0c	EFI_FILE_PROTOCOL.Close – Closing a directory does not affect access to files under that directory if the files were opened before the directory was closed at TPL_APPLICATION.	1. Call Open() to create directory and file handles under the directory at TPL_APPLICATION. 2. Call Close() to close the directory handle. 3. Read/Write/GetInfo/SetInfo/GetPosition/SetPosition to the file handles under the closed directory should be a success.

Number	GUID	Assertion	Test Description
5.7.3.2.4	0x46f37004, 0x407a, 0x481f, 0x9a, 0xe6, 0x9f, 0x74, 0x40, 0x93, 0xd7, 0xe8	`EFI_FILE_PROTOCOL.Close` – Closing a directory does not affect access to files under that directory if the files were opened before the directory was closed at `TPL_CALLBACK`.	1. Call `Open()` to create directory and file handles under the directory at `TPL_CALLBACK`. 2. Call `Close()` to close the directory handle. 3. `Read/Write/GetInfo/SetInfo/GetPosition/SetPosition` to the file handles under the closed directory should be a success.
5.7.3.2.5	0xc5da488d, 0x0bbb, 0x49f2, 0xb5, 0xc5, 0xb0, 0x3a, 0xbb, 0x40, 0xe0, 0x42	`EFI_FILE_PROTOCOL.Close – Re-Open` closed file or directory handle returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Open()` to create file or directory handles at `TPL_APPLICATION`. 2. Call `Close()` to close file handle and directory handles. 3. Call `Open()` to re-open the closed handles. The return code should be `EFI_SUCCESS`.
5.7.3.2.6	0xb9478756, 0x46c4, 0x4eaa, 0xa0, 0x35, 0x8b, 0xd2, 0x28, 0xbb, 0xd7, 0x9c	`EFI_FILE_PROTOCOL.Close – Re-Open` closed file or directory handle returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Open()` to create file or directory handles at `TPL_CALLBACK`. 2. Call `Close()` to close file handle and directory handles. 3. Call `Open()` to re-open the closed handles. The return code should be `EFI_SUCCESS`.

9.3.3 Delete()

Number	GUID	Assertion	Test Description
5.7.3.3.1	0xaf9e9d9c, 0x1814, 0x4623, 0x87, 0xac, 0xe5, 0xa3, 0xff, 0x79, 0xfa, 0xf2	`EFI_FILE_PROTOCOL.Delete – Delete()` file handle returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Delete()` to delete file handles at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.
5.7.3.3.2	0x7db63d3b, 0x7819, 0x4f45, 0xa1, 0xfd, 0x75, 0xeb, 0x18, 0xcc, 0xfc, 0x33	`EFI_FILE_PROTOCOL.Delete – Delete()` file handle returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Delete()` to delete file handles at `TPL_CALLBACK`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.7.3.3.3	0xb250cf45, 0x9dd8, 0x41f7, 0x8e, 0x27, 0x96, 0x5e, 0x89, 0xc2, 0xd6, 0x32	`EFI_FILE_PROTOCOL.Delete` – Re-open deleted file handle returns `EFI_NOT_FOUND` at `TPL_APPLICATION`.	1. Call `Delete()` to delete file handles at `TPL_APPLICATION`. 2. Call `Open()` to re-open the deleted file handle. The return code should be `EFI_NOT_FOUND`.
5.7.3.3.4	0xf4dc2e77, 0xd9c7, 0x40d0, 0x83, 0xbd, 0x8f, 0x1e, 0xc6, 0x64, 0x86, 0x69	`EFI_FILE_PROTOCOL.Delete` – Re-open deleted file handle returns `EFI_NOT_FOUND` at `TPL_CALLBACK`.	1. Call `Delete()` to delete file handles at `TPL_CALLBACK`. 2. Call `Open()` to re-open the deleted file handle. The return code should be `EFI_NOT_FOUND`.
5.7.3.3.5	0xb656663f, 0x5c23, 0x4e47, 0xa1, 0x77, 0xc3, 0x34, 0x14, 0x0c, 0x11, 0x07	`EFI_FILE_PROTOCOL.Delete – Delete()` directory handle returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Delete()` to delete directory handles at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.
5.7.3.3.6	0x0f51d637, 0xa67a, 0x4c97, 0x81, 0xcf, 0xbb, 0x8c, 0x4d, 0xf2, 0xdb, 0xdf	`EFI_FILE_PROTOCOL.Delete – Delete()` directory handle returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Delete()` to delete directory handles at `TPL_CALLBACK`. The return code should be `EFI_SUCCESS`.
5.7.3.3.7	0xe0678dae, 0x5aa6, 0x426c, 0xa4, 0xcb, 0x58, 0xa2, 0x7a, 0x9a, 0xa7, 0xb2	`EFI_FILE_PROTOCOL.Delete` – Re-open deleted directory handle returns `EFI_NOT_FOUND` at `TPL_APPLICATION`.	1. Call `Delete()` to delete directory handles at `TPL_APPLICATION`. 2. Call `Open()` to re-open the deleted directory handle. The return code should be `EFI_NOT_FOUND`.
5.7.3.3.8	0xb9c79e4e, 0x187f, 0x46c6, 0x8d, 0x0a, 0x71, 0x70, 0xca, 0x99, 0x31, 0xa7	`EFI_FILE_PROTOCOL.Delete` – Re-open deleted directory handle returns `EFI_NOT_FOUND` at `TPL_CALLBACK`.	1. Call `Delete()` to delete directory handles at `TPL_CALLBACK`. 2. Call `Open()` to re-open the deleted directory handle. The return code should be `EFI_NOT_FOUND`.

Number	GUID	Assertion	Test Description
5.7.3.3.9	0x12f4e25b, 0x77c7, 0x4b47, 0x98, 0xb3, 0xd4, 0xf1, 0x54, 0x24, 0x68, 0x98	EFI_FILE_PROTOCOL. Delete – Delete() nonempty directory returns EFI_WARN_DELETE_FAILURE at TPL_APPLICATION.	1. Call Open() to create directory and file handles under the directory at TPL_APPLICATION. 2. Call Delete() to delete the directory handle. The return code should be EFI_WARN_DELETE_FAILURE
5.7.3.3.10	0x11860155, 0x016e, 0x4c07, 0x83, 0x7a, 0xf1, 0x27, 0x59, 0x2b, 0xf0, 0x75	EFI_FILE_PROTOCOL. Delete – Delete() nonempty directory returns EFI_WARN_DELETE_FAILURE at TPL_CALLBACK.	1. Call Open() to create directory and file handles under the directory at TPL_CALLBACK. 2. Call Delete() to delete the directory handle. The return code should be EFI_WARN_DELETE_FAILURE.
5.7.3.3.11	0x619d8713, 0xd755, 0x4293, 0xbe, 0x3d, 0x19, 0xb0, 0x17, 0xa8, 0xd4, 0x09	EFI_FILE_PROTOCOL. Delete – Re-open of undeleted directory handle returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Open() to create directory and file handles under the directory at TPL_APPLICATION. 2. Call Delete() to delete the directory handle. The return code should be EFI_WARN_DELETE_FAILURE. 3. Call Open() to re-open the directory. The return code should be EFI_SUCCESS.
5.7.3.3.12	0xb9306618, 0x2613, 0x4a6a, 0xaa, 0x72, 0x5e, 0xf7, 0x2e, 0xc8, 0x07, 0xf6	EFI_FILE_PROTOCOL. Delete – Re-open of undeleted directory handle returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Open() to create directory and file handles under the directory at TPL_CALLBACK. 2. Call Delete() to delete the directory handle. The return code should be EFI_WARN_DELETE_FAILURE. 3. Call Open() to re-open the directory. The return code should be EFI_SUCCESS.
5.7.3.3.13	0xca4a0455, 0xee2a, 0x4260, 0x8e, 0xdc, 0x12, 0xb4, 0xd2, 0xc0, 0x1b, 0x79	EFI_FILE_PROTOCOL. Delete – Delete() on root directory returns EFI_WARN_DELETE_FAILURE.	1. Call Delete() on root directory. The return code should be EFI_WARN_DELETE_FAILURE.
5.7.3.3.14	0xda598731, 0xf3da, 0x4f63, 0xa3, 0x49, 0x15, 0x1e, 0x0b, 0x77, 0xe3, 0x6f	EFI_FILE_PROTOCOL. Delete – Open() on root directory returns EFI_SUCCESS after Delete() on root fails.	1. Call Delete() on root directory. It returns EFI_WARN_DELETE_FAILURE. 2. Call Open() on root. The return code should be EFI_SUCCESS.

9.3.4 Read()

Number	GUID	Assertion	Test Description
5.7.3.4.1	0xf98a984c, 0x0043, 0x481e, 0x93, 0x3a, 0x24, 0x4d, 0xc8, 0x6e, 0x71, 0x1a	EFI_FILE_PROTOCOL.Read – Read() from file handle returns EFI_SUCCESS except when read position is beyond file end at TPL_APPLICATION.	1. Call Read() to read from file handles at TPL_APPLICATION. The return code should be EFI_SUCCESS. If read position is beyond file end, the return code should be EFI_DEVICE_ERROR.
5.7.3.4.2	0x192d00c3, 0x604e, 0x49bb, 0xb0, 0xc2, 0x5b, 0x25, 0x69, 0x6e, 0xa9, 0x2f	EFI_FILE_PROTOCOL.Read – Read() from file handle returns EFI_SUCCESS except when read position is beyond file end at TPL_CALLBACK.	1. Call Read() to read from file handles at TPL_CALLBACK. The return code should be EFI_SUCCESS. If read position is beyond file end, the return code should be EFI_DEVICE_ERROR.
5.7.3.4.3	0xfbff4d9d, 0xe021, 0x482b, 0x8d, 0x92, 0x99, 0xd5, 0x26, 0x89, 0xe0, 0xd3	EFI_FILE_PROTOCOL.Read – GetPosition() after read returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Read() to read from file handles at TPL_APPLICATION. 2. Call GetPosition() to get current file position. The return code should be EFI_SUCCESS.
5.7.3.4.4	0x522c18d5, 0xe922, 0x4844, 0xbb, 0x59, 0x5f, 0xdd, 0x48, 0xf8, 0xfe, 0xbc	EFI_FILE_PROTOCOL.Read – GetPosition() after read returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Read() to read from file handles at TPL_CALLBACK. 2. Call GetPosition() to get current file position. The return code should be EFI_SUCCESS.
5.7.3.4.5	0x69decc47, 0xbc8d, 0x44e9, 0x92, 0x3c, 0x63, 0x02, 0x2d, 0xd2, 0xe1, 0x1f	EFI_FILE_PROTOCOL.Read – If read position is beyond file end, buffer size should be truncated at TPL_APPLICATION.	1. Call Read() to read from file handles at TPL_APPLICATION. 2. Call GetPosition() to get current file position. 3. If read position is beyond file end, buffer size should be truncated.
5.7.3.4.6	0x4ba1060f, 0xdaba, 0x4d5b, 0xb9, 0xce, 0xf8, 0x59, 0xbb, 0xbf, 0x52, 0x69	EFI_FILE_PROTOCOL.Read – If read position is beyond file end, buffer size should be truncated at TPL_CALLBACK.	1. Call Read() to read from file handles at TPL_CALLBACK. 2. Call GetPosition() to get current file position. 3. If read position is beyond file end, buffer size should be truncated.

Number	GUID	Assertion	Test Description
5.7.3.4.7	0xe0ebe6bd , 0x0fd2, 0x4c14, 0x84, 0xd5, 0xec, 0xd3, 0x96, 0x6a, 0x04, 0xed	EFI_FILE_PROTOCOL.Read – If read beyond file end, file position updated to the end of the file at TPL_APPLICATION.	1. Call Read() to read from file handles at TPL_APPLICATION. 2. Call GetPosition() to get current file position. 3. If read beyond file end, file position updated to the end of the file.
5.7.3.4.8	0xca6d5592, 0x48a9, 0x46c8, 0xa8, 0xa4, 0x5a, 0xd8, 0x4b, 0x07, 0x68, 0xf1	EFI_FILE_PROTOCOL.Read – If read postion is beyond file end, file position updated to the end of the file at TPL_CALLBACK.	1. Call Read() to read from file handles at TPL_CALLBACK. 2. Call GetPosition() to get current file position. 3. If read position is beyond file end, file position updated to the end of the file.
5.7.3.4.9	0x0b158040 , 0xb603, 0x49e2, 0xab, 0x3c, 0xfb, 0x75, 0x31, 0xdf, 0x68, 0xaa	EFI_FILE_PROTOCOL.Read – BufferSize is equal to the number of bytes read at TPL_APPLICATION.	1. Call Read() to read from file handles at TPL_APPLICATION. BufferSize is equal to the number of bytes read.
5.7.3.4.10	0xad1fd527, 0xf8d7, 0x4875, 0xab, 0x3d, 0x9c, 0x1f, 0xe0, 0x7e, 0x52, 0x41	EFI_FILE_PROTOCOL.Read – *BufferSize* is equal to the number of bytes read at TPL_CALLBACK.	1. Call Read() to read from file handles at TPL_CALLBACK. BufferSize is equal to the number of bytes read.
5.7.3.4.11	0x3ee4c586, 0x9f92, 0x4cc0, 0x9f, 0x32, 0xba, 0xaa, 0xb9, 0x56, 0xce, 0x7b	EFI_FILE_PROTOCOL.Read – If read position is within file size, file position is updated to the start position plus read bytes at TPL_APPLICATION.	1. Call Read() to read from file handles at TPL_APPLICATION. 2. Call GetPosition() to get current file position. 3. If read position is within file size, file position is updated to the start position plus read bytes.
5.7.3.4.12	0x2fa03a35, 0x34d7, 0x4ede, 0x94, 0xfa, 0xca, 0x2b, 0x78, 0xe1, 0xe5, 0xd6	EFI_FILE_PROTOCOL.Read – If read within file size, file position updated to the start position plus read bytes at TPL_CALLBACK.	1. Call Read() to read from file handles at TPL_CALLBACK. 2. Call GetPosition() to get current file position. 3. If read position is within file size, file position is updated to the start position plus read bytes.

Number	GUID	Assertion	Test Description
5.7.3.4.13	0x54013277 , 0xde8a, 0x4f8b, 0xa5, 0x8a, 0x60, 0xe4, 0x17, 0x4c, 0xcd, 0xa2	`EFI_FILE_PROTOCOL.Read` – Read content should be the same as written at `TPL_APPLICATION`.	1. Call `Write()` to write bytes into file handle. 2. Call `Read()` to read from file handles at `TPL_APPLICATION`. Read content should be the same as written.
5.7.3.4.14	0x74ab30a4 , 0xcb1b, 0x4d9b, 0x8c, 0x69, 0x3d, 0xf0, 0xda, 0x61, 0x16, 0x32	`EFI_FILE_PROTOCOL.Read` – Read content should be the same as written at `TPL_CALLBACK`.	1. Call `Write()` to write bytes into file handle. 2. Call `Read()` to read from file handles at `TPL_CALLBACK`. Read content should be the same as written.
5.7.3.4.15	0x2ff71629, 0x8548, 0x4f11, 0x92, 0x24, 0x43, 0x0e, 0xd1, 0x8e, 0xe9, 0x82	`EFI_FILE_PROTOCOL.Read – Read()` from directory handle with too small buffer returns `EFI_BUFFER_TOO_SMALL` at `TPL_APPLICATION`.	1. Call `Read()` to read from directory handle with too small buffer at `TPL_APPLICATION`. The return code should be `EFI_BUFFER_TOO_SMALL`.
5.7.3.4.16	0x3b46d893 , 0x289e, 0x4186, 0x9d, 0x13, 0x94, 0xcc, 0x4e, 0x96, 0x1b, 0xd4	`EFI_FILE_PROTOCOL.Read – Read()` from directory handle with too small buffer returns `EFI_BUFFER_TOO_SMALL` at `TPL_CALLBACK`.	1. Call `Read()` to read from directory handle with too small buffer at `TPL_CALLBACK`. The return code should be `EFI_BUFFER_TOO_SMALL`.
5.7.3.4.17	0x19979967 , 0xf6cb, 0x4043, 0xba, 0x15, 0xdd, 0x80, 0x5e, 0x9f, 0x62, 0xe8	`EFI_FILE_PROTOCOL.Read – Read()` from directory handle with valid parameter returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Read()` to read from directory handle with valid parameter at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.
5.7.3.4.18	0x25c7de0c, 0x56b7, 0x4e8b, 0x94, 0x9e, 0x59, 0x83, 0x0b, 0x46, 0x4a, 0xf4	`EFI_FILE_PROTOCOL.Read – Read()` from directory handle with valid parameter returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Read()` to read from directory handle with valid parameter at `TPL_APPLICATION`. The return code should be `EFI_CALLBACK`.

Number	GUID	Assertion	Test Description
5.7.3.4.19	0xacc83dc2, 0x84d4, 0x46fd, 0xa9, 0x51, 0x1c, 0x2f, 0x49, 0xd5, 0x97, 0x9c	`EFI_FILE_PROTOCOL.Read – Read()` at the end of the directory returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Read()` to at the end of the directory at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS` and read buffer size is set to 0.
5.7.3.4.20	0x882f4162, 0xb6b9, 0x456f, 0xbe, 0xb9, 0x2e, 0x2b, 0xa4, 0xc9, 0x58, 0x5a	`EFI_FILE_PROTOCOL.Read – Read()` at the end of the directory returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Read()` to read at the end of the directory at `TPL_APPLICATION`. The return code should be `EFI_CALLBACK` and read buffer size is set to 0.

9.3.5 Write()

Number	GUID	Assertion	Test Description
5.7.3.5.1	0x73c93917, 0xad5e, 0x4e21, 0xaa, 0xaa, 0x8e, 0x6a, 0x35, 0x85, 0xe9, 0x51	`EFI_FILE_PROTOCOL.Write – Write()` to file handle returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Write()` to write to file handle at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.
5.7.3.5.2	0xb58c7d6a, 0x90f6, 0x4a0b, 0xb8, 0x49, 0xdb, 0xba, 0x08, 0x4c, 0x22, 0xb6	`EFI_FILE_PROTOCOL.Write – Write()` to file handle returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `Write()` to write to file handle at `TPL_CALLBACK`. The return code should be `EFI_SUCCESS`.
5.7.3.5.3	0x9f38fdc4, 0xbbf6, 0x4d1b, 0xae, 0x1c, 0xbb, 0xe8, 0x89, 0xda, 0x8c, 0xc5	`EFI_FILE_PROTOCOL.Write – GetPostion()` after call of `Write()` returns `EFI_SUCCESS` at `TPL_APPLICATION`.	1. Call `Write()` to write to file handle at `TPL_APPLICATION`. 2. Call `GetPosition()`. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.7.3.5.4	0x1ca546ad, 0xac23, 0x4304, 0xa2, 0xff, 0xec, 0xb0, 0x23, 0xd5, 0xfb, 0x21	**EFI_FILE_PROTOCOL. Write –** **GetPostion()** after call of **Write()** returns **EFI_SUCCESS** at **TPL_CALLBACK**.	1. Call **Write()** to write to file handle at **TPL_CALLBACK**. 2. Call **GetPosition()**. The return code should be **EFI_SUCCESS**.
5.7.3.5.5	0xca3fdc12, 0x3e36, 0x4a38, 0xb9, 0x46, 0xb5, 0x83, 0x31, 0x67, 0x13, 0xc0	**EFI_FILE_PROTOCOL. Write** – *BufferSize* is updated as the number of bytes written at **TPL_APPLICATION**.	1. Call **Write()** to write to file handle at **TPL_APPLICATION**. *BufferSize* should be updated as the number of bytes written.
5.7.3.5.6	0x4e3680d0, 0xf1dc, 0x4736, 0x86, 0xcb, 0x6e, 0xb0, 0xc0, 0xc0, 0x8e, 0xdd	**EFI_FILE_PROTOCOL. Write** – *BufferSize* is updated as the number of bytes written at **TPL_CALLBACK**.	1. Call **Write()** to write to file handle at **TPL_CALLBACK**. **BufferSize** should be updated as the number of bytes written.
5.7.3.5.7	0x99e9e364, 0xeefb, 0x4b2a, 0xb3, 0x29, 0xe4, 0x8b, 0x31, 0x2c, 0xe4, 0x7c	**EFI_FILE_PROTOCOL. Write** – File position is updated after call of **Write()** at **TPL_APPLICATION**.	1. Call **Write()** to write to file handle at **TPL_APPLICATION**. 2. Call **GetPosition()**. Current file postion is updated to the end of written contents.
5.7.3.5.8	0x06a30897, 0xe2ed, 0x4c76, 0x99, 0xa4, 0xc4, 0x5d, 0x00, 0x9f, 0xef, 0x8d	**EFI_FILE_PROTOCOL. Write** – File position is updated after call of **Write()** at **TPL_CALLBACK**.	1. Call **Write()** to write to file handle at **TPL_CALLBACK**. 2. Call **GetPosition()**.Current file postion is updated to the end of written contents.
5.7.3.5.9	0x0af7cb57, 0x661e, 0x4b4f, 0xb4, 0xa4, 0xf6, 0xe3, 0x16, 0xe1, 0x52, 0x76	**EFI_FILE_PROTOCOL. Write** – File size is updated after call of **Write()** at **TPL_APPLICATION**.	1. Call **Write()** to write to file handle at **TPL_APPLICATION**. 2. Call **GetInfo()**. If write is beyond the end of the file, file size has grown.

Number	GUID	Assertion	Test Description
5.7.3.5.10	0x1d6b4c54, 0x51fe, 0x406e, 0xb5, 0x92, 0x22, 0xe2, 0xa8, 0x74, 0x7e, 0xdc	EFI_FILE_PROTOCOL.Write – File size is updated after call of Write() at TPL_CALLBACK.	1. Call Write() to write to file handle at TPL_CALLBACK. 2. Call GetInfo(). If write is beyond the end of the file, file size has grown.
5.7.3.5.11	0x67428a37, 0x56f9, 0x400a, 0xb1, 0x00, 0xd7, 0xf7, 0x68, 0x0c, 0x65, 0x5c	EFI_FILE_PROTOCOL.Write – Read() after Write() returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Write() to write to file handle at TPL_APPLICATION. 2. Call Read().The return code should be EFI_SUCCESS.
5.7.3.5.12	0xe5242bc2, 0x0b10, 0x462f, 0x89, 0xa4, 0xfb, 0xe1, 0x41, 0x55, 0xdb, 0x84	EFI_FILE_PROTOCOL.Write - Read() after Write() returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Write() to write to file handle at TPL_CALLBACK. 2. Call Read().The return code should be EFI_SUCCESS.
5.7.3.5.13	0x4838f93c, 0xd601, 0x4d76, 0x8c, 0x7a, 0x59, 0xfa, 0xfa, 0xa5, 0xb6, 0x6d	EFI_FILE_PROTOCOL.Write – Read() after Write() gets the same contents as written at TPL_APPLICATION.	1. Call Write() to write to file handle at TPL_APPLICATION. 2. Call Read(). It should return the same contents as written.
5.7.3.5.14	0x3ff81ec0, 0xf7ae, 0x42da, 0x82, 0xdd, 0xb8, 0x59, 0xa1, 0x14, 0x72, 0xe7	EFI_FILE_PROTOCOL.Write - Read() after Write() gets the same contents as written at TPL_CALLBACK.	1. Call Write() to write to file handle at TPL_CALLBACK. 2. Call Read().It should return the same contents as written.
5.7.3.5.15	0x29eb1c7e, 0xf4aa, 0x4fc4, 0xa9, 0x68, 0x98, 0xde, 0xa1, 0x75, 0x52, 0x0e	EFI_FILE_PROTOCOL.Write – Write() to the directory returns EFI_UNSUPPORTED.	1. Call Write() to write to a directory. The return code should be EFI_UNSUPPORTED.

Number	GUID	Assertion	Test Description
5.7.3.5.16	0xd1b25896, 0x9f3d, 0x467a, 0xbc, 0x92, 0x8a, 0x52, 0x04, 0x96, 0x6a, 0x10	EFI_FILE_PROTOCOL.Write – Write() to a read-only opened file returns EFI_ACCESS_DENIED.	1. Call Write() to write to a read-only opened file. The return code should be EFI_ACCESS_DENIED.

9.3.6 Flush()

Number	GUID	Assertion	Test Description
5.7.3.6.1	0xa2070225, 0x0018, 0x4953, 0xb8, 0xfa, 0xbd, 0x17, 0x21, 0xca, 0x68, 0x46	EFI_FILE_PROTOCOL.Flush – Flush() on file handle returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Flush() on file handle at TPL_APPLICATION. The return code should be EFI_SUCCESS.
5.7.3.6.2	0x83b6cdc5, 0xd813, 0x4000, 0xa9, 0x84, 0x07, 0xb6, 0x54, 0xc5, 0x1f, 0xe4	EFI_FILE_PROTOCOL.Flush – Flush() on file handle returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Flush() on file handle at TPL_CALLBACK. The return code should be EFI_SUCCESS.
5.7.3.6.3	0x9f7bfe1e, 0xd617, 0x4920, 0xab, 0x63, 0x2f, 0xae, 0x6d, 0xce, 0x77, 0x5d	EFI_FILE_PROTOCOL.Flush – Flush() on directory handle returns EFI_SUCCESS at TPL_APPLICATION.	1. Call Flush() on directory handle at TPL_APPLICATION. The return code should be EFI_SUCCESS.
5.7.3.6.4	0x42985ef5, 0x8c9b, 0x49df, 0x93, 0x3c, 0x30, 0xf8, 0xc0, 0x22, 0x8e, 0x4b	EFI_FILE_PROTOCOL.Flush – Flush() on directory handle returns EFI_SUCCESS at TPL_CALLBACK.	1. Call Flush() on directory handle at TPL_CALLBACK. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.6.5	0xc7900513, 0xe931, 0x404a, 0xa5, 0xe3, 0xe3, 0x48, 0x40, 0x7a, 0xb2, 0xa2	**EFI_FILE_PROTOCOL. Flush – Flush()** to a read-only opened file returns **EFI_ACCESS_DENIED**.	1. Call **Flush()** to flush a read-only opened file. The return code should be **EFI_ACCESS_DENIED**.

9.3.7 SetPosition()

Number	GUID	Assertion	Test Description
5.7.3.7.1	0x6b383ca4, 0xc8e4, 0x4fe2, 0xa8, 0xdb, 0x8b, 0x87, 0x85, 0x38, 0xb7, 0x7b	**EFI_FILE_PROTOCOL. SetPosition – SetPosiiton()** on file handle returns **EFI_SUCCESS** at **TPL_APPLICATION**.	1. Call **SetPosition()** on file handle at **TPL_APPLICATION**. The return code should be **EFI_SUCCESS**.
5.7.3.7.2	0x546093bf, 0x1ab1, 0x445c, 0x9e, 0x36, 0xb3, 0x90, 0x4c, 0xe5, 0x74, 0x7e	**EFI_FILE_PROTOCOL. SetPosition – SetPosiiton()** on file handle returns **EFI_SUCCESS** at **TPL_CALLBACK**.	1. Call **SetPosition()** on file handle at **TPL_APPLICATION**. The return code should be **EFI_CALLBACK**.
5.7.3.7.3	0x7fa447e4, 0xae1e, 0x490b, 0x89, 0x53, 0x60, 0xd7, 0xef, 0xd3, 0x0f, 0xed	**EFI_FILE_PROTOCOL. SetPosition – GetPosiiton()** on file handle after call of **SetPosition()** returns **EFI_SUCCESS** at **TPL_APPLICATION**.	1. Call **SetPosition()** on file handle at **TPL_APPLICATION**. 2. Call **GetPosition()**. The return code should be **EFI_SUCCESS**.
5.7.3.7.4	0x89e7eb29, 0xd715, 0x47bc, 0x94, 0xb1, 0xcd, 0xdf, 0x67, 0xd8, 0x44, 0x0a	**EFI_FILE_PROTOCOL. SetPosition – GetPosiiton()** on file handle after call of **SetPosition()** returns **EFI_SUCCESS** at **TPL_CALLBACK**.	1. Call **SetPosition()** on file handle at **TPL_APPLICATION**. 2. Call **GetPosition()**. The return code should be **EFI_CALLBACK**.

Number	GUID	Assertion	Test Description
5.7.3.7.5	0x19ca2741, 0x4dd5, 0x4df8, 0x9d, 0xb4, 0x84, 0xdb, 0xea, 0xc6, 0x7c, 0x63	`EFI_FILE_PROTOCOL.SetPosition – GetPosiiton()` on file handle after call of `SetPosition()` returns the same position as set at `TPL_APPLICATION`	1. Call `SetPosition()` on file handle at `TPL_APPLICATION`. 2. Call `GetPosition()`. The return position should be the same as set.
5.7.3.7.6	0x6e22f1ef, 0x664e, 0x4c58, 0x90, 0xea, 0x32, 0x92, 0x39, 0x02, 0xa6, 0x4f	`EFI_FILE_PROTOCOL.SetPosition – GetPosiiton()` on file handle after call of `SetPosition()` returns the same position as set at `TPL_CALLBACK`.	1. Call `SetPosition()` on file handle at `TPL_APPLICATION`. 2. Call `GetPosition()`. The return position should be the same as set.
5.7.3.7.7	0xde3f7243, 0xc732, 0x45d7, 0x97, 0x17, 0x4f, 0x85, 0x98, 0x8c, 0x85, 0xd8	`EFI_FILE_PROTOCOL.SetPosition – SetPosiiton()` on directory handle with 0 position returns `EFI_SUCCESS` at `TPL_APPLICATION`	1. Call `SetPosition()` on directory handle with 0 position at `TPL_APPLICATION`. The return code should be `EFI_SUCCESS`.
5.7.3.7.8	0x0f4c0762, 0x9746, 0x42a0, 0xba, 0xbf, 0x64, 0x32, 0xb8, 0xd8, 0x1f, 0xf9	`EFI_FILE_PROTOCOL.SetPosition – SetPosiiton()` on directory handle with 0 position returns `EFI_SUCCESS` at `TPL_CALLBACK`.	1. Call `SetPosition()` on directory handle with 0 position at `TPL_APPLICATION`. The return code should be `EFI_CALLBACK`.
5.7.3.7.9	0x5e0586cd, 0x7718, 0x4605, 0x9b, 0xa1, 0x4d, 0xa3, 0xd4, 0x2b, 0xf2, 0x51	`EFI_FILE_PROTOCOL.SetPosition – SetPosiiton()` on directory handle with non-0 position returns `EFI_UNSUPPORTED`.	1. Call `SetPosition()` on directory handle with non-0 position. The return code should be `EFI_UNSUPPORTED`.

9.3.8 GetPosition()

Number	GUID	Assertion	Test Description
5.7.3.8.1	0x9787af2d, 0xda90, 0x4945, 0xba, 0xaa, 0xe4, 0x13, 0x4e, 0x25, 0xe8, 0x8e	EFI_FILE_PROTOCOL. GetPosition – GetPosition() on file handle returns EFI_SUCCESS at TPL_APPLICATION	1. Call GetPosition() on file handle at TPL_APPLICATION. The return code should be EFI_SUCCESS.
5.7.3.8.2	0xfaf1daae, 0x3dbc, 0x484d, 0x9c, 0xe0, 0xd5, 0xc7, 0x30, 0xf3, 0x1d, 0x05	EFI_FILE_PROTOCOL. GetPosition – GetPosition() on file handle returns EFI_SUCCESS at TPL_CALLBACK.	1. Call GetPosition() on file handle at TPL_CALLBACK. The return code should be EFI_SUCCESS.
5.7.3.8.3	0x50e547cb, 0x0d88, 0x447b, 0xaa, 0x07, 0x22, 0x08, 0x8a, 0xc0, 0x05, 0xb5	EFI_FILE_PROTOCOL. GetPosition – GetPosition() on file handle after call of SetPosition() returns the set value at TPL_APPLICATION.	1. Call SetPosition() on file handle. 2. Call GetPosition() on file handle at TPL_APPLICATION. The return position should be the same value as set.
5.7.3.8.4	0x8c1a0c2b, 0x0362, 0x4ba8, 0x90, 0x80, 0x41, 0x66, 0x00, 0xcd, 0x12, 0x87	EFI_FILE_PROTOCOL. GetPosition – GetPosition() on file handle after call of SetPosition() returns the set value at TPL_CALLBACK.	1. Call SetPosition() on file handle. 2. Call GetPosition() on file handle at TPL_CALLBACK. The return position should be the same value as set.
5.7.3.8.5	0x9664e456, 0x0e74, 0x4d1f, 0x8e, 0x7b, 0x2c, 0x49, 0xf8, 0x94, 0xdb, 0x49	EFI_FILE_PROTOCOL. GetPosition – GetPosition() on directory handle returns EFI_UNSUPPORTED.	1. Call GetPosition() on directory handle. The return code should be EFI_UNSUPPORTED.

9.3.9 GetInfo()

Number	GUID	Assertion	Test Description
5.7.3.9.1	0xf93e4251, 0x75a8, 0x464e, 0xaa, 0xf9, 0x03, 0xa2, 0x9e, 0x6e, 0xdd, 0x6b	**EFI_FILE_PROTOCOL.GetInfo – GetInfo()** on file handle for **EFI_FILE_INFO** returns **EFI_SUCCESS** at **TPL_APPLICATION**.	1. Call **GetInfo()** on file handle for **EFI_FILE_INFO** at **TPL_APPLICATION**. The return code should be **EFI_SUCCESS**.
5.7.3.9.2	0xa40d7f41, 0x959f, 0x4c1a, 0x82, 0x02, 0x83, 0xc5, 0xda, 0x58, 0xbe, 0x9c	**EFI_FILE_PROTOCOL.GetInfo – GetInfo()** on file handle for **EFI_FILE_INFO** returns **EFI_SUCCESS** at **TPL_CALLBACK**.	1. Call **GetInfo()** on file handle for **EFI_FILE_INFO** at **TPL_CALLBACK**. The return code should be **EFI_SUCCESS**.
5.7.3.9.3	0x2055cdd1, 0xce8b, 0x4e95, 0xad, 0x6b, 0xfb, 0x5f, 0x95, 0x8c, 0xcc, 0x68	**EFI_FILE_PROTOCOL.GetInfo – GetInfo()** on file handle for **EFI_FILE_INFO** returns the same buffer size as *FileInfo->Size* at **TPL_APPLICATION**.	1. Call **GetInfo()** on file handle for **EFI_FILE_INFO** at **TPL_APPLICATION**. The return buffer size should be the same as *FileInfo->Size*.
5.7.3.9.4	0xe8099e1b, 0x193e, 0x4383, 0x88, 0x67, 0x06, 0xb1, 0xb8, 0x1f, 0x48, 0x38	**EFI_FILE_PROTOCOL.GetInfo – GetInfo()** on file handle for **EFI_FILE_INFO** returns the same buffer size as *FileInfo->Size* at **TPL_CALLBACK**.	1. Call **GetInfo()** on file handle for **EFI_FILE_INFO** at **TPL_CALLBACK**. The return buffer size should be the same as *FileInfo->Size*.
5.7.3.9.5	0x788dc48a, 0xdaac, 0x4c4d, 0x82, 0x5d, 0xe9, 0x46, 0x14, 0x00, 0x71, 0xbe	**EFI_FILE_PROTOCOL.GetInfo – GetInfo()** on file handle for **EFI_FILE_INFO** returns the same attribute set by **SetInfo()** at **TPL_APPLICATION**.	1. Call **SetInfo()** on file handle. 1. Call **GetInfo()** on file handle for **EFI_FILE_INFO** at **TPL_APPLICATION**. The return attribute should be the same as set value.

Number	GUID	Assertion	Test Description
5.7.3.9.6	0x32abd0a4, 0x768d, 0x45a8, 0x9c, 0xf6, 0x2d, 0xc3, 0x56, 0x80, 0x58, 0xa0	EFI_FILE_PROTOCOL.GetInfo – GetInfo() on file handle for EFI_FILE_INFO returns the same attribute set by SetInfo() at TPL_CALLBACK.	1. Call SetInfo() on file handle. 1. Call GetInfo() on file handle for EFI_FILE_INFO at TPL_CALLBACK. The return attribute should be the same as set value.
5.7.3.9.7	0xfea5ef36, 0x87e1, 0x4282, 0xbc, 0xb9, 0xde, 0x54, 0x5d, 0x20, 0xb7, 0x06	EFI_FILE_PROTOCOL.GetInfo – GetInfo() on file handle for EFI_FILE_INFO returns the same filename as created at TPL_APPLICATION.	1. Call Open() to create file handle. 1. Call GetInfo() on file handle for EFI_FILE_INFO at TPL_APPLICATION. The return filename should be the same as created.
5.7.3.9.8	0x93c186a6, 0x4e31, 0x4395, 0x87, 0x1a, 0x90, 0xcc, 0x91, 0xa2, 0x2f, 0xc2	EFI_FILE_PROTOCOL.GetInfo – GetInfo() on file handle for EFI_FILE_INFO returns the same filename as created at TPL_CALLBACK.	1. Call Open() to create file handle. 1. Call GetInfo() on file handle for EFI_FILE_INFO at TPL_CALLBACK. The return filename should be the same as created filename.
5.7.3.9.9	0x35187534, 0xba64, 0x4be4, 0xaa, 0x9c, 0xc2, 0x07, 0x53, 0x12, 0x0f, 0x57	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_INFO from root returns EFI_SUCCESS at TPL_APPLICATION.	1. Call GetInfo() to retrieve EFI_FILE_INFO from root. The return code should be EFI_SUCCESS.
5.7.3.9.10	0xcd2e69ad, 0xe1ce, 0x42ea, 0x80, 0xfe, 0xfc, 0x2b, 0x1e, 0x72, 0xb0, 0x3b	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_INFO from root returns EFI_SUCCESS at TPL_CALLBACK.	1. Call GetInfo() to retrieve EFI_FILE_INFO from root. The return code should be EFI_SUCCESS.
5.7.3.9.11	0x7fc5deb9, 0xf216, 0x462e, 0xbf, 0x6b, 0xa3, 0x02, 0x6f, 0x13, 0x88, 0xea	EFI_FILE_PROTOCOL.GetInfo – GetInfo() on root handle for EFI_FILE_INFO returns the same buffer size as *FileInfo->Size* at TPL_APPLICATION	1. Call GetInfo() on root handle for EFI_FILE_INFO at TPL_APPLICATION. The return buffer size should be the same as *FileInfo->Size*.

Number	GUID	Assertion	Test Description
5.7.3.9.12	0x8d390587, 0xe4ff, 0x4c55, 0xa1, 0x55, 0xba, 0x80, 0x7a, 0x19, 0xbe, 0xf1	EFI_FILE_PROTOCOL.GetInfo – GetInfo() on root handle for EFI_FILE_INFO returns the same buffer size as *FileInfo*->*Size* at TPL_CALLBACK.	1. Call GetInfo() on root handle for EFI_FILE_INFO at TPL_CALLBACK. The return buffer size should be the same as *FileInfo*->*Size*.
5.7.3.9.13	0xc01d216d, 0x9fdf, 0x4504, 0x99, 0x61, 0x3f, 0x4a, 0x08, 0xb7, 0x61, 0x43	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO returns EFI_SUCCESS at TPL_APPLICATION.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO. The return code should be EFI_SUCCESS.
5.7.3.9.14	0x04ae8ab0, 0xe2d6, 0x46e6, 0x98, 0x35, 0x63, 0x14, 0x21, 0x09, 0x5f, 0xac	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO returns EFI_SUCCESS at TPL_CALLBACK	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO. The return code should be EFI_SUCCESS.
5.7.3.9.15	0x75cd35a8, 0x8f56, 0x441d, 0x8a, 0x4a, 0x2a, 0xcd, 0x8a, 0x79, 0xea, 0x01	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO returns correct buffer size of EFI_FILE_SYSTEM_INFO structure at TPL_APPLICATION.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO. It should return correct buffer size of EFI_FILE_SYSTEM_INFO structure.
5.7.3.9.16	0xe4f4f6a2, 0x7538, 0x4c79, 0xaa, 0x3c, 0x67, 0x18, 0x4e, 0xc7, 0x0e, 0x16	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO returns correct buffer size of EFI_FILE_SYSTEM_INFO structure at TPL_CALLBACK	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO It should return correct buffer size of EFI_FILE_SYSTEM_INFO structure.

Number	GUID	Assertion	Test Description
5.7.3.9.17	0x59afd349, 0xf5a1, 0x4052, 0x9b, 0xb9, 0x22, 0x51, 0x24, 0x0f, 0xe3, 0x47	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO returns correct file system info set by SetInfo() at TPL_APPLICATION.	1. Call SetInfo(). 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO. It should return correct file system info set by SetInfo().
5.7.3.9.18	0xbe4e594f, 0x43c4, 0x42fc, 0xbe, 0x9e, 0xdc, 0xb7, 0xa8, 0x5a, 0x76, 0x7d	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO returns correct filesystem info set by SetInfo() at TPL_CALLBACK	1. Call SetInfo(). 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO It should return returns correct file system info set by SetInfo().
5.7.3.9.19	0x4e8fa0c4, 0x95bc, 0x415b, 0x93, 0x65, 0x12, 0x11, 0xea, 0x40, 0x6b, 0xad	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO after create new file and free space decreases at TPL_APPLICATION.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO and record free space number. 2. Call Open() to create new file. 3. Call GetInfo() again. Free space should decrease.
5.7.3.9.20	0x9fa8a442, 0x572f, 0x4d04, 0xba, 0x0d, 0x17, 0x17, 0x72, 0x8d, 0x7e, 0x27	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO after create new file and free space decreases at TPL_CALLBACK.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO and record free space number. 2. Call Open() to create new file. 3. Call GetInfo() again. Free space should decrease.
5.7.3.9.21	0x2970bb0b, 0xb080, 0x48a9, 0x93, 0x64, 0x5e, 0x78, 0xbe, 0xb3, 0xf1, 0x02	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL returns EFI_SUCCESS at TPL_APPLICATION.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL. The return code should be EFI_SUCCESS.
5.7.3.9.22	0xf5fe94d3, 0x0269, 0x44ff, 0xb1, 0x3b, 0x23, 0x63, 0xd0, 0x33, 0xfe, 0xd5	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL returns EFI_SUCCESS at TPL_CALLBACK.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.9.23	0xa5d8f95a, 0x5bba, 0x4f1b, 0x83, 0x35, 0x12, 0x3e, 0x29, 0x6e, 0xda, 0xb2	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO returns EFI_SUCCESS at TPL_APPLICATION.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO. The return code should be EFI_SUCCESS.
5.7.3.9.24	0xf8dea2ab, 0xef13, 0x4544, 0xbd, 0x76, 0x42, 0xad, 0x6c, 0xd6, 0x17, 0x96	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO returns EFI_SUCCESS at TPL_CALLBACK.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO. The return code should be EFI_SUCCESS.
5.7.3.9.25	0x22837b7c, 0x46fc, 0x4439, 0x95, 0x3b, 0xb0, 0x18, 0xce, 0xd3, 0xd7, 0x67	EFI_FILE_PROTOCOL.GetInfo – GetInfo() returns the same volume label for EFI_FILE_SYSTEM_INFO and EFI_FILE_SYSTEM_VOLUME_LABEL at TPL_APPLICATION.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL. 3. They should return the same volume label.
5.7.3.9.26	0x0772aef8, 0x1c09, 0x47e9, 0x83, 0xef, 0x76, 0xaa, 0x3d, 0x21, 0xfa, 0xa4	EFI_FILE_PROTOCOL.GetInfo – GetInfo() returns the same volume label for EFI_FILE_SYSTEM_INFO and EFI_FILE_SYSTEM_VOLUME_LABEL at TPL_CALLBACK.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL. 3. They should return the same volume label.
5.7.3.9.27	0xfeb18200, 0x0904, 0x46cb, 0x81, 0x2b, 0x1e, 0xea, 0x00, 0xc3, 0x29, 0xc3	EFI_FILE_PROTOCOL.GetInfo – GetInfo() with unsupported info type for the file returns EFI_UNSUPPORTED.	1. Call GetInfo() to retrieve unsupported info type for the file. The return code should be EFI_UNSUPPORTED.
5.7.3.9.28	0xdbdc09cc, 0x03d3, 0x4d56, 0x88, 0x76, 0xab, 0xa1, 0x3b, 0xf6, 0x68, 0xae	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_INFO with too small of a buffer returns EFI_BUFFER_TOO_SMALL.	1. Call GetInfo() to retrieve EFI_FILE_INFO with too small of a buffer. The return code should be EFI_BUFFER_TOO_SMALL.

Number	GUID	Assertion	Test Description
5.7.3.9.29	0x50e087ce, 0x802d, 0x46de, 0xa9, 0x13, 0x29, 0xa1, 0x8d, 0x2c, 0xc2, 0xff	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO with too small of a buffer returns EFI_BUFFER_TOO_SMALL.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO with too small of a buffer. The return code should be EFI_BUFFER_TOO_SMALL.
5.7.3.9.30	0x7a60bd66, 0x3b1e, 0x4818, 0xa1, 0x4b, 0xf8, 0x65, 0xf2, 0xc4, 0x76, 0x4e	EFI_FILE_PROTOCOL.GetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL with too small of a buffer returns EFI_BUFFER_TOO_SMALL.	1. Call GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL with too small of a buffer. The return code should be EFI_BUFFER_TOO_SMALL.

9.3.10 SetInfo()

Number	GUID	Assertion	Test Description
5.7.3.10.1	0x5eb09d11, 0x22ee, 0x43f7, 0xa6, 0xc1, 0x95, 0x92, 0xb5, 0x04, 0x70, 0xe7	EFI_FILE_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI_FILE_INFO returns EFI_SUCCESS at TPL_APPLICATION.	1. Call SetInfo() on file handle to set EFI_FILE_INFO at TPL_APPLICATION. The return code should be EFI_SUCCESS.
5.7.3.10.2	0x19f9c6f4, 0x2b6d, 0x4eb3, 0x80, 0xfb, 0x25, 0x55, 0x58, 0xf8, 0x47, 0x2f	EFI_FILE_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI_FILE_INFO returns EFI_SUCCESS at TPL_CALLBACK.	1. Call SetInfo() on file handle to set EFI_FILE_INFO at TPL_CALLBACK. The return code should be EFI_SUCCESS.
5.7.3.10.3	0x99f2a97f, 0xb249, 0x4cc3, 0xa4, 0x50, 0x56, 0x51, 0x7c, 0x2a, 0xfb, 0x35	EFI_FILE_PROTOCOL.SetInfo – GetInfo() on file handle to retrieve EFI_FILE_INFO returns values set by SetInfo() at TPL_APPLICATION.	1. Call SetInfo() on file handle to set EFI_FILE_INFO. 2. Call GetInfo() to retrieve EFI_FILE_INFO. It should return the values set by SetInfo().

Number	GUID	Assertion	Test Description
5.7.3.10.4	0x26615965, 0xe6b3, 0x43cb, 0x90, 0xb1, 0xcb, 0x00, 0x42, 0xe3, 0x34, 0xc5	EFI_FILE_PROTOCOL.SetInfo – SetInfo() on file handle to retrieve EFI_FILE_INFO returns values set by SetInfo() at TPL_CALLBACK.	1. Call SetInfo() on file handle to set EFI_FILE_INFO. 2. Call GetInfo() to retrieve EFI_FILE_INFO. It should return the values set by SetInfo().
5.7.3.10.5	0xb46741e9, 0x3545, 0x4b0e, 0x80, 0x12, 0xc9, 0x56, 0x6a, 0x7f, 0x83, 0xc5	EFI_FILE_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI_FILE_INFO returns EFI_SUCCESS at TPL_APPLICATION.	1. Call SetInfo() on file handle to set EFI_FILE_INFO at TPL_APPLICATION. The return code should be EFI_SUCCESS.
5.7.3.10.6	0xa1cc0c27, 0x55dc, 0x4cd8, 0x96, 0xe8, 0x57, 0x1b, 0x65, 0xc1, 0xdb, 0xde	EFI_FILE_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI_FILE_INFO returns EFI_SUCCESS at TPL_CALLBACK.	1. Call SetInfo() on file handle to set EFI_FILE_INFO at TPL_CALLBACK. The return code should be EFI_SUCCESS.
5.7.3.10.7	0x16494a12, 0xfc45, 0x4e30, 0x91, 0xac, 0x88, 0x1c, 0x9c, 0x88, 0xae, 0x4b	EFI_FILE_PROTOCOL.SetInfo – GetInfo() on file handle to retrieve EFI_FILE_INFO returns values set by SetInfo() at TPL_APPLICATION.	1. Call SetInfo() on file handle to set EFI_FILE_INFO. 2. Call GetInfo() to retrieve EFI_FILE_INFO. It should return the values set by SetInfo().
5.7.3.10.8	0xd843eacb, 0x2468, 0x4d4b, 0xa3, 0x51, 0xbf, 0x41, 0xd5, 0xdd, 0x9a, 0x16	EFI_FILE_PROTOCOL.SetInfo – GetInfo() on file handle to retrieve EFI_FILE_INFO returns values set by SetInfo() at TPL_CALLBACK.	1. Call SetInfo() on file handle to set EFI_FILE_INFO. 2. Call GetInfo() to retrieve EFI_FILE_INFO. It should return the values set by SetInfo().
5.7.3.10.9	0x4be420a1, 0xd7e7, 0x4327, 0x8e, 0x63, 0x59, 0x41, 0xef, 0x4b, 0xfd, 0x2a	EFI_FILE_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI_FILE_SYSTEM_INFO returns EFI_SUCCESS at TPL_APPLICATION	1. Call SetInfo() on file handle to set EFI_FILE_SYSTEM_INFO at TPL_APPLICATION. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.10.10	0x57880cd6, 0x6eb1, 0x40b5, 0xa3, 0xef, 0x28, 0x26, 0x94, 0x9a, 0xf8, 0x9c	EFI_FILE_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI_FILE_SYSTEM_INFO returns EFI_SUCCESS at TPL_CALLBACK.	1. Call SetInfo() on file handle to set EFI_FILE_SYSTEM_INFO at TPL_CALLBACK. The return code should be EFI_SUCCESS.
5.7.3.10.11	0x0cf2c5c5, 0xd976, 0x4fd4, 0x85, 0x07, 0x0a, 0x81, 0x88, 0x62, 0x45, 0x78	EFI_FILE_PROTOCOL.SetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO returns volume label set by SetInfo() at TPL_APPLICATION.	1. Call SetInfo() for EFI_FILE_SYSTEM_INFO to set volume label. 2. Call GetInfo() to to retrieve EFI_FILE_SYSTEM_INFO. The return volume label should be the same as set by SetInfo().
5.7.3.10.12	0xc68c8288, 0x020f, 0x460f, 0x81, 0xf8, 0x67, 0x35, 0x10, 0xd1, 0xfe, 0x6a	EFI_FILE_PROTOCOL.SetInfo – GetInfo() to retrieve EFI_FILE_SYSTEM_INFO returns volume label set by SetInfo() at TPL_CALLBACK.	1. Call SetInfo() for EFI_FILE_SYSTEM_INFO to set volume label. 2. Call GetInfo() to to retrieve EFI_FILE_SYSTEM_INFO. The return volume label should be the same as set by SetInfo().
5.7.3.10.13	0x12b68173, 0x7c8d, 0x4023, 0xaf, 0xcc, 0xf1, 0xc6, 0xbe, 0xb6, 0x1c, 0xef	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to set EFI_FILE_SYSTEM_VOLUME_LABEL returns EFI_SUCCESS at TPL_APPLICATION.	1. Call SetInfo() on file handle to set EFI_FILE_SYSTEM_VOLUME_LABEL. The return code should be EFI_SUCCESS.
5.7.3.10.14	0x6e869806, 0x1bc2, 0x40d5, 0xb3, 0x02, 0x61, 0xf8, 0xce, 0x56, 0xeb, 0xfe	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to set EFI_FILE_SYSTEM_VOLUME_LABEL returns EFI_SUCCESS at TPL_CALLBACK.	1. Call SetInfo() on file handle to set EFI_FILE_SYSTEM_VOLUME_LABEL. The return code should be EFI_SUCCESS.
5.7.3.10.15	0x725364f6, 0x6a23, 0x424a, 0x82, 0xaf, 0xd0, 0xd0, 0x47, 0xd3, 0xb8, 0x08	EFI_FILE_PROTOCOL.SetInfo – GetInfo() for EFI_FILE_SYSTEM_VOLUME_LABEL returns the same volume label as set by SetInfo() at TPL_APPLICATION.	1. Call SetInfo() on file handle to set EFI_FILE_SYSTEM_VOLUME_LABEL. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL. The return volume label should be the same as set by SetInfo().

Number	GUID	Assertion	Test Description
5.7.3.10.16	0xcdab6fd9, 0x93aa, 0x4820, 0xb1, 0xa1, 0x71, 0xea, 0x3e, 0x7f, 0xab, 0x26	EFI_FILE_PROTOCOL.SetInfo – GetInfo() for EFI_FILE_SYSTEM_VOLUME_LABEL returns the same volume label as set by SetInfo() at TPL_CALLBACK.	1. Call SetInfo() on file handle to set EFI_FILE_SYSTEM_VOLUME_LABEL. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_VOLUME_LABEL. The return volume label should be the same as set by SetInfo().
5.7.3.10.17	0xf700f5f8, 0xecac, 0x45fb, 0x9d, 0x2d, 0x34, 0xe9, 0x46, 0x66, 0x07, 0x38	EFI_FILE_PROTOCOL.SetInfo – GetInfo() for EFI_FILE_SYSTEM_INFO returns the same volume label as set by SetInfo() at TPL_APPLICATION.	1. Call SetInfo() on file handle to set EFI_FILE_SYSTEM_VOLUME_LABEL. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO. The return volume label should be the same as set by SetInfo().
5.7.3.10.18	0x384840cd, 0x9a3f, 0x44c3, 0x87, 0xd8, 0xcd, 0xd9, 0xab, 0xd2, 0x17, 0x96	EFI_FILE_PROTOCOL.SetInfo – GetInfo() for EFI_FILE_SYSTEM_INFO returns the same volume label as set by SetInfo() at TPL_CALLBACK.	1. Call SetInfo() on file handle to set EFI_FILE_SYSTEM_VOLUME_LABEL. 2. Call GetInfo() to retrieve EFI_FILE_SYSTEM_INFO. The return volume label should be the same as set by SetInfo().
5.7.3.10.19	0x2a58594e, 0xd06a, 0x4f44, 0xa2, 0x6e, 0xa3, 0x49, 0x36, 0xde, 0x05, 0xef	EFI_FILE_PROTOCOL.SetInfo – SetInfo() with unsupported info type returns EFI_UNSUPPORTED.	1. Call SetInfo() with unsupported info type. The return code should be EFI_UNSUPPORTED.
5.7.3.10.20	0x164feeba, 0xf3ed, 0x482a, 0x83, 0xac, 0x89, 0x48, 0x0a, 0x1d, 0x9a, 0xc9	EFI_FILE_PROTOCOL.SetInfo – SetInfo() with the read-only opened file handle to change file size returns EFI_ACCESS_DENIED.	1. Call Open() to open file handle in read-only mode. 2. Call SetInfo() to change file size. The return code should be EFI_ACCESS_DENIED.
5.7.3.10.21	0x1a74e8f3, 0x62ad, 0x47ef, 0x92, 0xe6, 0x6d, 0x47, 0x23, 0x21, 0xd2, 0xb0	EFI_FILE_PROTOCOL.SetInfo – SetInfo() with the read-only opened file handle to change file name returns EFI_ACCESS_DENIED.	1. Call Open() to open file handle in read-only mode. 2. Call SetInfo() to change file name. The return code should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.7.3.10.22	0x75c4d3e4, 0x17fa, 0x4f02, 0xb1, 0x15, 0x72, 0x0c, 0x0f, 0x1a, 0xe2, 0xe1	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to set EFI_FILE_INFO with too small of a buffer returns EFI_BUFFER_TOO_SMALL.	1. Call SetInfo() to set EFI_FILE_INFO with too small of a buffer. The return code should be EFI_BUFFER_TOO_SMALL.
5.7.3.10.23	0x36d0ed31, 0x21f0, 0x48c2, 0x89, 0x74, 0x6b, 0x6e, 0xca, 0x41, 0x20, 0x3c	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to set EFI_FILE_SYSTEM_INFO with too small of a buffer returns EFI_BUFFER_TOO_SMALL.	1. Call SetInfo() to set EFI_FILE_SYSTEM_INFO with too small of a buffer. The return code should be EFI_BUFFER_TOO_SMALL.
5.7.3.10.24	0xc7bfe9bf, 0x92bf, 0x4301, 0x82, 0x17, 0x75, 0x66, 0x2e, 0xa5, 0x24, 0x37	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to set EFI_FILE_SYSTEM_VOLUME_LABEL with too small of a buffer returns EFI_BUFFER_TOO_SMALL.	1. Call SetInfo() to set EFI_FILE_SYSTEM_VOLUME_LABEL with too small of a buffer. The return code should be EFI_BUFFER_TOO_SMALL.
5.7.3.10.25	0x86eb2a14, 0x668a, 0x4ad6, 0xbc, 0x8a, 0x56, 0x67, 0x79, 0x09, 0x94, 0xe5	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to set illegal size, physical size and invalid attribute of EFI_FILE_INFO for the file does not change the original settings.	1. Call GetInfo() to store original size, physical size and attribute of EFI_FILE_INFO of the file. 2. Call SetInfo() to set illegal size, physical size and invalid attribute of EFI_FILE_INFO for the file. 3. Call GetInfo() again to get current size, physical size and attribute of EFI_FILE_INFO of the file. It should return the same value as original settings.
5.7.3.10.26	0x63c55abc, 0x16d6, 0x4ac9, 0xb7, 0x8c, 0x45, 0x44, 0xbe, 0x70, 0x81, 0x54	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to set illegal filename of EFI_FILE_INFO for the file does not change the filename.	1. Call Open() to create file handle with valid filename. 2. Call SetInfo() to set illegal filename of EFI_FILE_INFO for the file. 3. Call GetInfo() to get current filename of EFI_FILE_INFO of the file. It should return the same value as original filename.
5.7.3.10.27	0x7ba04c1e, 0xcd95, 0x4a3c, 0xa3, 0xba, 0xa5, 0x82, 0xf1, 0x6c, 0x46, 0xbc	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to set illegal size, physical size and invalid attribute of EFI_FILE_INFO for the directory does not change the original settings.	1. Call GetInfo() to store original size, physical size and attribute of EFI_FILE_INFO of the directory. 2. Call SetInfo() to set illegal size, physical size and invalid attribute of EFI_FILE_INFO for the directory. 3. Call GetInfo() again to get current size, physical size and attribute of EFI_FILE_INFO of the directory. It should return the same value as original settings.

Number	GUID	Assertion	Test Description
5.7.3.10.28	0x6a09725c, 0x51c7, 0x44f3, 0x85, 0x74, 0xda, 0xc3, 0x6e, 0xc7, 0x0f, 0x86	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to set illegal filename of EFI_FILE_INFO for the directory does not change the filename.	1. Call Open() to create directory handle with valid filename. 2. Call SetInfo() to set illegal filename of EFI_FILE_INFO for the directory. 3. Call GetInfo() to get current filename of EFI_FILE_INFO of the directory. It should return the same value as original filename.
5.7.3.10.29	0x5bef76ad, 0x4a40, 0x401c, 0x83, 0xd3, 0x9c, 0x73, 0x72, 0x3b, 0xb4, 0x58	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to change all the fields except the VolumeLabel of the EFI_FILE_SYSTEM_INFO does not change the original settings.	1. Call GetInfo() to store original value of fields of EFI_FILE_SYSTEM_INFO. 2. Call SetInfo() to change all the fields except the VolumeLabel of the EFI_FILE_SYSTEM_INFO. 3. Call GetInfo() again to get current value of fields of EFI_FILE_SYSTEM_INFO. It should return the same value as original settings.
5.7.3.10.30	0x4857f42c, 0xb998, 0x4667, 0x8f, 0x11, 0xdb, 0xed, 0x7a, 0xd5, 0xe0, 0xac	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to change file attribute to read-only returns EFI_SUCCESS.	1. Call Open() to create a file handle. 2. Call SetInfo() to set file attribute to read-only. The return code should be EFI_SUCCESS.
5.7.3.10.31	0xa9df1e64, 0xe769, 0x4d16, 0xa0, 0xd5, 0xb5, 0x59, 0xce, 0x90, 0xcf, 0x2b	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to change file attribute to read-only changes the file attribute from read-write to read-only.	1. Call Open() to create a file handle. 2. Call SetInfo() to set file attribute to read-only. 3. Call GetInfo() to get file attribute. It should be read-only.
5.7.3.10.32	0xb5481965, 0xf157, 0x4037, 0x89, 0xab, 0x14, 0x6e, 0xa6, 0xc9, 0x44, 0x1a	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to change file attribute from read-only to read-write returns EFI_SUCCESS.	1. Call Open() with read-only open mode to open a read-only file. 2. Call SetInfo() to set file attribute to read-write. The return code should be EFI_SUCCESS.
5.7.3.10.33	0x3535af93, 0x32df, 0x44bb, 0xa0, 0xaf, 0xce, 0x2d, 0x38, 0xe7, 0xd6, 0xfb	EFI_FILE_PROTOCOL.SetInfo – SetInfo() to change file attribute to read-write changes the file attribute from read-only to read-write.	1. Call Open() with read-only open mode to open a read-only file. 2. Call SetInfo() to set file attribute to read-write. 3. Call GetInfo() to get file attribute. It should be read-write.

Number	GUID	Assertion	Test Description
5.7.3.10.34	0x8821c678, 0xde6e, 0x49bf, 0x94, 0xcd, 0x9f, 0x4b, 0xa0, 0xa2, 0x15, 0x22	**EFI_FILE_PROTOCOL.SetInfo – SetInfo()** to change file name to an existing file name returns **EFI_ACCESS_DENIED**	1. Call **Open()** to create two file handle. 2. Call **SetInfo()** to set one file name to the other file name. The return code should be **EFI_ACCESS_DENIED.**
5.7.3.10.35	0x69afc35a, 0xcf85, 0x4365, 0xac, 0xca, 0xa5, 0x3c, 0x48, 0xcb, 0xd3, 0x51	**EFI_FILE_PROTOCOL.SetInfo – SetInfo()** to change file system volume info on a read-only media returns **EFI_WRITE_PROTECTED**	1. Get system volume info to see if it is a read-only media 2. Call **SetInfo()** to change file system volume info. The return code should be **EFI_WRITE_PROTECTED**
5.7.3.10.36	0x669bf242, 0xd3ca, 0x4b73, 0xa6, 0xdd, 0x8b, 0x2a, 0xf3, 0xfb, 0xa6, 0x28	**EFI_FILE_PROTOCOL.SetInfo – SetInfo()** to change file system volume label on a read-only media returns **EFI_WRITE_PROTECTED**	1. Get system volume info to see if it is a read-only media 2. Call **SetInfo()** to change file system volume label. The return code should be **EFI_WRITE_PROTECTED**
5.7.3.10.37	0x33218d68, 0x5245, 0x4bab, 0x9c, 0x1d, 0xc, 0x4b, 0xca, 0xd9, 0x4, 0x87	**EFI_FILE_PROTOCOL.SetInfo – SetInfo()** to change file info on a read-only media returns **EFI_WRITE_PROTECTED**	1. Get system volume info to see if it is a read-only media 2. Call **SetInfo()** to change file info of root dir. The return code should be **EFI_WRITE_PROTECTED**

9.3.11 OpenEx()

Number	**GUID**	**Assertion**	**Test Description**
5.7.3.11.1	0xb6cff935, 0x32ef, 0x4865,0x9e, 0xd9, 0x09, 0x62, 0x87, 0xf2, 0x2a, 0x66	**EFI_FILE_PROTOCOL.OpenEx – OpenEx()** to async create file under root directory with pure filename returns **EFI_SUCCESS**.	Async call **OpenEx()** to create file under root directory with pure filename, the return status should be **EFI_SUCCESS**. The status in OpenFileFinishList should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.7.3.11.2	0x5e767a63, 0x577a, 0x4628,0xb6, 0xe9, 0x91, 0xb7, 0xd9, 0xaa, 0x05, 0xcb	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file under root directory with pure filename returns EFI_SUCCESS.	Async call OpenEx() to create file under root directory with pure filename, the return status should be EFI_SUCCESS. The OpenFileFailList should be empty.
5.7.3.11.3	0x611a8daf, 0x274c, 0x4bd5,0xa7, 0xba, 0xc1, 0x85, 0x43, 0xd3, 0x7f, 0x74	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file under root directory with pure filename returns EFI_SUCCESS.	Async call OpenEx() to create file under root directory with pure filename, the return status should be EFI_SUCCESS. The OpenFileExecuteList should be empty.
5.7.3.11.4	0x0a6985e4, 0xfe17, 0x4740,0x95, 0x7a, 0xe9, 0xc0, 0x5b, 0x45, 0x02, 0xe0	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to sync create file under root directory with pure filename returns EFI_SUCCESS.	Sync call OpenEx() to create file under root directory with pure filename, the return status should be EFI_SUCCESS.
5.7.3.11.5	0x0c61f052, 0x2ae3, 0x4219,0xad, 0x79, 0x4b, 0xdd, 0x95, 0xc1, 0x78, 0xc3	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create directory under root directory returns EFI_SUCCESS.	Async call OpenEx() to create subdirectory under root directory, the return status should be EFI_SUCCESS.
5.7.3.11.6	0x41087c41, 0xb9a9, 0x4943,0xb8, 0x22, 0x9f, 0x9b, 0x41, 0x78, 0xa6, 0x49	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file under root directory with filename containing sub directory name returns EFI_SUCCESS.	Async call OpenEx() to create file under root directory with filename containing sub directory name, the return status should be EFI_SUCCESS.
5.7.3.11.7	0x6277ccac, 0x481c, 0x4cb2, 0xac, 0x96, 0x89, 0x96, 0x79, 0xf2, 0xa9, 0x19	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create directory under root directory returns EFI_SUCCESS.	Async call OpenEx() to create directory under root directory, the return status should be EFI_SUCCESS. The OpenDirFailList should be empty.

Number	GUID	Assertion	Test Description
5.7.3.11.8	0x518c70d5, 0x4070, 0x4b81,0x9d, 0xb3, 0xcb, 0x20, 0xd6, 0x7f, 0x11, 0x1c	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file under root directory with filename containing sub directory name returns EFI_SUCCESS.	Async call OpenEx() to create file under root directory with filename containing sub directory name, the return status should be EFI_SUCCESS. The OpenFileFailList should be empty.
5.7.3.11.9	0x44fa0576, 0x08cd, 0x48c2,0x9b, 0x71, 0x5f, 0x63, 0xc2, 0xb3, 0x97, 0x10	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create directory under root directory returns EFI_SUCCESS.	Async call OpenEx() to create directory under root directory, the return status should be EFI_SUCCESS. The OpenDirExecuteList should be empty.
5.7.3.11.10	0xef745935, 0x0937, 0x4b11,0xa7, 0xca, 0x65, 0xaf, 0x0b, 0xf0, 0x45, 0x44	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file under root directory with filename containing sub directory name returns EFI_SUCCESS.	Async call OpenEx() to create file under root directory with filename containing sub directory name, the return status should be EFI_SUCCESS. The OpenFileExecuteList should be empty.
5.7.3.11.11	0x3c64e927, 0x68e7, 0x4668,0xae, 0xa8, 0xc2, 0xc7, 0xdc, 0x15, 0x0c, 0x3f	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to sync create directory under root directory returns EFI_SUCCESS.	Sync call OpenEx() to create directory under root directory, the return status should be EFI_SUCCESS.
5.7.3.11.12	0x233a928b, 0x8f5d, 0x483a,0xab, 0x03, 0x2d, 0x03, 0xf1, 0xa3, 0xdc, 0x26	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to sync create file under root directory with filename containing sub directory name returns EFI_SUCCESS.	Sync call OpenEx() to create file under root directory with filename containing sub directory name, the return status should be EFI_SUCCESS.
5.7.3.11.13	0x959a9093, 0xa975, 0x42a9,0x9b, 0x83, 0x32, 0x4a, 0x79, 0xca, 0x2f, 0x1b	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file under sub directory with pure name returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Async call OpenEx() to create file with pure file name under directory(dir1) , the return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.11.14	0x43ad5688, 0xbc02, 0x4870,0xb8, 0x85, 0x02, 0x86, 0xdd, 0x54, 0xb2, 0x76	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file under sub directory with pure name returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Async call OpenEx() to create file with pure file name under directory(dir1) , the return status should be EFI_SUCCESS. The OpenFileFailList should be empty.
5.7.3.11.15	0x90908639, 0x141f, 0x4632,0x85, 0xca, 0x7d, 0x6e, 0x83, 0xe5, 0x57, 0x47	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file under sub directory with pure name returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Async call OpenEx() to create file with pure file name under directory(dir1) , the return status should be EFI_SUCCESS. The OpenFileExecuteList should be empty.
5.7.3.11.16	0x8eb7f8cc, 0x6d0d, 0x4c10, 0xbd, 0x94, 0xdc, 0x32, 0x7d, 0x2e, 0x6d, 0x3d	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to sync create file under sub directory with pure name returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Sync call OpenEx() to create file with pure file name under directory(dir1) , the return status should be EFI_SUCCESS.
5.7.3.11.17	0x3431780c, 0x56da, 0x4628,0x86, 0xa2, 0xa3, 0x08, 0xf2, 0xe9, 0x88, 0x27	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file under sub directory and filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Call Open() to create directory(dir2) under dir1. Async call OpenEx() to create file with file name containing sub directory name (dir2/pure name) under directory(dir1) , the return status should be EFI_SUCCESS.
5.7.3.11.18	0x7d9eacf0, 0x0167, 0x4ef7, 0xa7, 0xf2, 0x31, 0xb5, 0x3e, 0xc4, 0xcb, 0x8a	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file under sub directory and filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Call Open() to create directory(dir2) under dir1. Async call OpenEx() to create file with file name containing sub directory name (dir2/pure name) under directory(dir1) , the return status should be EFI_SUCCESS. The OpenFileFailList should be empty.

Number	GUID	Assertion	Test Description
5.7.3.11.19	0xf9dad61f, 0xfc35, 0x4fd6, 0x86, 0x0b, 0x7b, 0x8b, 0x2e, 0xbf, 0x89, 0x63	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file under sub directory and filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Call Open() to create directory(dir2) under dir1. Async call OpenEx() to create file with file name containing sub directory name (dir2/pure name) under directory(dir1) , the return status should be EFI_SUCCESS. The OpenFileExecuteList should be empty.
5.7.3.11.20	0xf87622cf, 0x13c6, 0x412e,0x86, 0xa6, 0x8e, 0x7f, 0xf2, 0x63, 0x0a, 0x8e	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to sync create file under sub directory and filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Call Open() to create directory(dir2) under dir1. Sync call OpenEx() to create file with file name containing sub directory name (dir2/pure name) under directory(dir1) , the return status should be EFI_SUCCESS.
5.7.3.11.21	0xbfc2a163, 0xe8d5, 0x45df, 0x8f, 0x6b, 0x01, 0x0a, 0xee, 0x48, 0x89, 0xc0	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file with sub directory handle and filename containing absolute file path returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Call Open() to create directory(dir2) under dir1. Async call OpenEx() to create file containing absolute file path (/dir1/dir2/pure name) under sub directory(dir2), the return status should be EFI_SUCCESS.
5.7.3.11.22	0x8ffd05e8, 0xaa76, 0x4fcb, 0x93, 0xe4, 0x19, 0xa2, 0x0c, 0x2b, 0xa9, 0x04	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file with sub directory handle and filename containing absolute file path returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Call Open() to create directory(dir2) under dir1. Async call OpenEx() to create file containing absolute file path (/dir1/dir2/pure name) under sub directory(dir2), the return status should be EFI_SUCCESS. The OpenFileFailList should be empty.

Number	GUID	Assertion	Test Description
5.7.3.11.23	0x41fe9684, 0x113b, 0x415f, 0xaf, 0xbf, 0xee, 0x48, 0x10, 0x8a, 0x70, 0xc2	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async create file with sub directory handle and filename containing absolute file path returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Call Open() to create directory(dir2) under dir1. Async call OpenEx() to create file containing absolute file path (/dir1/dir2/ pure name) under sub directory(dir2), the return status should be EFI_SUCCESS. The OpenFileExecuteList should be empty.
5.7.3.11.24	0xd5c326a3, 0x07ad, 0x490e,0x9b, 0xdc, 0xa8, 0xe3, 0x4d, 0x7a, 0xae, 0x8a	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to sync create file with sub directory handle and filename containing absolute file path returns EFI_SUCCESS.	Call Open() to create directory(dir1) under root. Call Open() to create directory(dir2) under dir1. Sync call OpenEx() to create file containing absolute file path (/dir1/dir2/ pure name) under sub directory(dir2), the return status should be EFI_SUCCESS.
5.7.3.11.25	0x55825138, 0x793d, 0x4aaa, 0xab, 0xcc, 0x4d, 0x4a, 0xbd, 0xb2, 0x17, 0xef	EFI_FILE_PROTOCOL.OpenEx – OpenEx() to async open the existing file under root directory with pure filename returns EFI_SUCCESS.	Call Open() to create file under root. Call SetInfo() to set file size to 1. Async call OpenEx() to open the file Again, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.11.26	0xa4a53615, 0x7939, 0x4dcf, 0xbf, 0xb6, 0xc7, 0x4e, 0xe3, 0x3e, 0x93, 0x30	**EFI_FILE_PROTOCOL.OpenEx - OpenEx()** to async open the existing file under root directory with pure filename returns **EFI_SUCCESS**.	Call **Open()** to create file under root. Call **SetInfo()** to set file size to 1. Async call **OpenEx()** to open the file Again, the return status should be **EFI_SUCCESS**. To get the file size, it should be equal to 1. Call **SetInfo()** & **Write()**, if the Open Mode is read-only, the return status should be **EFI_ACCESS_DENIED**. Otherwise, it should be **EFI_SUCCESS**.
5.7.3.11.27	0x4014c563, 0x7c95, 0x4323,0xa2, 0xd1, 0xbb, 0x94, 0x26, 0x74, 0xc9, 0xa3	**EFI_FILE_PROTOCOL.OpenEx - OpenEx()** to async open the existing file under root directory with pure filename returns **EFI_SUCCESS**.	Call **Open()** to create file under root. Call **SetInfo()** to set file size to 1. Async call **OpenEx()** to open the file Again, the return status should be **EFI_SUCCESS**. To get the file size, it should be equal to 1. Call **SetInfo()** & **Write()**, if the Open Mode is read-only, the return status should be **EFI_ACCESS_DENIED**. Otherwise, it should be **EFI_SUCCESS**.
5.7.3.11.28	0x5a646037, 0xbe58, 0x41d8,0xb4, 0x91, 0x84, 0x03, 0xb9, 0xf8, 0xa7, 0x44	**EFI_FILE_PROTOCOL.OpenEx - OpenEx()** to sync open the existing file under root directory with pure filename returns **EFI_SUCCESS**.	Call **Open()** to create file under root. Call **SetInfo()** to set file size to 1. Sync call **OpenEx()** to open the existing file again, the return status should be **EFI_SUCCESS.** To get the file size, it should be equal to 1. Call **SetInfo()** & **Write()**, if the Open Mode is read-only, the return status should be **EFI_ACCESS_DENIED**. Otherwise, it should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.7.3.11.29	0xa398b24a , 0x568f, 0x4762,0xb 1, 0xcb, 0x52, 0x25, 0xa7, 0x0e, 0x2f, 0x1f	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file under root directory with filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create file under dir1. Call SetInfo() to set file size to 1. Async call OpenEx() to open the existing file under root directory with filename containing sub directory name, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.
5.7.3.11.30	0xbab0c3fc, 0x8630, 0x43bf, 0x97, 0x88, 0x6d, 0x96, 0xcd, 0x3a, 0x6e, 0x7c	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file under root directory with filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create file under dir1. Call SetInfo() to set file size to 1. Async call OpenEx() to open the existing file under root directory with filename containing sub directory name, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.11.31	0xffc5787b, 0x29a5, 0x4704,0x84, 0xd9, 0xd8, 0xb6, 0x6e, 0x62, 0x9c, 0xc2	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file under root directory with filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create file under dir1. Call SetInfo() to set file size to 1. Async call OpenEx() to open the existing file under root directory with filename containing sub directory name, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.
5.7.3.11.32	0xe9d202ed, 0x2e34, 0x4686,0x9a, 0xe3, 0x9b, 0x41, 0x5b, 0xaa, 0xbc, 0x72	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to sync open the existing file under root directory with filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create file under dir1. Call SetInfo() to set file size to 1. Sync call OpenEx() to open the existing file under root directory with filename containing sub directory name, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.
5.7.3.11.33	0x3c57480f, 0xc2f3, 0x4cee, 0xab, 0xef, 0x54, 0x8d, 0x69, 0x56, 0xae, 0x89	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file under sub directory with pure filename returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create file under dir1. Call SetInfo() to set file size to 1. Async call OpenEx() to open the existing file under sub directory with pure filename, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.11.34	0x5850bc3c, 0x1b0f, 0x4bda,0x9e, 0x3c, 0x9c, 0x17, 0xf1, 0x9d, 0xf7, 0x53	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file under sub directory with pure filename returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create file under dir1. Call SetInfo() to set file size to 1. Async call OpenEx() to open the existing file under sub directory with pure filename, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.
5.7.3.11.35	0x13ce6d88, 0xd770, 0x470f, 0xb7, 0x3d, 0x60, 0x25, 0x18, 0xc2, 0xd2, 0xbf	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file under sub directory with pure filename returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create file under dir1. Call SetInfo() to set file size to 1. Async call OpenEx() to open the existing file under sub directory with pure filename, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.
5.7.3.11.36	0xc2535525, 0xbe07, 0x4980,0xb9, 0x46, 0x7f, 0x87, 0x09, 0xe2, 0x12, 0xbe	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to sync open the existing file under sub directory with pure filename returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create file under dir1. Call SetInfo() to set file size to 1. Sync call OpenEx() to open the existing file under sub directory with pure filename, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.11.37	0x7b0dcc35, 0xc3ea, 0x43cc, 0xac, 0xa9, 0x6a, 0x60, 0x1c, 0x3d, 0xe5, 0x45	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create sub directory(dir2)under dir1. Call Open() to create file under dir2. Call SetInfo() to set file size to 1. Async call OpenEx() to async open the existing file with sub directory(dir1) handle and filename(/dir2/filename) containing sub directory name, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.
5.7.3.11.38	0x8a6ef609, 0xe8dc, 0x40a2,0xb4, 0x18, 0xd0, 0xa4, 0xdf, 0x4d, 0x3f, 0xa3	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create sub directory(dir2)under dir1. Call Open() to create file under dir2. Call SetInfo() to set file size to 1. Async call OpenEx() to async open the existing file with sub directory(dir1) handle and filename(/dir2/filename) containing sub directory name, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.11.39	0x5cfc5d39, 0x197c, 0x48dd, 0x9c, 0x7f, 0x98, 0x51, 0x64, 0x96, 0x79, 0xe7	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create sub directory(dir2)under dir1. Call Open() to create file under dir2. Call SetInfo() to set file size to 1. Async call OpenEx() to async open the existing file with sub directory(dir1) handle and filename(/dir2/filename) containing sub directory name, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.
5.7.3.11.40	0x1f3f5ccf, 0xdc02, 0x4200,0x81, 0xd0, 0x02, 0x34, 0x32, 0x60, 0xf2, 0xe5	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to sync open the existing file with sub directory handle and filename containing sub directory name returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create sub directory(dir2)under dir1. Call Open() to create file under dir2. Call SetInfo() to set file size to 1. Sync call OpenEx() to sync open the existing file with sub directory(dir1) handle and filename(/dir2/filename) containing sub directory name, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.11.41	0x83351bef, 0x2368, 0x442e,0x89, 0xe6, 0xd2, 0xd5, 0xe9, 0xaf, 0x4a, 0x40	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing absolute file path returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create sub directory(dir2) under dir1. Call Open() to create file under dir2. Call SetInfo() to set file size to 1. Async call OpenEx() to async open the existing file with sub directory handle and filename containing absolute file path, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.
5.7.3.11.42	0x1e8c1e14, 0x47d8, 0x4a23,0xb2, 0xd6, 0x4b, 0xe0, 0x99, 0xf4, 0xa5, 0xdf	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing absolute file path returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create sub directory(dir2) under dir1. Call Open() to create file under dir2. Call SetInfo() to set file size to 1. Async call OpenEx() to async open the existing file with sub directory handle and filename containing absolute file path, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.3.11.43	0x70486db6, 0x12f9, 0x4f6e, 0xa3, 0xf2, 0xed, 0xb4, 0x21, 0x27, 0x45, 0xbc	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing absolute file path returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create sub directory(dir2) under dir1. Call Open() to create file under dir2. Call SetInfo() to set file size to 1. Async call OpenEx() to async open the existing file with sub directory handle and filename containing absolute file path, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.
5.7.3.11.44	0x69996cd2, 0xf087, 0x42e9,0xb7, 0xf6, 0x7c, 0x04, 0x18, 0x76, 0x36, 0xd7	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to sync open the existing file with sub directory handle and filename containing absolute file path returns EFI_SUCCESS.	Call Open() to create directory(dir1). Call Open() to create sub directory(dir2) under dir1. Call Open() to create file under dir2. Call SetInfo() to set file size to 1. Sync call OpenEx() to sync open the existing file with sub directory handle and filename containing absolute file path, the return status should be EFI_SUCCESS. To get the file size, it should be equal to 1. Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI_ACCESS_DENIED. Otherwise, it should be EFI_SUCCESS.
5.7.3.11.45	0xad02d93d, 0xf2e8, 0x4f25, 0x93, 0xce, 0x94, 0x06, 0x77, 0xb6, 0xe1, 0xb2	EFI_FILE_PROTOCOL.OpenEx - OpenEx() to async & sync open with non-existent file name returns EFI_NOT_FOUND.	Async & Sync call OpenEx() to open with non-existent file name, the return status should be EFI_NOT_FOUND.

Number	GUID	Assertion	Test Description
5.7.3.11.46	0xcab7c260, 0xa290, 0x4845,0xb7, 0x03, 0xb1, 0x9f, 0xed, 0xf9, 0x84, 0xeb	**EFI_FILE_PROTOCOL.OpenEx** – **OpenEx()** to async & sync open with non-existent file path returns **EFI_NOT_FOUND**.	Async & Sync call **OpenEx()** to open with non-existent file path, the return status should be **EFI_NOT_FOUND**.
5.7.3.11.47	0x33273ae, 0x2471, 0x4c08,0xb0, 0x8d, 0xeb, 0xd9, 0xdd, 0xbd, 0x57, 0x81	**EFI_FILE_PROTOCOL.OpenEx** – **OpenEx()** to async & sync open with invalid open-mode returns **EFI_INVALID_PARAMETER**.	Async & Sync call **OpenEx()** to open with invalid open-mode, the return status should be **EFI_INVALID_PARAMETER**.

9.3.12 ReadEx

Number	GUID	Assertion	Test Description
5.7.3.12.1	0xce038e00, 0x833c, 0x4b2e, 0x9e, 0x50, 0x79, 0xed, 0xc, 0x74, 0xf2, 0x50	**EFI_FILE_PROTOCOL.ReadEx** – **ReadEx()** to async read data from a file returns **EFI_SUCCESS.**	Call **Open()** to create a file. Call **Write()** to write data to the file. Async Call **ReadEx()** from valid setposition & ReadLength, the return status should be **EFI_SUCCESS** and ReadLength should be equal to the Token's BufferSize. Call **GetPosition()** the PositionAfterRead should be equal to the sum of SetPosition and ReadLength. Compare the content of ReadBuffer with the data set in step2.
5.7.3.12.2	0x05857ebf, 0xc920, 0x474a, 0x97, 0x4d, 0x85, 0x8d, 0x83, 0x98, 0x81, 0x6f	**EFI_FILE_PROTOCOL.ReadEx** – **ReadEx()** to async read data from a file returns **EFI_SUCCESS**.	Call **Open()** to create a file. Call **Write()** to write data to the file. Async Call **ReadEx()** from valid setposition & ReadLength, the return status should be **EFI_SUCCESS** and ReadLength should be equal to the Token's BufferSize. Call **GetPosition()** the PositionAfterRead should be equal to the sum of SetPosition and ReadLength. Compare the content of ReadBuffer with the data set in step2. The ReadFailList should be empty.

5.7.3.12.3	0x858ccc86, 0x9739, 0x437e, 0x82, 0xff, 0x29, 0x8a, 0x34, 0x7f, 0xc4, 0x45	**EFI_FILE_PROTOCOL.ReadEx - ReadEx()** to async read data from a file returns **EFI_SUCCESS**.	Call **Open()** to create a file. Call **Write()** to write data to the file. Async Call **ReadEx()** from valid setposition & ReadLength, the return status should be **EFI_SUCCESS** and ReadLength should be equal to the Token's BufferSize. Call **GetPosition()** the PositionAfterRead should be equal to the sum of SetPosition and ReadLength. Compare the content of ReadBuffer with the data set in step2. The ReadExecuteList should be empty.
5.7.3.12.4	0xccb9106f, 0x79ee, 0x4ec1, 0x98, 0xa5, 0x16, 0x8d, 0xe1, 0xa6, 0xb8, 0xf3	**EFI_FILE_PROTOCOL.ReadEx - ReadEx()** to sync read data from a file returns **EFI_SUCCESS**.	Call **Open()** to create a file. Call **Write()** to write data to the file. Sync Call **ReadEx()** from valid setposition & ReadLength, the return status should be **EFI_SUCCESS** and ReadLength should be equal to the Token's BufferSize. Call **GetPosition()** the PositionAfterRead should be equal to the sum of SetPosition and ReadLength. Compare the content of ReadBuffer with the data set in step2.
5.7.3.12.5	0xd01cdf69, 0x1b1b, 0x42fc, 0x92, 0x3f, 0x1d, 0xc1, 0x90, 0x92, 0x03, 0xc7	**EFI_FILE_PROTOCOL.ReadEx - ReadEx()** to async read data from a directory.	Call **Open()** to create a directory. Call **Open()** to create a file under the directory opened in step1. Async Call **ReadEx()** from different setposition & ReadLength, if the Setposition is 0 and the ReadLength is smaller than **SIZE_OF_EFI_FILE_INFO** + 4, the return status should be **EFI_BUFFER_TOO_SMALL**, else if the Setposition is 0 and the ReadLength is not less than **SIZE_OF_EFI_FILE_INFO** + 4, the return status should be **EFI_SUCCESS**, if the Setposition is at the end of directory, the return status should be **EFI_SUCCESS**.

5.7.3.12.6	0x05241cbf, 0xf260, 0x41d7, 0xb1, 0x93, 0x3b, 0x27, 0x7f, 0x72, 0x12, 0x4c	**EFI_FILE_PROTOCOL.ReadEx** - **ReadEx()** to async read data from a directory.	Call **Open()** to create a directory. Call **Open()** to create a file under the directory opened in step1. Async Call **ReadEx()** from different setposition & ReadLength, if the Setposition is 0 and the ReadLength is smaller than **SIZE_OF_EFI_FILE_INFO** + 4, the return status should be **EFI_BUFFER_TOO_SMALL**, else if the Setposition is 0 and the ReadLength is not less than **SIZE_OF_EFI_FILE_INFO** + 4, the return status should be **EFI_SUCCESS**, if the Setposition is at the end of directory, the return status should be **EFI_SUCCESS**.
5.7.3.12.7	0xcfbb86c0, 0xc6c6, 0x40ca, 0x8e, 0xc8, 0x0d, 0x76, 0xd0, 0xef, 0x50, 0xe7	**EFI_FILE_PROTOCOL.ReadEx** - **ReadEx()** to async read data from a directory.	Call **Open()** to create a directory. Call **Open()** to create a file under the directory opened in step1. Async Call **ReadEx()** from different setposition & ReadLength, if the Setposition is 0 and the ReadLength is smaller than **SIZE_OF_EFI_FILE_INFO** + 4, the return status should be **EFI_BUFFER_TOO_SMALL**, else if the Setposition is 0 and the ReadLength is not less than **SIZE_OF_EFI_FILE_INFO** + 4, the return status should be **EFI_SUCCESS**, if the Setposition is at the end of directory, the return status should be **EFI_SUCCESS**. The ReadExecuteList should be empty.

5.7.3.12.8	0xe8e8665c, 0xa44f, 0x491b, 0xb7, 0xe0, 0x56, 0x09, 0xc2, 0xbc, 0x20, 0xee	**EFI_FILE_PROTOCOL.ReadEx - ReadEx()** to sync read data from a directory.	Call **Open()** to create a directory. Call **Open()** to create a file under the directory opened in step1. Sync Call **ReadEx()** from different setposition & ReadLength, if the Setposition is 0 and the ReadLength is small than **SIZE_OF_EFI_FILE_INFO** + 4, the return status should be **EFI_BUFFER_TOO_SMALL**, else if the Setposition is 0 and the ReadLength is not less than **SIZE_OF_EFI_FILE_INFO** + 4, the return status should be **EFI_SUCCESS**, if the Setposition is at the end of directory, the return status should be **EFI_SUCCESS.**
5.7.3.12.9	0x864c9887, 0x7205, 0x4e15,0xad, 0x9f, 0x7a, 0x94, 0xec, 0xf0, 0xc2, 0xd8	**EFI_FILE_PROTOCOL.ReadEx - ReadEx()** async & sync read data from a file with the fileposition beyond the end of the file returns **EFI_DEVICE_ERROR**.	Async & Sync Call **ReadEx()** read data from a file with the fileposition beyond the end of the file, the return status should be **EFI_DEVICE_ERROR.**
5.7.3.12.10	0x12bc7ab7, 0x4ac5, 0x4cf3, 0xa5, 0x54, 0x6b, 0x34, 0xc9, 0x5d, 0x0c, 0xea	**EFI_FILE_PROTOCOL.ReadEx - ReadEx()** async & sync read data from a file which has been deleted returns **EFI_DEVICE_ERROR**.	Async & Sync Call **ReadEx()** read data from a file which has been deleted, the return status should be **EFI_DEVICE_ERROR**.

9.3.13 WriteEX

Number	GUID	Assertion	Test Description
5.7.3.13.1	0x077c1f80, 0xa887, 0x417d, 0xa9, 0xd6, 0xd9, 0x54, 0xca, 0x0b, 0x94, 0x7b	**EFI_FILE_PROTOCOL.WriteEx - WriteEx()** to async write data into a normal file returns **EFI_SUCCESS**.	Call **Open()** to create a file. Async Call **WriteEx()** from valid setposition & WriteLength, the return status should be **EFI_SUCCESS** and FileHandle's position after write should be equal to the sum of Setposition and WriteLength. Call **Read()**,then compare the content of ReadBuffer with the data writed to the file in step2, they should be the same.

Number	GUID	Assertion	Test Description
5.7.3.13.2	0xf75bdc5a, 0xfd02, 0x444d, 0x9b, 0xb1, 0xda, 0x70, 0x2e, 0x2a, 0x86, 0x13	**EFI_FILE_PROTOCOL.WriteEx – WriteEx()** to async write data into a normal file returns **EFI_SUCCESS**.	Call **Open()** to create a file. Async Call **WriteEx()** from valid setposition & WriteLength, the return status should be **EFI_SUCCESS** and FileHandle's position after write should be equal to the sum of Setposition and WriteLength. Call **Read()**,then compare the content of ReadBuffer with the data writed to the file in step2, they should be the same. The WriteFailList should be empty.
5.7.3.13.3	0xc105380e, 0x4c6d, 0x4e49, 0x8d, 0xe8, 0x1a, 0x0c, 0xc0, 0x77, 0x3e, 0xdc	**EFI_FILE_PROTOCOL.WriteEx – WriteEx()** to async write data into a normal file returns **EFI_SUCCESS**.	Call **Open()** to create a file. Async Call **WriteEx()** from valid setposition & WriteLength, the return status should be **EFI_SUCCESS** and FileHandle's position after write should be equal to the sum of Setposition and WriteLength. Call **Read()**,then compare the content of ReadBuffer with the data writed to the file in step2, they should be the same. The WriteExecuteList should be empty.
5.7.3.13.4	0x67e49003, 0xf68c, 0x44bd, 0xb6, 0xee, 0xa5, 0xc8, 0x01, 0x06, 0xe7, 0xc1	**EFI_FILE_PROTOCOL.WriteEx – WriteEx()** to async write data into a normal file returns **EFI_SUCCESS**.	Call **Open()** to create a file. Sync Call **WriteEx()** from valid setposition & WriteLength, the return status should be **EFI_SUCCESS** and FileHandle's position after write should be equal to the sum of Setposition and WriteLength. Call **Read()**,then compare the content of ReadBuffer with the data writed to the file in step2, they should be the same.
5.7.3.13.5	0xbe6ccb33, 0x351f, 0x488c, 0x86, 0x42, 0x65, 0x47, 0xa1, 0x35, 0x79, 0x0c	**EFI_FILE_PROTOCOL.WriteEx – WriteEx()** to async write data into multi files returns **EFI_SUCCESS**.	Call **Open()** to create three file. Async Call **WriteEx()** to write data to different file with different position and write length, the return status should be **EFI_SUCCESS**. Compare the position after write, the writelength with the expect value. Call **Read()**,then compare the content of ReadBuffer with the data writed to the file in step2, they should be the same.

Number	GUID	Assertion	Test Description
5.7.3.13.6	0x0aaacd7f, 0xeb8b, 0x4e91, 0x9b, 0xcd, 0x30, 0x57, 0xe6, 0x10, 0x57, 0x69	**EFI_FILE_PROTOCOL.WriteEx - WriteEx()** to async write data into multi files returns **EFI_SUCCESS**.	Call **Open()** to create three file. Async Call **WriteEx()** to write data to different file with different position and write length, the return status should be **EFI_SUCCESS**. Compare the position after write, the writelength with the expect value. Call **Read()**, then compare the content of ReadBuffer with the data writed to the file in step2, they should be the same. The WriteMultiFailList should be empty.
5.7.3.13.7	0x4c7ec69e, 0x9615, 0x4274, 0xa0, 0x99, 0xb1, 0xd3, 0x48, 0x88, 0xd6, 0x70	**EFI_FILE_PROTOCOL.WriteEx - WriteEx()** to async write data into multi files returns **EFI_SUCCESS**.	Call **Open()** to create three file. Async Call **WriteEx()** to write data to different file with different position and write length, the return status should be **EFI_SUCCESS**. Compare the position after write, the writelength with the expect value. Call **Read()**,then compare the content of ReadBuffer with the data writed to the file in step2, they should be the same. The WriteMultiExecuteList should be empty.
5.7.3.13.8	0x03186ac5, 0xb4b2, 0x4d2d, 0xa8, 0x67, 0xb9, 0x10, 0xdd, 0x1f, 0x64, 0xad	**EFI_FILE_PROTOCOL.WriteEx - WriteEx()** to async write data into multi files returns **EFI_SUCCESS**.	Call **Open()** to create three file. Sync Call **WriteEx()** to write data to different file with different position and write length, the return status should be **EFI_SUCCESS**. Compare the position after write, the writelength with the expect value. Call **Read()**,then compare the content of ReadBuffer with the data writed to the file in step2, they should be the same.
5.7.3.13.9	0xc51c0c6d, 0xdfc6, 0x4ea7,0xb4, 0x36, 0x83, 0xae, 0x3a, 0x3f, 0x49, 0xd2	**EFI_FILE_PROTOCOL.WriteEx - WriteEx()** to async & sync write data to a directory returns **EFI_UNSUPPORTED**.	Call **WriteEx()** to async & sync write data to a directory, the return status should be **EFI_DEVICE_ERROR**.

Number	GUID	Assertion	Test Description
5.7.3.13.10	0xc9af9973, 0x76af, 0x4701,0x88, 0xc0, 0xff, 0x61, 0x0e, 0x37, 0x74, 0x0a	**EFI_FILE_PROTOCOL.WriteEx - WriteEx()** to async & sync write data to a file which was opened read-only returns **EFI_ACCESS_DENIED**.	Async & sync Call **WriteEx()** to write data to a file which was opened read-only, the return status should be **EFI_ACCESS_DENIED**.
5.7.3.13.11	0xa056bcff, 0xdb0b, 0x4733,0x88, 0x9a, 0xb1, 0xca, 0x52, 0xac, 0x58, 0xe9	**EFI_FILE_PROTOCOL.WriteEx - WriteEx()** to async & sync write data to a file which has been deleted returns **EFI_DEVICE_ERROR**.	Async & sync Call **WriteEx()** to write data to a file which has been deleted, the return status should be **EFI_DEVICE_ERROR**.

9.3.14 FlushEx

Number	GUID	Assertion	Test Description
5.7.3.14.1	0x31473e47, 0xa40d, 0x43a0, 0xb7, 0xb8, 0x91, 0xd3, 0x29, 0x41, 0x75, 0x9d	**EFI_FILE_PROTOCOL.FlushEx - FlushEx ()** to async flush data into a normal file returns **EFI_SUCCESS**.	Call **Open()** to create a file. Call **Write()** to write data to the file. Async Call **FlushEx()**, the return status should be **EFI_SUCCESS**.
5.7.3.14.2	0x55702a2c, 0x0eef, 0x4ded, 0xa6, 0xd9, 0x2f, 0xd7, 0x9a, 0xbb, 0x88, 0x5f	**EFI_FILE_PROTOCOL.FlushEx - FlushEx ()** to async flush data into a normal file returns EFI_SUCCESS.	Call **Open()** to create a file. Call **Write()** to write data to the file. Async Call **FlushEx()**, the return status should be **EFI_SUCCESS**. The flushFileFailList should be empty.
5.7.3.14.3	0x258a6597, 0xd2ef, 0x4711, 0xa9, 0x89, 0xaa, 0xf0, 0xf9, 0x6f, 0x01, 0x0c	**EFI_FILE_PROTOCOL.FlushEx - FlushEx ()** to async flush data into a normal file returns **EFI_SUCCESS**.	Call **Open()** to create a file. Call **Write()** to write data to the file. Async Call **FlushEx()**, the return status should be **EFI_SUCCESS**. The FlushFileExecuteList should be empty.
5.7.3.14.4	0xafd40ec9, 0x5027, 0x42a8, 0xb0, 0x2c, 0x0c, 0xb5, 0x80, 0x86, 0xd7, 0x9c	**EFI_FILE_PROTOCOL.FlushEx - FlushEx ()** to sync flush data into a normal file returns **EFI_SUCCESS**.	Call **Open()** to create a file. Call **Write()** to write data to the file. Sync Call **FlushEx()**, the return status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.7.3.14.5	0x6aa8b399, 0x1b2f, 0x48d7, 0xa5, 0x34, 0x56, 0xc9, 0x68, 0xd6, 0xae, 0x11	EFI_FILE_PROTOCOL.FlushEx - FlushEx () to async flush data into a normal directory returns EFI_SUCCESS.	Call Open() to create a directory. Call Open() to Create files under the directory. Async Call FlushEx(), the return status should be EFI_SUCCESS.
5.7.3.14.6	0xac3897ad, 0xd9c1, 0x4442, 0x84, 0x4b, 0x5c, 0xa1, 0x5c, 0x32, 0x80, 0x0b	EFI_FILE_PROTOCOL.FlushEx - FlushEx () to async flush data into a normal directory returns EFI_SUCCESS.	Call Open() to create a directory. Call Open() to Create files under the directory. Async Call FlushEx(), the return status should be EFI_SUCCESS. The FlushDirFailList should be empty.
5.7.3.14.7	0x3b9ed07d, 0xa0ea, 0x4719, 0xa2, 0xc9, 0xad, 0x54, 0x57, 0xc1, 0x5a, 0x73	EFI_FILE_PROTOCOL.FlushEx - FlushEx () to async flush data into a normal directory returns EFI_SUCCESS.	Call Open() to create a directory. Call Open() to Create files under the directory. Async Call FlushEx(), the return status should be EFI_SUCCESS. The FlushDirExecuteList should be empty.
5.7.3.14.8	0x93ebe8a5, 0xf66b, 0x4532, 0x95, 0x77, 0x51, 0xe9, 0xdc, 0xda, 0xb6, 0x81	EFI_FILE_PROTOCOL.FlushEx - FlushEx () to sync flush data into a normal directory returns EFI_SUCCESS.	Call Open() to create a directory. Call Open() to Create files under the directory. Sync Call FlushEx(), the return status should be EFI_SUCCESS.
5.7.3.14.9	0xce7774fa, 0xd04c, 0x45a6, 0xb7, 0x0b, 0xcd, 0x91, 0xa2, 0x76, 0xf9, 0x15	EFI_FILE_PROTOCOL.FlushEx - FlushEx () to async & sync flush data to a file whose open mode was read-only returns EFI_ACCESS_DENIED.	Call Open() to create a directory. Call Open() to open the file in the mode of Read-Only. Async & Sync Call FlushEx(), the return status should be EFI_ACCESS_DENIED.

9.3.15 Read-Only File System check points

Number	GUID	Assertion	Test Description
5.7.3.15.1	0xad3516c1, 0xbf24, 0x4923, 0xb8, 0x84, 0x53, 0x8b, 0x04, 0x2f, 0xb8, 0x25	EFI_FILE_PROTOCOL .GetInfo - GetInfo () get the consistent ReadOnly attribute from EFI_FILE_INFO and EFI_FILE_SYSTEM_INFO.	Call GetInfo() to check the ReadOnly attribute from EFI_FILE_INFO and EFI_FILE_SYSTEM_INFO. The value should be consistent.
5.7.3.15.2	0x5b704b82, 0xe081, 0x4c4a, 0x9d, 0x65, 0x71, 0x00, 0x79, 0xd1, 0x1f, 0x64	EFI_FILE_PROTOCOL .SetPosition - SetPosition () return EFI_UNSUPPORTED when the position is not 0 and file handle is the root directory on a volume.	Call SetPosition() when the position is not 0 and file handle is the root directory on a volume, the return status should be EFI_UNSUPPORTED.
5.7.3.15.3	0xd9cbe15a, 0x956a, 0x4e54, 0xa3, 0x50, 0xdf, 0x53, 0xdc, 0x7d, 0xe2, 0x5b	EFI_FILE_PROTOCOL .SetPosition - SetPosition () return EFI_SUCCESS when the position is 0 and file handle is the root directory on a volume.	Call SetPosition () return EFI_SUCCESS when the position is 0 and file handle is the root directory on a volume.
5.7.3.15.4	0xa8aadad0, 0x8545, 0x4098, 0x8a, 0x34, 0x2a, 0x03, 0xc2, 0x2b, 0xc0, 0xf6	EFI_FILE_PROTOCOL .GetPosition - GetPosition () return EFI_UNSUPPORTED when the file handle is the root directory on a volume.	Call GetPosition () return EFI_UNSUPPORTED when the file handle is the root directory on a volume.
5.7.3.15.5	0xb20660fc, 0xb957, 0x49d7, 0x8d, 0x93, 0x5c, 0x3f, 0x73, 0x6e, 0xd5, 0xf5	EFI_FILE_PROTOCOL .SetInfo - SetInfo () return EFI_WRITE_PROTECTED when the InformationType is EFI_FILE_SYSTEM_VOLUME_LABEL_ID or EFI_FILE_PROTOCOL_SYSTEM_INFO_ID and the media is read-only.	Call SetInfo() return EFI_WRITE_PROTECTED when the InformationType is EFI_FILE_SYSTEM_VOLUME_LABEL_ID or EFI_FILE_PROTOCOL_SYSTEM_INFO_ID and the media is read-only.

Number	GUID	Assertion	Test Description
5.7.3.15.6	0x04d6b761, 0xdeac, 0x4801, 0xb7, 0x39, 0xdb, 0x81, 0x8f, 0x46, 0xcf, 0x11	EFI_FILE_PROTOCOL.Write - Write () return EFI_UNSUPPORTED when the file handle is one directory.	Call Write() return EFI_UNSUPPORTED when the file handle is one directory.
5.7.3.15.7	0xbe5cddad, 0x2d54, 0x463e, 0xaf, 0xde, 0x68, 0x1c, 0xb9, 0x08, 0xa8, 0xa0	EFI_FILE_PROTOCOL.Read - Read () return EFI_BUFFER_TOO_SMALL when the file handle is one directory and buffer is not large enough to hold the directory entry.	Call Read() return EFI_BUFFER_TOO_SMALL when the file handle is one directory and buffer is not large enough to hold the directory entry.
5.7.3.15.8	0x06950775, 0xa32a, 0x421e, 0x8f, 0xce, 0xd8, 0xb4, 0xc1, 0x43, 0x17, 0xd1	EFI_FILE_PROTOCOL.Open - Open () return EFI_WRITE_PROTECTED when try to open the file with EFI_FILE_MODE_READ|EFI_FILE_MODE_WRITE or EFI_FILE_MODE_READ|EFI_FILE_MODE_WRITE|EFI_FILE_MODE_CREATE attribute while the file is on the read-only media.	Call Open() return EFI_WRITE_PROTECTED when try to open the file with EFI_FILE_MODE_READ|EFI_FILE_MODE_WRITE or EFI_FILE_MODE_READ|EFI_FILE_MODE_WRITE|EFI_FILE_MODE_CREATE attribute while the file is on the read-only media..
5.7.3.15.9	0xd529dfd8, 0x23cb, 0x4548, 0xa2, 0x81, 0x6f, 0x59, 0x1f, 0x9c, 0x54, 0x8d	EFI_FILE_PROTOCOL.Open - Open () return EFI_NOT_FOUND when try to open one no-existed file.	Call Open() return EFI_NOT_FOUND when try to open one no-existed file.
5.7.3.15.10	0xb0091f09, 0x6121, 0x40e8, 0x93, 0x1d, 0xea, 0x6b, 0xa4, 0x6b, 0xbb, 0x09	EFI_FILE_PROTOCOL.Open - Open () return EFI_SUCCESS when try to open one existed file with EFI_FILE_MODE_READ attribute.	Call Open() return EFI_SUCCESS when try to open one existed file with EFI_FILE_MODE_READ attribute.

Number	GUID	Assertion	Test Description
5.7.3.15.11	0xa42a8e9c, 0x4a31, 0x4b0a, 0xab, 0x2e, 0x7f, 0xd4, 0x2d, 0x42, 0x45, 0xf1	EFI_FILE_PROTOCOL.GetInfo - GetInfo () return EFI_UNSUPPORTED when the InformationType is not defined in the UEFI Specification.	Call GetInfo() return EFI_UNSUPPORTED when the InformationType is not defined in the UEFI Specification.
5.7.3.15.12	0x07dc8d79, 0x8349, 0x4e9e, 0x9b, 0xa4, 0x72, 0x68, 0x92, 0x0d, 0x2e, 0x35	EFI_FILE_PROTOCOL.GetInfo - GetInfo () return EFI_BUFFER_TOO_SMALL when the Buffer is not large enough to hold the EFI_FILE_INFO.	Call GetInfo() return EFI_BUFFER_TOO_SMALL when the Buffer is not large enough to hold the EFI_FILE_INFO.
5.7.3.15.13	0x54afc2f4, 0x26bd, 0x4161, 0x90, 0x5e, 0xd9, 0x24, 0xd1, 0x34, 0x24, 0x27	EFI_FILE_PROTOCOL.GetInfo - GetInfo () return EFI_SUCCESS with the correct parameters.	Call GetInfo() return EFI_SUCCESS with the correct parameters.
5.7.3.15.14	0xabaea718, 0xe1f9, 0x4edc, 0x98, 0xb2, 0x47, 0x18, 0xe4, 0xf7, 0x6b, 0x70	EFI_FILE_PROTOCOL.SetInfo - SetInfo () return EFI_WRITE_PROTECTED to retrieve the EFI_FILE_INFO or EFI_UNSUPPORTED when InformationType is not defined in the UEFI Specification.	Call SetInfo() return EFI_WRITE_PROTECTED to retrieve the EFI_FILE_INFO or EFI_UNSUPPORTED when InformationType is not defined in the UEFI Specification.
5.7.3.15.15	0x68a6c62b, 0xc1e0, 0x44d0, 0xba, 0xdb, 0x08, 0x85, 0x63, 0x37, 0x3f, 0xd7	EFI_FILE_PROTOCOL. GetPosition - GetPosition () return EFI_SUCCESS and one reasonable Position.	Call GetPosition() return EFI_SUCCESS and one reasonable Position.

Number	GUID	Assertion	Test Description
5.7.3.15.16	0x2f83c19f, 0xc757, 0x4975, 0xa5, 0xea, 0x6a, 0x4e, 0xab, 0xa7, 0xce, 0x48	EFI_FILE_PROTOCOL. Write - Write () return EFI_WRITE_PROTECTED when the media is read-only.	Call Write() return EFI_WRITE_PROTECTED when the media is read-only.
5.7.3.15.17	0x3c0a4e4a, 0x43f4, 0x4b24, 0xb7, 0x64, 0xd8, 0x3c, 0x18, 0x63, 0xab, 0x81	EFI_FILE_PROTOCOL. Flush - Flush () return EFI_WRITE_PROTECTED when the media is read-only.	Call Flush() return EFI_WRITE_PROTECTED when the media is read-only.
5.7.3.15.18	0xece0ade2, 0x027e, 0x4c21, 0x91, 0x50, 0x33, 0x3c, 0x3e, 0x47, 0xea, 0x0b	EFI_FILE_PROTOCOL. Read - Read () return EFI_SUCCESS and the output should be consistent in multi read operations.	1. Call SetPosition() to set the position at 0. 2. Call Read() to read all content and save to FileBuf and get the file size. 3. Call GetPosition() to get the current position after the read operation. 4. Three returned status should be EFI_SUCCESS, Position should equal with file size, and the FileSize of EFI_FILE_INFO should be equal with file size. 5. Read the file from variable positions, the output should be consistent with file content read from step 2.
5.7.3.15.19	0x5ee32a7f, 0x0a63, 0x4803, 0x8a, 0xe8, 0x01, 0x9c, 0x07, 0x2a, 0xed, 0xb1	EFI_FILE_PROTOCOL. Delete - Delete () return EFI_WARN_DELETE_FAILURE.	Call Delete () return EFI_WARN_DELETE_FAILURE
5.7.3.15.20	0x3f8b11ec, 0x6b9e, 0x440c, 0x92, 0x0b, 0xb5, 0x63, 0xf3, 0xfd, 0x2b, 0xa7	EFI_FILE_PROTOCOL. Close - Close () one existed opened file return EFI_SUCCESS.	1. Call Open() to open one existed file. 2. Call Close () return EFI_SUCCESS.

9.4 EFI_DISK_IO_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_FILE_PROTOCOL Section.

9.4.1 ReadDisk()

Number	GUID	Assertion	Test Description
5.7.4.1.1	0x26912470, 0xf463, 0x4f8e, 0x8a, 0x33, 0xf3, 0x8f, 0x9c, 0xc8, 0x0d, 0x04	**EFI_DISK_IO_PROTOCOL.ReadDisk – ReadDisk()** returns **EFI_SUCCESS** with valid parameter.	Locate Block I/O interface that is associated with specified Disk I/O interface. For device with a **EFI_BLOCK_IO_MEDIA.MediaPresent** value of **TRUE**, and for different valid *OffSet* parameter and *BufferSize* parameter: 1. Call **ReadDisk()** with the *OffSet* and *BufferSize* Expected Behavior: The return code of **ReadDisk()** should be **EFI_SUCCESS**.
5.7.4.1.2	0x9603aba0, 0xb4dd, 0x4ab6, 0x93, 0xcb, 0x52, 0x3a, 0x5b, 0x6f, 0xa5, 0x58	**EFI_DISK_IO_PROTOCOL.ReadDisk – ReadDisk()** returns **EFI_MEDIA_CHANGED** with *MediaId* is not the ID for the current media in the device.	Locate Block I/O interface that is associated with specified Disk I/O interface. For device with a **EFI_BLOCK_IO_MEDIA.MediaPresent** value of **TRUE**: 1. Call **ReadDisk()** with valid parameters and a *MediaId* value of actual *MediaId* + 5. 2. Call **ReadDisk()** with valid parameters and a *MediaId* value of actual *MediaId* + 1. 3. Call **ReadDisk()** with valid parameters and a *MediaId* value of actual *MediaId* – 1. 4. Call **ReadDisk()** with valid parameters and a *MediaId* value of actual *MediaId* – 5. 5. Call **ReadDisk()** with valid parameters and a *MediaId* value of 0. Expected Behavior: For that new *MediaId* not equal to old *MediaId*, the return code must be **EFI_MEDIA_CHANGED**.

Number	GUID	Assertion	Test Description
5.7.4.1.3	0x6a6d39d0, 0x311d, 0x410f, 0x96, 0x2e, 0x96, 0xef, 0xfb, 0x39, 0x99, 0x44	**EFI_DISK_IO_PROTOCOL.ReadDisk – ReadDisk()** returns **EFI_INVALID_PARAMETER** with invalid device addresses.	Locate Block I/O interface that is associated with specified Disk I/O interface. For device with a **EFI_BLOCK_IO_MEDIA.MediaPresent** value of **TRUE**: 1. Call **ReadDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** + 1. 2. Call **ReadDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** + 10. 3. Call **ReadDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** – *BufferSize* + 1. 4. Call **ReadDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** – *BufferSize* + 2. 5. Call **ReadDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** – *BufferSize* + 3. 6. Call **ReadDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** – *BufferSize* + 4. Expected Behavior: The return code must be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.7.4.1.4	0xb0d7a6e7, 0x49f1, 0x40d5, 0xa9, 0x29, 0x1a, 0xd5, 0xd4, 0x27, 0x70, 0xbf	`EFI_DISK_IO_PROTOCOL.ReadDisk – ReadDisk()` returns `EFI_NO_MEDIA` with no media present in the device.	Locate Block I/O interface that is associated with specified Disk I/O interface. For device with a `EFI_BLOCK_IO_MEDIA.MediaPresent` value of `FALSE`: 1. Call `ReadDisk()` with valid parameter. Expected Behavior: The return code must be `EFI_NO_MEDIA`.

9.4.2 WriteDisk()

Number	GUID	Assertion	Test Description
5.7.4.2.1	0xc3d66c15, 0xb8ad, 0x45ad, 0xbe, 0xb7, 0x38, 0xfe, 0xc9, 0x52, 0x81, 0x5e	**EFI_DISK_IO_PROTOCOL.WriteDisk – WriteDisk()** returns **EFI_SUCCESS** to write proper data to non-readonly disk with valid parameter.	Locate Block I/O interface that is associated with specified Disk I/O interface. For non-readonly disk with a **EFI_BLOCK_IO_MEDIA.MediaPresent** value of **TRUE** and for different valid *OffSet* parameter and *BufferSize* parameter: 1. Call **ReadDisk()** with the *OffSet* and *BufferSize*. 2. Call **WriteDisk()** with same *OffSet* and *BufferSize* to write the specified buffer (different to buffer read from the last call of **ReadDisk()**) to the disk. 3. Call **ReadDisk()** with same *OffSet* and *BufferSize*. 4. Call **WriteDisk()** with same *OffSet* and *BufferSize* to write the buffer data read from the first **ReadDisk()** call. 5. Call **ReadDisk()** with same *OffSet* and *BufferSize* again. Expected Behavior: For each action, the return code should be **EFI_SUCCESS**. For each *OffSet* and *BufferSize*, the buffer data read by first and last calling **ReadDisk()** should be the same. For each *OffSet* and *BufferSize*, the buffer data return in the second call of **ReadDisk()** should be the same with the originally buffer data written to device in the first call of **WriteDisk()**.

Number	GUID	Assertion	Test Description
5.7.4.2.2	0x36d696b1, 0x1902, 0x46b7, 0x9a, 0x62, 0x85, 0x25, 0x1d, 0xf5, 0xec, 0x25	**EFI_DISK_IO_PROTOCOL.WriteDisk – WriteDisk()** returns **EFI_MEDIA_CHANGED** with *MediaId* is not the ID for the current media in the device.	Locate Block I/O interface that is associated with specified Disk I/O interface. For non-readonly device with a **EFI_BLOCK_IO_MEDIA.MediaPresent** value of **TRUE**: 1. Call **WriteDisk()** with valid parameters and a *MediaId* value of actual **EFI_BLOCK_IO_MEDIA.MediaId** + 5. 2. Call **WriteDisk()** with valid parameters and a *MediaId* value of actual **EFI_BLOCK_IO_MEDIA.MediaId** + 1. 3. Call **WriteDisk()** with valid parameters and a *MediaId* value of actual **EFI_BLOCK_IO_MEDIA.MediaId** – 1. 4. Call **WriteDisk()** with valid parameters and a *MediaId* value of actual **EFI_BLOCK_IO_MEDIA.MediaId** – 5. 5. Call **WriteDisk()** with valid parameters and a *MediaId* value of 0. Expected Behavior: For that new *MediaId* not equal to old *MediaId*, the return code must be **EFI_MEDIA_CHANGED**.

Number	GUID	Assertion	Test Description
5.7.4.2.3	0xc6eea54a, 0xde3a, 0x425a, 0xa6, 0x42, 0x79, 0xf4, 0xb7, 0x9a, 0x62, 0x36	**EFI_DISK_IO_PROTOCOL.WriteDisk – WriteDisk()** returns **EFI_INVALID_PARAMETER** with invalid device addresses.	Locate Block I/O interface that is associated with specified Disk I/O interface. For non-readonly device with a **EFI_BLOCK_IO_MEDIA.MediaPresent** value of **TRUE**: 1. Call **WriteDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** + 1. 2. Call **WriteDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** + 10. 3. Call **WriteDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** – *BufferSize* + 1. 4. Call **WriteDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** – *BufferSize* + 2. 5. Call **WriteDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** – *BufferSize* + 3. 6. Call **WriteDisk()** with an *OffSet* value of **EFI_BLOCK_IO_MEDIA.LastBlock * EFI_BLOCK_IO_MEDIA.BlockSize** – *BufferSize* + 4. Expected Behavior: The return code must be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.7.4.2.4	0x83a99320, 0x0831, 0x42d6, 0x8b, 0xec, 0x8d, 0xfd, 0x3d, 0xe4, 0x63, 0x78	EFI_DISK_IO_PROTOCOL.WriteDisk – WriteDisk() returns EFI_WRITE_PROTECTED with a write-protected device.	Locate Block I/O interface that is associated with specified Disk I/O interface. For read-only device with a EFI_BLOCK_IO_MEDIA.MediaPresent value of TRUE: 1. Call WriteDisk() with valid parameter to write data to device. Expected Behavior: The return code must be EFI_WRITE_PROTECTED.
5.7.4.2.5	0x0299b063, 0x21a8, 0x4811, 0x80, 0xe2, 0x8c, 0x4f, 0xfd, 0x3e, 0xd0, 0xa4	EFI_DISK_IO_PROTOCOL.WriteDisk – WriteDisk() returns EFI_NO_MEDIA with no media in the device.	Locate Block I/O interface that is associated with specified Disk I/O interface. For device with a EFI_BLOCK_IO_MEDIA.MediaPresent value of FALSE: 1. Call WriteDisk() with valid parameter to write data to device. Expected Behavior: The return code must be EFI_NO_MEDIA.

9.5 EFI_BLOCK_IO_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_BLOCK_IO_ PROTOCOL Section.

9.5.1 Reset()

Number	GUID	Assertion	Test Description
5.7.5.1.1	0x61ee3a34, 0x62a2, 0x4214, 0xb0, 0x76, 0x50, 0x73, 0xb1, 0x77, 0x15, 0x6c	EFI_BLOCK_IO_PROTOCOL.Reset - Reset() returns EFI_SUCCESS with an *ExtendedVerification* value of TRUE.	1. Call Reset() with an *ExtendedVerification* value of TRUE Expected Behavior: The return code should be EFI_SUCCESS. The private data for the device, which is stored in Media data structure, should be kept unchanged,

Number	GUID	Assertion	Test Description
5.7.5.1.2	0x98530f3d, 0x8bd8, 0x44a1, 0x9d, 0x06, 0x08, 0x03, 0x9f, 0xdf, 0xec, 0x63	**EFI_BLOCK_IO_PROTOCOL.Reset** – **Reset()** returns **EFI_SUCCESS** with an *ExtendedVerification* value of **FALSE**.	1. Call **Reset()** with an *ExtendedVerification* value of **FALSE** Expected Behavior: The return code should be **EFI_SUCCESS**. The private data for the device, which is stored in Media data structure, should be kept unchanged.

9.5.2 ReadBlocks()

Number	GUID	Assertion	Test Description
5.7.5.2.1	0x9efe26c2, 0xc565, 0x478a, 0xa0, 0xb4, 0x05, 0xa8, 0xfd, 0x2e, 0x7e, 0x3e	**EFI_BLOCK_IO_PROTOCOL.ReadBlocks** – **ReadBlocks()** returns **EFI_SUCCESS** with valid parameter.	(Can only be invoked when media is present.) 1. Call **ReadBlocks()** with different *LBA* and *BufferSize*. The return code should be **EFI_SUCCESS**.
5.7.5.2.2	0x6dec8f5c, 0xf6ec, 0x47b4, 0xbb, 0x0c, 0xaa, 0x4a, 0x69, 0x39, 0xe2, 0xf0	**EFI_BLOCK_IO_PROTOCOL.ReadBlocks** – **ReadBlocks()** returns **EFI_MEDIA_CHANGED** with *MediaId* is not the ID for the current media in the device.	For device with a **MediaPresent** value of **TRUE**: 1. Call **ReadBlocks()** with valid parameters and a *MediaId* value of actual *MediaId* + 5 2. Call **ReadBlocks()** with valid parameters and a *MediaId* value of actual *MediaId* + 1 3. Call **ReadBlocks()** with valid parameters and a *MediaId* value of actual *MediaId* – 1 4. Call **ReadBlocks()** with valid parameters and a *MediaId* value of actual *MediaId* – 5 5. Call **ReadBlocks()** with valid parameters and a *MediaId* value of 0 Expected Behavior: The return code must be **EFI_MEDIA_CHANGED**.

Number	GUID	Assertion	Test Description
5.7.5.2.3	0x05927e73, 0x8b41, 0x4cc7, 0x8e, 0xf2, 0x7c, 0x7a, 0xfb, 0x78, 0xf5, 0x3e	**EFI_BLOCK_IO_PROTOCOL.ReadBlocks – ReadBlocks()** returns **EFI_BAD_BUFFER_SIZE** with invalid *BufferSize* parameter. (Can only be invoked when media is present.)	For device with a **MediaPresent** value of **TRUE** and a **BlockSize** value other than 1: 1. Call **ReadBlocks()** with valid parameters and a *BufferSize* value of **BlockSize** + 1 2. Call **ReadBlocks()** with valid parameters and a *BufferSize* value of 2**BlockSize* – 1 3. Call **ReadBlocks()** with valid parameters and a *BufferSize* value of 2***BlockSize** + 1 4. Call **ReadBlocks()** with valid parameters and a *BufferSize* value of 3***BlockSize** – 1 Expected Behavior: All return codes must be **EFI_BAD_BUFFER_SIZE**.
5.7.5.2.4	0x09de1965, 0x3719, 0x463b, 0xa8, 0xd1, 0xd2, 0x78, 0xd7, 0xd6, 0x58, 0x2c	**EFI_BLOCK_IO_PROTOCOL.ReadBlocks – ReadBlocks()** returns **EFI_INVALID_PARAMETER** with invalid *LBA* parameter.	For device with a **MediaPresent** value of **TRUE**: 1. Call **ReadBlocks()** with valid parameters and an *LBA* value of **LastBlock** + 1 2. Call **ReadBlocks()** with valid parameters and an *LBA* value of **LastBlock** + 100 3. Call **ReadBlocks()** with valid parameters and an *LBA* value of **LastBlock** – *BufferSize/* **BlockSize** + 1 4. Call **ReadBlocks()** with valid parameters and an *LBA* value of **LastBlock** – *BufferSize/* **BlockSize** + 2 5. Call **ReadBlocks()** with valid parameters and an *LBA* value of **LastBlock** – *BufferSize/* **BlockSize** + 3 6. Call **ReadBlocks()** with valid parameters and an *LBA* value of **LastBlock** – *BufferSize/* **BlockSize** + 100 Expected Behavior: The return code must be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.7.5.2.5	0x91cfde2c, 0x619e, 0x4c88, 0x80, 0x0d, 0x99, 0xce, 0x53, 0xad, 0x3b, 0x25	EFI_BLOCK_IO_PROTOCOL.ReadBlocks – ReadBlocks() returns EFI_NO_MEDIA with no media present in the device.	For device with a MediaPresent value of FALSE: 1. Call ReadBlocks() with valid parameter. Expected Behavior: The return code must be EFI_NO_MEDIA.
5.7.5.2.6	0x8cf48053, 0x8e2e, 0x40c9, 0x90, 0xfa, 0x65, 0x33, 0x0b, 0xbf, 0x33, 0x69	EFI_BLOCK_IO_PROTOCOL.ReadBlocks – ReadBlocks() returns EFI_INVALID_PARAMETER with *Buffer* is not on proper lower alignment. (Can only be invoked when media present and IoAlign is larger than 1.)	For device with a MediaPresent value of TRUE and IoAlign more than 1: 1. Call ReadBlocks() with valid parameter and a *Buffer* value of (*Buffer*/IoAlign) * IoAlign + Remainder (Remainder goes from 1 to Min(IoAlign-1, 5)). Expected Behavior: The return code must be EFI_INVALID_PARAMETER.
5.7.5.2.7	0x9284cf69, 0x7570, 0x4da4, 0xa7, 0xa2, 0x40, 0x5d, 0x27, 0x9d, 0x0c, 0xa7	EFI_BLOCK_IO_PROTOCOL.ReadBlocks – ReadBlocks() returns EFI_INVALID_PARAMETER with *Buffer* is not on proper alignment. (Can only be invoked when media present and IoAlign is larger than 1.)	For device with a MediaPresent value of TRUE and IoAlign more than 1: 1. Call ReadBlocks() with valid parameter and a *Buffer* value of (*Buffer*/IoAlign) * IoAlign + Remainder (Remainder goes from IoAlign-1 down to Max(IoAlign-6, 1)). Expected Behavior: The return code must be EFI_INVALID_PARAMETER.

9.5.3 WriteBlocks()

Number	GUID	Assertion	Test Description
5.7.5.3.1	0x7bbdf28f, 0xb2ea, 0x42c0, 0xa8, 0xfe, 0x6a, 0xdc, 0x00, 0x38, 0x35, 0x77	EFI_BLOCK_IO_PROTOCOL.Writeblocks – WriteBlocks() returns EFI_SUCCESS with valid parameters.	(Can only be invoked when media is present and not read-only.) 1. Call ReadBlocks() to get the original data in the media. 2. Call WriteBlocks() with the new data. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.5.3.2	0x1fb19cbd, 0x7219, 0x4853, 0xa2, 0xaa, 0xeb, 0xe5, 0x17, 0xaa, 0xad, 0xe6	**EFI_BLOCK_IO_PROTOCOL.Writeblocks – ReadBlocks()** gets the same data as what are written before.	(Can only be invoked when media is present and not read-only.) 1. Call **ReadBlocks()** to get the original data in the media. 2. Call **WriteBlocks()** with the new data. 3. Call **ReadBlocks()** to get the data in the media. The data should be the same as the new data written before.
5.7.5.3.3	0x48340af1, 0x8425, 0x4847, 0xaa, 0x69, 0x56, 0x52, 0xd6, 0x61, 0x6e, 0x08	**EFI_BLOCK_IO_PROTOCOL.Writeblocks – WriteBlocks()** must return **EFI_SUCCESS** after being called twice with valid parameters.	(Can only be invoked when media is present and not read-only.) 1. Call **ReadBlocks()** to get the original data in the media. 2. Call **WriteBlocks()** with the new data. 3. Call **ReadBlocks()** to get the data in the media 4. Call **WriteBlocks()** with the original data. The return code should be **EFI_SUCCESS**.
5.7.5.3.4	0xa4383f2b, 0xf875, 0x4f57, 0x95, 0xfe, 0xce, 0x65, 0x5a, 0x4d, 0xc6, 0xb0	**EFI_BLOCK_IO_PROTOCOL.Writeblocks – WriteBlocks()** returns **EFI_MEDIA_CHANGED** with invalid *MediaId*.	For non-readonly device with a **MediaPresent** value of **TRUE**: 1. Call **WriteBlocks()** with valid parameters and a *MediaId* value of actual *MediaId* + 5 2. Call **WriteBlocks()** with valid parameters and a *MediaId* value of actual *MediaId* + 1 3. Call **WriteBlocks()** with valid parameters and a *MediaId* value of actual *MediaId* – 1 4. Call **WriteBlocks()** with valid parameters and a *MediaId* value of actual *MediaId* – 5 5. Call **WriteBlocks()** with valid parameters and a *MediaId* value of 0 Expected Behavior: The return code must be **EFI_MEDIA_CHANGED**.

Number	GUID	Assertion	Test Description
5.7.5.3.5	0xbf9eabdd, 0x1745, 0x4418, 0xaf, 0xf8, 0x12, 0x5e, 0x02, 0x18, 0x94, 0xaa	**EFI_BLOCK_IO_PROTOCOL.Writeblocks – ReadBlocks()** get the original data written before.	(Can only be invoked when media is present and not read-only.) 1. Call **ReadBlocks()** to get the original data in the media. 2. Call **WriteBlocks()** with the new data. 3. Call **ReadBlocks()** to get the data in the media 4. Call **WriteBlocks()** with the original data. 5. Call **ReadBlocks()** to get the data in the media. The data should be the same as the original data written before.
5.7.5.3.6	0xa77c46e0, 0x6df6, 0x4d63, 0xaf, 0x8d, 0xae, 0xb7, 0xae, 0x7d, 0x2b, 0x12	**EFI_BLOCK_IO_PROTOCOL.Writeblocks – WriteBlocks()** returns **EFI_BAD_BUFFER_SIZE** with invalid *BufferSize*.	For non-readonly device with a **MediaPresent** value of **TRUE**: 1. Call **WriteBlocks()** with valid parameters and a *BufferSize* value of **BlockSize** + 1. 2. Call **WriteBlocks()** with valid parameters and a *BufferSize* value of 2***BlockSize** – 1. 3. Call **WriteBlocks()** with valid parameters and a *BufferSize* value of 2***BlockSize** + 1 4. Call **WriteBlocks()** with valid parameters and a *BufferSize* value of 3***BlockSize** – 1. Expected Behavior: The return code must be **EFI_BAD_BUFFER_SIZE**.

Number	GUID	Assertion	Test Description
5.7.5.3.7	0x98e637f8, 0x9a1c, 0x42f9, 0xa6, 0xe2, 0x2e, 0xe8, 0x5f, 0x70, 0x2b, 0x98	**EFI_BLOCK_IO_PROTOCOL.Writeblocks – WriteBlocks()** return **EFI_INVALID_PARAMETER** with invalid *LBA* parameter.	For non-readonly device with a **MediaPresent** value of **TRUE**: 1. Call **WriteBlocks()** with valid parameters and an *LBA* value of **LastBlock** + 1. 2. Call **ReadBlocks()** with valid parameters and an *LBA* value of **LastBlock** + 100. 3. Call **ReadBlocks()** with valid parameters and an *LBA* value of **LastBlock** – *BufferSize*/ **BlockSize** + 1 4. Call **ReadBlocks()** with valid parameters and an *LBA* value of **LastBlock** – *BufferSize*/ **BlockSize** + 2. 5. Call **ReadBlocks()** with valid parameters and an *LBA* value of **LastBlock** – *BufferSize*/ **BlockSize** + 3. 6. Call **ReadBlocks()** with valid parameters and an *LBA* value of **LastBlock** – *BufferSize*/ **BlockSize** + 100. Expected Behavior: The return code must be **EFI_INVALID_PARAMETER**.
5.7.5.3.8	0xedb9cf57, 0x1900, 0x45f2, 0x9a, 0x5a, 0xf1, 0x3b, 0x31, 0xdf, 0x36, 0x6a	**EFI_BLOCK_IO_PROTOCOL.Writeblocks – WriteBlocks()** returns **EFI_WRITE_PROTECTED** with write-protected device.	For read-only device with a **MediaPresent** value of **TRUE**: 1. Call **WriteBlocks()** with valid parameter to write data to device. Expected Behavior: The return code must be **EFI_WRITE_PROTECTED**.
5.7.5.3.9	0x7abcfa31, 0x7456, 0x40ae, 0x93, 0x51, 0x1c, 0xf4, 0x50, 0x1c, 0x08, 0xc9	**EFI_BLOCK_IO_PROTOCOL.Writeblocks – WriteBlocks()** returns **EFI_NO_MEDIA** with no media in the device.	For non-readonly device with a **MediaPresent** value of **FALSE**: 1. Call **WriteBlocks()** with valid parameter to write data to device. Expected Behavior: The return code must be **EFI_NO_MEDIA**.

Number	GUID	Assertion	Test Description
5.7.5.3.10	0x8a7d6ab3, 0x2c11, 0x41e3, 0xa4, 0x30, 0xfe, 0x3c, 0x50, 0xcc, 0x57, 0xad	EFI_BLOCK_IO_PROTOCOL.Writeblocks – WriteBlocks() returns EFI_INVALID_PARAMETER with *Buffer* is not on proper lower alignment.	For non-readonly device with a MediaPresent value of TRUE and IoAlign more than 1: 1. Call WriteBlocks() with valid parameter and a *Buffer* value of (*Buffer*/IoAlign) * IoAlign + Remainder (Remainder goes from 1 to Min(IoAlign-1, 5)). Expected Behavior: The return code must be EFI_INVALID_PARAMETER.
5.7.5.3.11	0xb9d363bf, 0x9c50, 0x4671, 0x88, 0x55, 0xce, 0xfc, 0xc6, 0xb8, 0x24, 0xaa	EFI_BLOCK_IO_PROTOCOL.Writeblocks – WriteBlocks() returns EFI_INVALID_PARAMETER with *Buffer* is not on proper alignment.	For non-readonly device with a MediaPresent value of TRUE and IoAlign more than 1: 1. Call WriteBlocks() with valid parameter and a *Buffer* value of (*Buffer*/IoAlign) * IoAlign + Remainder (Remainder goes from IoAlign-1 down to Max(IoAlign-6, 1)). Expected Behavior: The return code must be EFI_INVALID_PARAMETER.

9.5.4 FlushBlocks()

Number	GUID	Assertion	Test Description
5.7.5.4.1	0x5f220c61, 0x24b5, 0x4c71, 0x8e, 0x5a, 0x78, 0xbd, 0x0a, 0xc6, 0x77, 0xf6	EFI_BLOCK_IO_PROTOCOL.FlushBlocks – FlushBlocks() returns EFI_NO_MEDIA with no media presented	For device with a MediaPresent value of FALSE and a WriteCaching value of TRUE: 1. Call FlushBlocks. Expected Behavior: The return code must be EFI_NO_MEDIA.

9.5.5 Media Info Check

No	GUID	Assertion	Test Description
5.7.5.5.1	0xb8a45208, 0xf7b0, 0x443c, 0x8c, 0xce, 0xeb, 0x81, 0xb6, 0x6c, 0x00, 0x4a	EFI_BLOCK_IO_PROTOCOL.Media-LogicalBlocksPerPhysicalBlock should be 0 when LogicalPartition is TRUE and Revision is greater than or equal to EFI_BLOCK_IO_PROTOCOL_REVISION2.	LogicalBlocksPerPhysicalBlock should be 0 when LogicalPartition is TRUE and Revision is greater than or equal to EFI_BLOCK_IO_PROTOCOL_REVISION2.

5.7.5.5.2	0xe08ff5f4, 0x92de, 0x4cc9, 0x81, 0x22, 0x6b, 0x48, 0x7c, 0x67, 0x0c, 0x9b	EFI_BLOCK_IO_PROTOCOL. Media-OptimalTransferLengthGranularity should be 0 when LogicalPartition is TRUE and Revision is greater than or equal to EFI_BLOCK_IO_PROTOCOL_REVISION3.	OptimalTransferLengthGranularity should be 0 when LogicalPartition is TRUE and Revision is greater than or equal to EFI_BLOCK_IO_PROTOCOL_REVISION3.

9.6 EFI_UNICODE_COLLATION_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_UNICODE_COLLATION_ PROTOCOL Section.

9.6.1 StriColl()

Number	GUID	Assertion	Test Description
5.7.6.1.1	0x3bf9028a, 0x599c, 0x44e0, 0xa7, 0xdf, 0xa6, 0x87, 0xcf, 0x9e, 0x15, 0xf4	EFI_UNICODE_COLLATION_PROTOCOL.StriColl - StriColl() with valid parameter returns correct status of comparison between *String1* and *String2*.	1. Call StriColl(). The return code should correspond to the string comparison result.

9.6.2 MetaiMatch()

Number	GUID	Assertion	Test Description
5.7.6.2.1	0x60291ba4, 0x7170, 0x4f5c, 0x84, 0x20, 0x11, 0x07, 0x85, 0x49, 0x2e, 0x6d	EFI_UNICODE_COLLATION_PROTOCOL.MetaiMatch - MetaiMatch() returns correct status of pattern match.	1. Call MetaiMatch(). The return code should correspond to the pattern match result.

9.6.3 StrLwr()

Number	GUID	Assertion	Test Description
5.7.6.3.1	0x9d69a782, 0x672b, 0x43db, 0xac, 0x24, 0x16, 0x59, 0xa3, 0x9d, 0xa7, 0x5e	EFI_UNICODE_COLLATION_PROTOCOL.StrLwr - StrLwr() convert the string to lowercase.	1. Call StrLwr(). It should convert the string to lowercase.

5.7.6.3.2	0x2e743a2a, 0x52a3, 0x411d, 0x95, 0x2a, 0x42, 0x0c, 0x47, 0x76, 0x90, 0x4c	`EFI_UNICODE_COLLATION_PROTOCOL.StrLwr` – `StrLwr()` convert the string to lowercase.	1. Call `StrLwr()` to convert string to lowercase and store lowercase string in buffer. 2. Call `StrUpr()` to convert lower case string to uppercase. 3. Call `StrLwr()` to convert uppercase string to lowercase. The lowercase string should be equal to lowercase string stored in buffer.

9.6.4 StrUpr()

Number	GUID	Assertion	Test Description
5.7.6.4.1	0x1b8390f4, 0xc5ac, 0x4342, 0x85, 0x55, 0x70, 0x74, 0xe5, 0xa2, 0x10, 0x2b	`EFI_UNICODE_COLLATION_PROTOCOL.StrUpr` – `StrUpr()` convert the string to Uppercase.	1. Call `StrUpr()`. It should convert the string to uppercase.
5.7.6.4.2	0x6179f1fb, 0x54c5, 0x4844, 0xba, 0x17, 0x31, 0x4f, 0xe3, 0x57, 0xb4, 0xe3	`EFI_UNICODE_COLLATION_PROTOCOL.StrUpr` – `StrUpr()` convert the string to Uppercase.	1. Call `StrUpr()` to convert string to uppercase and store uppercase string in buffer. 2. Call `StrLwr()` to convert upper case string to lowercase. 3. Call `StrUpr()` to convert lowercase string to uppercase. The uppercase string should be equal to uppercase string stored in buffer.

9.6.5 FatToStr()

Number	GUID	Assertion	Test Description
5.7.6.5.1	0x07f17163, 0x6f7d, 0x428f, 0xad, 0x13, 0xe4, 0xd0, 0x0b, 0x3a, 0x45, 0x64	`EFI_UNICODE_COLLATION_PROTOCOL.FatToStr` – `FatToStr()` with *`FatSize`* equal to the size of Fat String converts Fat string to Unicode string correctly.	1. Call `FatToStr()` with *`FatSize`* equal to the size of Fat String. It should convert Fat string to Unicode string correctly.

Number	GUID	Assertion	Test Description
5.7.6.5.2	0x17ea04a7, 0xa56e, 0x4733, 0x83, 0x20, 0x79, 0x33, 0x09, 0x31, 0xef, 0xac	**EFI_UNICODE_COLLATION_PROTOCOL.FatToStr - FatToStr()** with *FatSize* larger than the size of Fat String converts Fat string to Unicode string correctly	1. Call **FatToStr()** with *FatSize* larger than the size of Fat String. It should convert Fat string to Unicode string correctly.
5.7.6.5.3	0x2e89ebe3, 0x44bd, 0x4e02, 0xba, 0x50, 0x90, 0x05, 0xc0, 0xfc, 0x08, 0xdd	**EFI_UNICODE_COLLATION_PROTOCOL.FatToStr - FatToStr()** with *FatSize* smaller than the size of Fat String converts Fat string to Unicode string correctly.	1. Call **FatToStr()** with *FatSize* smaller than the size of Fat String. It should convert Fat string to Unicode string correctly.

9.6.6 StrToFat()

Number	GUID	Assertion	Test Description
5.7.6.6.1	0x6f780647, 0xef48, 0x4c1c, 0x87, 0xa6, 0x95, 0xe2, 0x50, 0x0e, 0x2e, 0x0b	**EFI_UNICODE_COLLATION_PROTOCOL.StrToFat - StrToFat()** with *FatSize* equal to the size of Fat String converts Unicode string to Fat string correctly.	1. Call **StrToFat()** with *FatSize* equal to the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns **TRUE** and characters were substituted with '_'.
5.7.6.6.2	0x5eea066e, 0xf73e, 0x4d36, 0x91, 0x25, 0xa0, 0x8a, 0x54, 0x6e, 0xee, 0x27	**EFI_UNICODE_COLLATION_PROTOCOL.StrToFat - StrToFat()** with *FatSize* larger than the size of Fat String converts Unicode string to Fat string correctly.	1. Call **StrToFat()** with *FatSize* larger than the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns **TRUE** and characters were substituted with '_'.
5.7.6.6.3	0x58ae3ae9, 0x3dac, 0x41bf, 0x8d, 0x01, 0xd5, 0x91, 0xe3, 0xef, 0x62, 0x62	**EFI_UNICODE_COLLATION_PROTOCOL.StrToFat - StrToFat()** with *FatSize* smaller than the size of Fat String converts Unicode string to Fat string correctly.	1. Call **StrToFat()** with *FatSize* smaller than the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns **TRUE** and characters were substituted with '_'.

9.7 EFI_UNICODE_COLLATION2_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_UNICODE_COLLATION2_ PROTOCOL Section.

9.7.1 StriColl()

Number	GUID	Assertion	Test Description
5.7.7.1.1	0x6a69637d, 0x5ada, 0x40fd, 0x93, 0x05, 0xe1, 0x06, 0xc9, 0xff, 0xa1, 0xbd	**EFI_UNICODE_COLLATION2_PROTOCOL.StriColl - StriColl()** with valid parameter returns correct status of comparison between *String1* and *String2*.	1. Call **StriColl()**. The return code should correspond to the string comparison result.

9.7.2 MetaiMatch()

Number	GUID	Assertion	Test Description
5.7.7.2.1	0x49f68d03, 0xfef1, 0x460f, 0x8e, 0xdd, 0x27, 0xdb, 0x15, 0x22, 0xa3, 0xa3	**EFI_UNICODE_COLLATION2_PROTOCOL.MetaiMatch - MetaiMatch()** returns correct status of pattern match.	1. Call **MetaiMatch()**. The return code should correspond to the pattern match result.

9.7.3 StrLwr()

Number	GUID	Assertion	Test Description
5.7.7.3.1	0xa8a08682, 0xf9d9, 0x4471, 0x85, 0x53, 0x48, 0x0b, 0x42, 0x1d, 0x66, 0x5b	**EFI_UNICODE_COLLATION2_PROTOCOL.StrLwr - StrLwr()** convert the string to lowercase.	1. Call **StrLwr()**. It should convert the string to lowercase.
5.7.7.3.2	0xfb87853f, 0xa47b, 0x405b, 0x85, 0x5f, 0xc6, 0xbe, 0x18, 0x8b, 0xc3, 0x30	**EFI_UNICODE_COLLATION2_PROTOCOL.StrLwr - StrLwr()** convert the string to lowercase.	1. Call **StrLwr()** to convert string to lowercase and store lowercase string in buffer. 2. Call **StrUpr()** to convert lower case string to uppercase. 3. Call **StrLwr()** to convert uppercase string to lowercase. The lowercase string should be equal to lowercase string stored in buffer.

9.7.4 StrUpr()

Number	GUID	Assertion	Test Description
5.7.7.4.1	0x6f390d73, 0xe8c7, 0x4032, 0xb5, 0xcb, 0xc0, 0xf6, 0xa8, 0x18, 0xe1, 0x87	`EFI_UNICODE_COLLATION2_PROTOCOL.StrUpr - StrUpr()` convert the string to Uppercase.	1. Call `StrUpr()`. It should convert the string to uppercase.
5.7.7.4.2	0xf559dbaa, 0xdeb6, 0x4591, 0xbf, 0x16, 0xf1, 0x4b, 0x50, 0x6c, 0xac, 0xae	`EFI_UNICODE_COLLATION2_PROTOCOL.StrUpr - StrUpr()` convert the string to Uppercase.	1. Call `StrUpr()` to convert string to uppercase and store uppercase string in buffer. 2. Call `StrLwr()` to convert upper case string to lowercase. 3. Call `StrUpr()` to convert lowercase string to uppercase. The uppercase string should be equal to uppercase string stored in buffer.

9.7.5 FatToStr()

Number	GUID	Assertion	Test Description
5.7.7.5.1	0x99a47923, 0xd2e9, 0x4114, 0xba, 0xc0, 0x46, 0x2b, 0xaa, 0x5a, 0xe5, 0xf3	`EFI_UNICODE_COLLATION2_PROTOCOL.FatToStr - FatToStr()` with *`FatSize`* equal to the size of Fat String converts Fat string to Unicode string correctly.	1. Call `FatToStr()` with *`FatSize`* equal to the size of Fat String. It should convert Fat string to Unicode string correctly.
5.7.7.5.2	0xd5dc3c74, 0x268a, 0x499d, 0xb3, 0x8b, 0xcb, 0x2d, 0x69, 0x71, 0x19, 0xb3	`EFI_UNICODE_COLLATION2_PROTOCOL.FatToStr - FatToStr()` with *`FatSize`* larger than the size of Fat String converts Fat string to Unicode string correctly	1. Call `FatToStr()` with *`FatSize`* larger than the size of Fat String. It should convert Fat string to Unicode string correctly.
5.7.7.5.3	0x305c644e, 0x002f, 0x466f, 0xae, 0x41, 0x4f, 0x22, 0xff, 0xda, 0x05, 0xfc	`EFI_UNICODE_COLLATION2_PROTOCOL.FatToStr - FatToStr()` with *`FatSize`* smaller than the size of Fat String converts Fat string to Unicode string correctly.	1. Call `FatToStr()` with *`FatSize`* smaller than the size of Fat String. It should convert Fat string to Unicode string correctly.

9.7.6 StrToFat()

Number	GUID	Assertion	Test Description
5.7.7.6.1	0x7b8b1cb5, 0xa3b9, 0x410d, 0x96, 0xf0, 0x3d, 0x88, 0x96, 0xe9, 0x03, 0x9a	**EFI_UNICODE_COLLATION2_PROTOCOL.StrToFat - StrToFat()** with *FatSize* equal to the size of Fat String converts Unicode string to Fat string correctly.	1. Call **StrToFat()** with *FatSize* equal to the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns **TRUE** and characters were substituted with '_'.
5.7.7.6.2	0x9c40c459, 0x0a09, 0x4382, 0x89, 0x79, 0x01, 0xea, 0x30, 0x35, 0xdd, 0xf4	**EFI_UNICODE_COLLATION2_PROTOCOL.StrToFat - StrToFat()** with *FatSize* larger than the size of Fat String converts Unicode string to Fat string correctly.	1. Call **StrToFat()** with *FatSize* larger than the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns **TRUE** and characters were substituted with '_'.
5.7.7.6.3	0x8d0e58cc, 0x4494, 0x4684, 0xaf, 0x6c, 0xdc, 0xf9, 0x1b, 0x77, 0x6c, 0x6b	**EFI_UNICODE_COLLATION2_PROTOCOL.StrToFat - StrToFat()** with *FatSize* smaller than the size of Fat String converts Unicode string to Fat string correctly.	1. Call **StrToFat()** with *FatSize* smaller than the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns **TRUE** and characters were substituted with '_'.

9.8 EFI_ATA_PASS_THRU_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_ATA_PASS_THRU_PROTOCOL Section .

9.8.1 GetNextPort()

Number	GUID	Assertion	Test Description
5.7.8.1.1	0xbad50e59, 0x9423, 0x427d, 0xa7, 0x5d, 0x69, 0x1c, 0x90, 0xb7, 0xf9, 0x75	**EFI_ATA_PASS_THRU_PROTOCOL.GetNextPort – GetNextPort()** should return invalid parameter if input port is invalid.	1. Call **GetNextPort()** with *Port* being a not available port. 2. The return code should be **EFI_INVALID_PARAMETER**.

5.7.8.1.2	0xc3e87aa1, 0x6e9c, 0x478f, 0x9b, 0xd5, 0x39, 0x50, 0x8, 0x01, 0x28, 0x96	**EFI_ATA_PASS_THRU_PROTOCOL.GetNextPort – GetNextPort()** should return invalid parameter if port is not 0xFFFF and port was not returned on a previous call.	1. Call **GetNextPort()** when *Port* is not 0xFFFF and *Port* was not returned on a previous call. 2. The return code should be **EFI_INVALID_PARAMETER**.
5.7.8.1.3	0x5f658292, 0xa409, 0x4d67, 0xba, 0x13, 0x4, 0xc2, 0x51, 0x85, 0xf2, 0x80	**EFI_ATA_PASS_THRU_PROTOCOL.GetNextPort – GetNextPort()** could iterate all available port.	1. Call **GetNextPort()** with *Port* as 0xFFFF to start iterate ports. 2. The iteration should ended up with a return code **EFI_NOT_FOUND**.

9.8.2 BuildDevicePath()

Number	GUID	Assertion	Test Description
5.7.8.2.1	0xd72e6a78, 0x5292, 0x4493, 0x90, 0x40, 0xb0, 0x44, 0x5a, 0x9c, 0x17, 0x14	**EFI_ATA_PASS_THRU_PROTOCOL.BuildDevicePath – BuildDevicePath()** with NULL parameter.	1. Call **BuildDevicePath()** with with *Port* and *PortMultiplierPort* identifying an available device and *DevicePath* being NULL. The return code should be **EFI_INVALID_PARAMETER**
5.7.8.2.2	0xa42a0e01, 0x7b80, 0x46e4, 0xa7, 0x57, 0x86, 0xc4, 0xec, 0x53, 0xf4, 0xe4	**EFI_ATA_PASS_THRU_PROTOCOL.BuildDevicePath – BuildDevicePath()** with invalid port.	1. Call **BuildDevicePath()** with with *Port* being not available and other parameters valid. The return code should be **EFI_NOT_FOUND**
5.7.8.2.3	0x322f00c1, 0xf6bf, 0x41ed, 0xae, 0xfd, 0xaa, 0xc4, 0x8f, 0x3f, 0xa9, 0xdb	**EFI_ATA_PASS_THRU_PROTOCOL.BuildDevicePath – BuildDevicePath()** with invalid device.	1. Call **BuildDevicePath()** with with *PortMultiplierPort* being not available and other parameters valid. The return code should be **EFI_NOT_FOUND**
5.7.8.2.4	0x230d44b6, 0xce53, 0x42b6, 0x9b, 0xa6, 0x3d, 0x11, 0x5d, 0x49, 0x2b, 0x33	**EFI_ATA_PASS_THRU_PROTOCOL.BuildDevicePath – BuildDevicePath()** with available device, device path should be created.	1. Call **BuildDevicePath()** with with *Port* and *PortMultiplierPort* identifying an available device. The return code should be **EFI_SUCCESS**

9.8.3 GetDevice()

Number	GUID	Assertion	Test Description
5.7.8.3.1	0x0f2f0849, 0x690b, 0x48ea, 0x8e, 0x35, 0x64, 0x36, 0x3f, 0xaa, 0x8c, 0x5c	**EFI_ATA_PASS_THRU_PROTOCOL.GetDevice – GetDevice()** with NULL device path.	1. Call **GetDevice()** with the *DevicePath* being NULL. The *Port* and *PortMultiplierPort* are valid, the return code should be **EFI_INVALID_PARAMETER**
5.7.8.3.2	0x7602bd0a, 0x1c05, 0x49e5, 0xa8, 0xd4, 0xc6, 0x3, 0x8c, 0x43, 0x9a, 0xf9	**EFI_ATA_PASS_THRU_PROTOCOL.GetDevice – GetDevice()** with NULL port.	1. Call **GetDevice()** with the *DevicePath* being valid. The *Port* being NULL, *PortMultiplierPort* is valid, the return code should be **EFI_INVALID_PARAMETER**
5.7.8.3.3	0x2b64d49a, 0x1f1b, 0x4610, 0xa2, 0x66, 0xde, 0x32, 0xa1, 0x7, 0x2b, 0x32	**EFI_ATA_PASS_THRU_PROTOCOL.GetDevice – GetDevice()** with NULL device.	1. Call **GetDevice()** with the *DevicePath* being valid. The *Port* being valid, *PortMultiplierPort* is NULL, the return code should be **EFI_INVALID_PARAMETER**
5.7.8.3.4	0x07830eaf, 0xba30, 0x4224, 0xab, 0xc4, 0x42, 0x42, 0x8b, 0x7a, 0x4, 0x5d	**EFI_ATA_PASS_THRU_PROTOCOL.GetDevice – GetDevice()** with invalid device path.	1. Call **GetDevice()** with the *DevicePath* being of type 'End Device Path'. The *Port* and *PortMultiplierPort* are valid, the return code should be **EFI_UNSUPPORTED** or **EFI_NOT_FOUND**
5.7.8.3.5	0x7ea827e4, 0x522c, 0x44b6, 0x99, 0xe4, 0x25, 0x93, 0x19, 0xba, 0xcc, 0x57	**EFI_ATA_PASS_THRU_PROTOCOL.GetDevice – GetDevice()** with correct device path. The device represented by this device path should be achieved.	1. Call **GetDevice()** with the *DevicePath* that representing one available device. The *Port* and *PortMultiplierPort* of the device should be got and the return code should be **EFI_SUCCESS**

9.8.4 ResetPort()

Number	GUID	Assertion	Test Description
5.7.8.4.1	0x5e0080d2, 0x4065, 0x4b92, 0xa4, 0x61, 0x52, 0x49, 0xf3, 0x8f, 0xaf, 0x55	**EFI_ATA_PASS_THRU_PROTOCOL.ResetPort – ResetPort()** with available port.	1. Call ResetPort**()** with *Port* as one available port. The return code should be **EFI_SUCCESS** or **EFI_UNSUPPORTED.**

9.8.5 ResetDevice()

Number	GUID	Assertion	Test Description
5.7.8.5.1	0x206ae2fc, 0x3f34, 0x4afe, 0x82, 0x44, 0x40, 0x27, 0x57, 0x60, 0x98, 0x31	**EFI_ATA_PASS_THRU_PROTOCOL.ResetDevice – ResetDevice()** with invalid port.	1. Call ResetDevice**()** with *Port* being invalid and *PortMultiplierPort* as zero. The return code should be **EFI_INVALID_PARAMETER**
5.7.8.5.2	0xd9378047, 0x9b4b, 0x4abf, 0xaa, 0x6b, 0xe3, 0xcd, 0xb6, 0xc4, 0x19, 0x39	**EFI_ATA_PASS_THRU_PROTOCOL.ResetDevice – ResetDevice()** with invalid device.	1. Call ResetDevice**()** with *Port* being valid and *PortMultiplierPort* being invalid. The return code should be **EFI_INVALID_PARAMETER**
5.7.8.5.3	0xa400bc81, 0x9e48, 0x469b, 0xa0, 0x97, 0xd0, 0x8, 0x45, 0xb6, 0x69, 0xe8	**EFI_ATA_PASS_THRU_PROTOCOL.ResetDevice – ResetDevice()** with available device.	1. Call ResetDevice**()** with *Port* and *PortMultiplierPort* as one available device. The return code should be **EFI_SUCCESS** or **EFI_UNSUPPORTED**

9.8.6 GetNextDevice()

Number	GUID	Assertion	Test Description
5.7.8.6.1	0xc564ad60, 0x32ce, 0x4f5f, 0x86, 0x7a, 0xef, 0x9f, 0xef, 0x5e, 0x94, 0xa2	**EFI_ATA_PASS_THRU_PROTOCOL.GetNextDevice –GetNextDevice()** with invalid device number.	1. Call **GetNextPort()** with *Port* as **0xFFFF** to start iterate ports. 2. Call **GetNextDevice()** with *Port* as one available port and *PortMultiplierPort* being invalid. 3. The iteration should ended up with a return code **EFI_INVALID_PARAMETER**.
5.7.8.6.2	0x0e5c99ba, 0xd36c, 0x4775, 0x91, 0x31, 0x76, 0x6a, 0x6e, 0x8c, 0x53, 0x6b	**EFI_ATA_PASS_THRU_PROTOCOL.GetNextDevice –GetNextDevice()** should return invalid parameter if *PortMultiplierPort* is not 0xFFFF and *PortMultiplierPort* was not returned on a previous call.	1. Call **GetNextPort()** when *PortMultiplierPort* is not 0xFFFF and *PortMultiplierPort* was not returned on a previous call. 2. The return code should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.7.8.6.3	0xd89631f3, 0xbd59, 0x4959, 0xba, 0x10, 0x3f, 0xa9, 0x94, 0x62, 0x02, 0xdf	**EFI_ATA_PASS_THRU_PROTOCOL.GetNextDevice – GetNextDevice()** could iterate all available devices on one port.	1.**GetNextPort()** with *Port* as **0xFFFF**to start iterate ports. 2. Call **GetNextPort()** with *Port* as one available port and *PortMultiplierPort* as 0xFFFF to start iterate devicess. 3. The iteration should ended up with a return code **EFI_NOT_FOUND**.

9.8.7 PassThru()

Number	GUID	Assertion	Test Description
5.7.8.7.1	0x7d6fcacd, 0x3463, 0x41c8, 0xa5, 0x1, 0xa2, 0x99, 0x40, 0x44, 0x59, 0xb8	**EFI_ATA_PASS_THRU_PROTOCOL.PassThru – PassThru()** with Non-IoAligned InDataBuffer.	1. Call **PassThru()**with *Event* being NULL *Packet.InDataBuffer* set to be not aligned with **EFI_ATA_PASS_THRU_PROTOCOL**.Mode.IoAlign. The return code should be **EFI_INVALID_PARAMETER**.
5.7.8.7.2	0x745295b5, 0xc36b, 0x4b23, 0xaf, 0xc7, 0xd4, 0xcc, 0xc0, 0x1d, 0xb6, 0x4f	**EFI_ATA_PASS_THRU_PROTOCOL.PassThru – PassThru()** with Non-IoAligned Asb.	1. Call **PassThru()**with *Event* being NULL *Packet.Asb* set to be not aligned with **EFI_ATA_PASS_THRU_PROTOCOL**.Mode.IoAlign. The return code should be **EFI_INVALID_PARAMETER**.
5.7.8.7.3	0xaf9489a2, 0x23f3, 0x4962, 0x9d, 0x8f, 0xd2, 0xc0, 0xa7, 0xcb, 0x2f, 0xb1	**EFI_ATA_PASS_THRU_PROTOCOL.PassThru – PassThru()** with Non-IoAligned OutDataBuffer.	1. Call **PassThru()**with *Event* being NULL *Packet.OutDataBuffer* set to be not aligned with **EFI_ATA_PASS_THRU_PROTOCOL**.Mode.IoAlign. The return code should be **EFI_INVALID_PARAMETER**.
5.7.8.7.4	0xd584b074, 0xa8cd, 0x438c, 0xb5, 0x18, 0xb1, 0xec, 0x59, 0xfa, 0xc8, 0xee	**EFI_ATA_PASS_THRU_PROTOCOL.PassThru – PassThru()** with invalid port.	1. Call **PassThru()**with *Event* being NULL *Packet* contents valid, *Port* as invalid. The return code should be **EFI_INVALID_PARAMETER**.
5.7.8.7.5	0x4cd806fd, 0x3742, 0x44e9, 0xa6, 0x19, 0xdf, 0x2d, 0x37, 0x47, 0xe7, 0x8f	**EFI_ATA_PASS_THRU_PROTOCOL.PassThru – PassThru()** with invalid device.	1. Call **PassThru()**with *Event* being NULL *Packet* contents valid, *Port* as valid, *PortMultiplierPort* being invalid. The return code should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.7.8.7.6	0xa648ab45, 0x898b, 0x4b44, 0xab, 0x9e, 0x24, 0x6b, 0xc6, 0x49, 0xc9, 0xfd	**EFI_ATA_PASS_THRU_PROTOCOL.PassThru – PassThru()** with too long buffer size.	1. Call **PassThru()** with *Event* being NULL *Packet.InDataBufferLength* being 0xFFFFFFFF, *Port* and *PortMultiplierPort* being valid. The return code should be **EFI_BAD_BUFFER_SIZE**.
5.7.8.7.7	0xe5c8314a, 0xa2b8, 0x42d2, 0xb1, 0x27, 0x97, 0xad, 0x78, 0x74, 0xd5, 0x30	**EFI_ATA_PASS_THRU_PROTOCOL.PassThru – PassThru()** sends ATA command 'Identify Device' to an available device with several valid **EFI_ATA_PASS_THRU_COMMAND_PACKET** and **EFI_EVENT** inputs.	Below are the three possible separate test procedure that corresponds to this test assertion 1. Call **PassThru()** with *Port* and *PortMultiplierPort* representing one available device. The *Packet.Acb.AtaCommand* is set to be the value of 'Identify Device' command 0xEC, *Packet.Asb* and *Packet.InDataBuffer* are allocated and adjusted according to **EFI_ATA_PASS_THRU_PROTOCOL**.Mode.IoAlign value, *Packet.Timeout* set to be 2 seconds, *Packet.Length* set to be block granularity, *Packet.InTransferLength* being 1 to indicate one block, *Packet.Protocol* being **EFI_ATA_PASS_THRU_PROTOCOL_PIO_DATA_IN**. The return code should be **EFI_SUCCESS** and *Packet.Asb.AtaStatus* should reflect the ATA command has been executed successfully.

Number	GUID	Assertion	Test Description
			2. Call **PassThru()** with *Port* and *PortMultiplierPort* representing one available device. The *Packet.Acb.AtaCommand* is set to be the value of ‘Identify Device’ command 0xEC, *Packet.Asb* and *Packet.InDataBuffer* are allocated and adjusted according to **EFI_ATA_PASS_THRU_PROTOCOL**.Mode.IoAlign value, *Packet.Timeout* set to be 2 seconds, *Packet.Length* set to be byte granularity, *Packet.InTransferLength* being 512 to indicate one block, *Packet.Protocol* being **EFI_ATA_PASS_THRU_PROTOCOL_PIO_DATA_IN**. The return code should be **EFI_SUCCESS** and *Packet.Asb.AtaStatus* should reflect the ATA command has been executed successfully.

Number	GUID	Assertion	Test Description
			3. Call `PassThru()` with *`Port`* and *`PortMultiplierPort`* representing one available device. *`Event`* being a callback-TPL event with a notification function that updates a global vairable. By checking `EFI_ATA_PASS_THRU_PROTOCOL`.Mode.Attributes to determine whether non-blocking IO is supported. The *`Packet.Acb.AtaCommand`* is set to be the value of 'Identify Device' command 0xEC, *`Packet.Asb`* and *`Packet.InDataBuffer`* are allocated and adjusted according to `EFI_ATA_PASS_THRU_PROTOCOL`.Mode.IoAlign value, *`Packet.Timeout`* set to be 2 seconds, *`Packet.Length`* set to be block granularity, *`Packet.InTransferLength`* being 1 to indicate one block, *`Packet.Protocol`* being `EFI_ATA_PASS_THRU_PROTOCOL_PIO_DATA_IN`. If non-blocking mode is not supported, the global variable should keep unchanged. The return code should be `EFI_SUCCESS` and *`Packet.Asb.AtaStatus`* should reflect the ATA command has been executed successfully.
5.7.8.7.8	0xeb7841b9, 0x2a4a, 0x45b1, 0xa9, 0x9f, 0x67, 0x7a, 0xb4, 0xcd, 0x79, 0xa2	`EFI_ATA_PASS_THRU_PROTOCOL.PassThru – PassThru()` returns `EFI_SUCCESS`.	1. Call `PassThru()` with *`Event`* being NULL *`Packet.Length`* set to be block granularity. The return code should be `EFI_SUCCESS`.
5.7.8.7.9	0x9662da7d, 0x6f98, 0x4051, 0xb1, 0x87, 0x85, 0xb0, 0xf4, 0xb5, 0x3a, 0xf1	`EFI_ATA_PASS_THRU_PROTOCOL.PassThru – PassThru()` returns `EFI_SUCCESS`.	1. Call `PassThru()` with *`Event`* being NULL *`Packet.Length`* set to be byte granularity. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.7.8.7.10	0x5787ed6f, 0xa984, 0x4b15, 0xb2, 0xf3, 0xa0, 0xd1, 0xb8, 0xce, 0x61, 0x89	`EFI_ATA_PASS_THRU_PROTOCOL.PassThru – PassThru()` returns `EFI_SUCCESS`.	1. Call `PassThru()` with *`Event`* being a callback-TPL event with a notification function that updates a global variable. By checking `EFI_ATA_PASS_THRU_PROTOCOL`.Mode.Attributes to determine whether non-blocking IO is supported. If supported, the global variable will be updated in the event's notification function and the return code should be `EFI_SUCCESS`.
5.7.8.7.11	0x202b3252, 0x5c89, 0x41bf, 0x9b, 0x42, 0x94, 0x58, 0x56, 0xc8, 0xcc, 0x7e	`EFI_ATA_PASS_THRU_PROTOCOL.PassThru – PassThru()` returns `EFI_SUCCESS`.	1. Call `PassThru()` with *`Event`* being a callback-TPL event with a notification function that updates a global variable. *`Packet.Length`* set to block granularity. By checking `EFI_ATA_PASS_THRU_PROTOCOL.Mode.Attributes` to determine whether nonblockingIO is supported. If supported, the global variable will be updated in the event's notification function and the return code should be `EFI_SUCCESS`.

9.8.8 Mode Conformance

Number	GUID	Assertion	Test Description
5.7.8.8.1	0xdcb2c498, 0x4d12, 0x4351, 0xb4, 0xd7, 0x85, 0x33, 0x2c, 0x51, 0xd8, 0xf7	`EFI_ATA_PASS_THRU_PROTOCOL.Mode – Mode` attributes should be physical, logical or both.	1. Check Mode.Attributes to be `EFI_ATA_PASS_THRU_ATTRIBUTES_PHYSICAL , EFI_ATA_PASS_THRU_ATTRIBUTES_LOGICAL or EFI_ATA_PASS_THRU_ATTRIBUTES_PHYSICAL | EFI_ATA_PASS_THRU_ATTRIBUTES_LOGICAL`
5.7.8.8.2	0x8ccb89ab, 0x2bbe, 0x4766, 0xa9, 0x5, 0x2d, 0x1e, 0xa6, 0xb4, 0x54, 0x6b	`EFI_ATA_PASS_THRU_PROTOCOL.Mode – Mode IoAlign` should be 0, 1 or a power of 2.	Check Mode.IoAlign to be 0, 1 or a power of 2.

9.9 EFI_BLOCK_IO2_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_BLOCK_IO2_PROTOCOL Section.

9.9.1 ReadBlocksEx()

Number	GUID	Assertion	Test Description
5.7.9.1.1	0x36a2dbdb, 0x6d88, 0x4807,0xaf, 0xa5, 0x7b, 0xef, 0xc4, 0xb1, 0xfe, 0xaa	**EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx()** returns **EFI_MEDIA_CHANGED** with invalid **MediaID**	1. Sync & Async Call **ReadBlockEx()** with invalid **MediaID**. The return code should be **EFI_MEDIA_CHANGED**
5.7.9.1.2	0x45d515fd, 0xa64f, 0x47bd, 0x9a, 0x84, 0x1f, 0xe4, 0x86, 0x6a, 0x32, 0x8a	**EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx()** returns **EFI_BAD_BUFFER_SIZE** with bad *blocksize*	1.Sync & Asnyc Call **ReadblockEx()** with *BufferSize* not being a multiple of the intrinsic block size of the device. The return code should be **EFI_BAD_BUFFER_SIZE**
5.7.9.1.3	0x896937aa, 0x65ba, 0x4354, 0xab, 0xf7, 0xd8, 0x4f, 0xe8, 0x9f, 0xbc, 0x8	**EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx()** returns **EFI_INVALID_PARAMETER** with invalid **LBA** parameter	1. Sync & Async Call **ReadblockEx()** call with invalid **LBA** parameter. The return code should be **EFI_INVALID_PARAMETER**
5.7.9.1.4	0xd54c2dc4, 0x8fed, 0x4ce1, 0xac, 0x7b, 0xc6, 0x7a, 0x48, 0x4e, 0x2, 0x7	**EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx()** returns **EFI_INVALID_PARAMETER** with return data which are smaller than *BufferSize* passed in	1. Sync & Async Call **ReadblockEx()** returns data which are smaller than *BufferSize* passed in. The return code should be **EFI_INVALID_PARAMETER**
5.7.9.1.5	0xc75d447c, 0x29c5, 0x4882, 0x80, 0xc5, 0x42, 0x67, 0xe0, 0xa2, 0x5c, 0xfc	**EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx()** returns **EFI_INVALID_PARAMETER** with block alignment should be power of 2	1.Sync & Async Call **ReadblockEx()** block alignment should be power of 2. The return code should be **EFI_INVALID_PARAMETER**
5.7.9.1.6	0x1ce01e1c, 0xedde, 0x4a37, 0x96, 0x2f, 0x3a, 0x32, 0x8a, 0x54, 0xc1, 0xf5	**EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx()** returns **EFI_NO_MEDIA** when read from device without media present in the device	1.Sync & Async Call **ReadblockEx()** from device without media present in the device. The return code should be **EFI_NO_MEDIA**

Number	GUID	Assertion	Test Description
5.7.9.1.7	0x47b8309d, 0xf783, 0x4679, 0x95, 0xbb, 0x47, 0x58, 0x10, 0x7, 0x2, 0x7c	EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI_SUCCESS when Async call with proper parameter from valid media	1. Async Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI_SUCCESS
5.7.9.1.8	0x7abe441a, 0x7118, 0x4394, 0x81, 0xb8, 0xb9, 0x22, 0xa2, 0x87, 0xd2, 0x1f	EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI_SUCCESS when Async call with proper parameter from valid media	1. Async Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI_SUCCESS
5.7.9.1.9	0x3167dc14, 0xcf85, 0x4158, 0x9c, 0xec, 0x7a, 0x3, 0xdd, 0xc, 0xfd, 0xa1	EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI_SUCCESS when Async call with proper parameter from valid media. The async registered events haven't be signaled.	1. Async Call ReadblockEx() from device with proper parameter from valid media. All events should be signaled successfully.
5.7.9.1.10	0xc4726d6f, 0x148e, 0x4a06, 0xa0, 0x92, 0xd4, 0x6b, 0xa8, 0x7c, 0x16, 0x63	EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI_SUCCESS when Sync call with proper parameter from valid media	1. Sync Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI_SUCCESS
5.7.9.1.11	0x6e61c6ee, 0x2328, 0x45c5, 0x99, 0xe9, 0xcb, 0x66, 0x99, 0xcf, 0x56, 0xe	EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI_SUCCESS when Batch Async call with proper parameter from valid media	1. Batch Async Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI_SUCCESS
5.7.9.1.12	0x639fca8b, 0x394e, 0x4c1a, 0x81, 0x15, 0xc2, 0x55, 0xc6, 0xbd, 0x4b, 0x95	EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI_SUCCESS when Mixed Sync & Async call with proper parameter from valid media. Async Read Call failed	1. Mixed Sync & Async Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI_SUCCESS
5.7.9.1.13	0x6d90ad93, 0xf492, 0x4a10, 0xa0, 0xbc, 0xb5, 0x30, 0x54, 0x36, 0x28, 0x54	EFI_BLOCK_IO2_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI_SUCCESS when Mixed Sync & Async call with proper parameter from valid media.	1. Mixed Sync & Async Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI_SUCCESS

Number	GUID	Assertion	Test Description
5.7.9.1.14	0x7ce315f8, 0xb5d1, 0x4691, 0x92, 0x63, 0x66, 0x4b, 0xe0, 0x70, 0x85, 0x47	`EFI_BLOCK_IO2_PROTOCOL.ReadBlocksEx – ReadBlocksEx()` returns `EFI_SUCCESS` when Mixed Sync & Async call with proper parameter from valid media. Sync Read Call failed	1. Mixed Sync & Async Call `ReadblockEx()` from device with proper parameter from valid media. The return code should be `EFI_SUCCESS`

9.9.2 WriteBlocksEx()

Number	GUID	Assertion	Test Description
5.7.9.2.1	0xbe0e99b7, 0x62a0, 0x45ff, 0x92, 0x11, 0x55, 0xe7, 0xe4, 0xda, 0xa, 0x86	`EFI_BLOCK_IO2_PROTOCOL.WriteBlocksEx – WriteBlocksEx()` returns `EFI_MEDIA_CHANGED` with invalid `MediaID`	1. Sync & Async Call `WriteBlockEx()` with invalid `MediaID`. The return code should be `EFI_MEDIA_CHANGED`
5.7.9.2.2	0x7253b26e, 0xbb34, 0x49fa, 0x92, 0xe, 0xfa, 0x5f, 0xef, 0xa6, 0xf5, 0x5a	`EFI_BLOCK_IO2_PROTOCOL.WriteBlocksEx – WriteBlocksEx()` returns `EFI_BAD_BUFFER_SIZE` with bad *`blocksize`*	1.Sync & Asnyc Call `WriteBlocksEx()` with *`BufferSize`* not being a multiple of the intrinsic block size of the device. The return code should be `EFI_BAD_BUFFER_SIZE`
5.7.9.2.3	0x34009928, 0x8f89, 0x42d3, 0xb0, 0x20, 0x9f, 0x7f, 0x54, 0xef, 0x75, 0xe2	`EFI_BLOCK_IO2_PROTOCOL.WriteBlocksEx – WriteBlocksEx()` returns `EFI_INVALID_PARAMETER` with invalid `LBA` parameter	1. Sync & Async Call *`WriteBlocksEx()`* call with invalid `LBA` parameter. The return code should be `EFI_INVALID_PARAMETER`
5.7.9.2.4	0x3ca09c43, 0xfd3f, 0x4e88, 0x92, 0xc3, 0x97, 0xe6, 0xe1, 0x76, 0xb7, 0x3c	`EFI_BLOCK_IO2_PROTOCOL.WriteBlocksEx– WriteBlocksEx()` returns `EFI_INVALID_PARAMETER` with block alignment should be power of 2	1.Sync & Async Call `WriteBlocksEx()` block alignment should be power of 2. The return code should be `EFI_INVALID_PARAMETER`

Number	GUID	Assertion	Test Description
5.7.9.2.5	0x27c8f6f8, 0x984d, 0x49ff, 0xa6, 0xab, 0xf9, 0xc1, 0x21, 0x39, 0x2f, 0x7e	**EFI_BLOCK_IO2_PROTOCOL. WriteBlocksEx- WriteBlocksEx()** returns *EFI_INVALID_PARAMETER* with unaligned data buffer	1.Sync & Async Call **WriteBlocksEx()** unaligned data buffer. The return code should be **EFI_INVALID_PARAMETER**
5.7.9.2.6	0xf3807d6, 0x930e, 0x4ea2, 0xaa, 0xd9, 0x54, 0xc6, 0x73, 0xe9, 0xb5, 0x51	**EFI_BLOCK_IO2_PROTOCOL. WriteBlocksEx- WriteBlocksEx()** returns **EFI_NO_MEDIA** when write to device without media present in the device	1.Sync & Async Call **WriteBlocksEx()** to device without media present in the device. The return code should be **EFI_NO_MEDIA**
5.7.9.2.7	0xb9c4f106, 0x6658, 0x430f, 0x87, 0xcb, 0xe0, 0xde, 0x2c, 0xbf, 0xb9, 0x5c	**EFI_BLOCK_IO2_PROTOCOL. WriteBlocksEx- WriteBlocksEx()** when write to a read-only device	1.Sync & Async Call **WriteBlocksEx()** to read-only device. The return code should be **EFI_NO_MEDIA**
5.7.9.2.8	0x1a0cf746, 0xe5bf, 0x4b0d, 0x84, 0xf, 0x2e, 0x2b, 0xfe, 0x94, 0xdc, 0xdc	**EFI_BLOCK_IO2_PROTOCOL. WriteBlocksEx – WriteBlocksEx()** returns **EFI_SUCCESS** when Async call with proper parameter to valid media	1. Async Call **WriteBlocksEx()** to device with proper parameter to valid media. The return code should be **EFI_SUCCESS**
5.7.9.2.9	0xf8a6b15d, 0xc85b, 0x4b59, 0xbe, 0x7f, 0x2f, 0x84, 0xf9, 0x77, 0xe, 0x4a	**EFI_BLOCK_IO2_PROTOCOL. WriteBlocksEx- WriteBlocksEx()** returns **EFI_SUCCESS** when Async call with proper parameter to valid media	1. Async Call **WriteBlocksEx()** to device with proper parameter to valid media. The return code should be **EFI_SUCCESS**
5.7.9.2.10	0x964c1c44, 0xb693, 0x43ca, 0x88, 0xce, 0xb5, 0x34, 0xa6, 0x42, 0xff, 0x80	**EFI_BLOCK_IO2_PROTOCOL. WriteBlocksEx- WriteBlocksEx()** returns **EFI_SUCCESS** when Async call with proper parameter to valid media. The async registered events haven't be signaled.	1. Async Call **WriteBlocksEx()** to device with proper parameter to valid media. All events should be signaled successfully.
5.7.9.2.11	0xad588be4, 0x138f, 0x4874, 0x92, 0xf, 0xef, 0xa1, 0xd3, 0x79, 0xe5, 0x17	**EFI_BLOCK_IO2_PROTOCOL. WriteBlocksEx- WriteBlocksEx()** returns **EFI_SUCCESS** when Sync call with proper parameter to valid media	1. Sync Call **WriteBlocksEx()** to device with proper parameter to valid media. The return code should be **EFI_SUCCESS**

Number	GUID	Assertion	Test Description
5.7.9.2.12	0x9e70ff56, 0x7e0e, 0x404f, 0xab, 0x10, 0x7f, 0x6a, 0x1e, 0x1f, 0xbb, 0xf7	EFI_BLOCK_IO2_PROTOCOL. WriteBlocksEx-WriteBlocksEx() returns EFI_SUCCESS when Batch Async call with proper parameter to valid media	1. Batch Async Call WriteBlocksEx() to device with proper parameter to valid media. The return code should be EFI_SUCCESS
5.7.9.2.13	0x6d41db68, 0xffe3, 0x4676, 0x89, 0xba, 0xe4, 0xc8, 0xdb, 0xc6, 0x4f, 0xcc	EFI_BLOCK_IO2_PROTOCOL. WriteBlocksEx-WriteBlocksEx() returns EFI_SUCCESS when Mixed Sync & Async call with proper parameter to valid media. Async Read Call failed	1. Mixed Sync & Async Call WriteBlocksEx() to device with proper parameter to valid media. The return code should be EFI_SUCCESS
5.7.9.2.14	0xe532b760, 0xd561, 0x43be, 0xa5, 0x51, 0x62, 0xb5, 0x63, 0x16, 0x9f, 0xbc	EFI_BLOCK_IO2_PROTOCOL. WriteBlocksEx-WriteBlocksEx() returns EFI_SUCCESS when Mixed Sync & Async call with proper parameter to valid media.	1. Mixed Sync & Async Call WriteBlocksEx() to device with proper parameter to valid media. The return code should be EFI_SUCCESS
5.7.9.2.15	0x11a6bb4a, 0xa943, 0x4006, 0xbc, 0xb0, 0x57, 0x6c, 0xb8, 0x68, 0xae, 0x95	EFI_BLOCK_IO2_PROTOCOL. WriteBlocksEx-WriteBlocksEx() returns EFI_SUCCESS when Mixed Sync & Async call with proper parameter to valid media. Sync Read Call failed	1. Mixed Sync & Async Call WriteBlocksEx() to device with proper parameter to valid media. The return code should be EFI_SUCCESS

9.9.3 FlashBlocksEx()

Number	GUID	Assertion	Test Description
5.7.9.3.1	0x457168d, 0x1ded, 0x4c01, 0xb7, 0x84, 0xa7, 0xa9, 0xd6, 0xaa, 0xb, 0x81	EFI_BLOCK_IO2_PROTOCOL. FlashBlocksEx-FlashBlocksEx() returns EFI_NO_MEDIA with a device with no media	1. Sync & Async Call FlashBlocksEx() with no Media on device The return code should be EFI_NO_MEDIA
5.7.9.3.2	0x6a1de6c8, 0xe02b, 0x4a50, 0x80, 0x6d, 0x9a, 0x51, 0x5c, 0xc1, 0xe4, 0xe3	EFI_BLOCK_IO2_PROTOCOL. FlashBlocksEx-FlashBlocksEx() returns EFI_WRITE_PROTECTED with a read-only device with media	1. Sync & Async Call FlashBlocksEx() with a read-only media on device The return code should be EFI_WRITE_PROTECTED

5.7.9.3.3	0xc97de60f, 0x87cf, 0x45b9, 0x98, 0x9b, 0x8, 0x9d, 0x3, 0xe0, 0xf3, 0xf6	**EFI_BLOCK_IO2_PROTOCOL. FlashBlocksEx- FlashBlocksEx()** returns **EFI_SUCCESS** with a right device with media & all event signaled should be signaled	1. Async Call **FlashBlocksEx()** with a media on a right device The return code should be **EFI_SUCCESS**

9.9.4 Media Info Check

Number	GUID	Assertion	Test Description
5.7.9.4.1	0x8251405e, 0xe716, 0x4ecd, 0x83, 0x55, 0xc9, 0xf5, 0x60, 0x4b, 0xf2, 0x4d	**EFI_BLOCK_IO2_PROTOCOL. Media-LogicalBlocksPerPhysicalBlock** should be 0 when **LogicalPartition** is **TRUE** and **Revision** is greater than or equal to **EFI_BLOCK_IO_PROTOCOL_REVISION2.**	**LogicalBlocksPerPhysicalBlock** should be 0 when **LogicalPartition** is **TRUE** and **Revision** is greater than or equal to **EFI_BLOCK_IO_PROTOCOL_REVISION2**.
5.7.9.4.2	0x6739b945, 0x2498, 0x4a1c, 0x87, 0xb0, 0x85, 0xa4, 0xbe, 0xf6, 0x53, 0x7c	**EFI_BLOCK_IO2_PROTOCOL. Media-OptimalTransferLengthGranularity** should be 0 when **LogicalPartition** is **TRUE** and **Revision** is greater than or equal to **EFI_BLOCK_IO_PROTOCOL_REVISION3**.	**OptimalTransferLengthGranularity** should be 0 when **LogicalPartition** is **TRUE** and **Revision** is greater than or equal to **EFI_BLOCK_IO_PROTOCOL_REVISION3**.

9.10 EFI_STORAGE_SECURITY_COMMAND_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_STORAGE_SECURITY_COMMAND_PROTOCOL Section.

9.10.1 ReceiveData()

Number	GUID	Assertion	Test Description
5.7.10.1.1	0x35749acf, 0xeed8, 0x4230, 0xbc, 0x18, 0xde, 0x1f, 0x8b, 0x7c, 0xfa, 0xef	EFI_STORAGE_SECURITY_ COMMAND.ReceiveData – ReceiveData() should not return EFI ERROR. When PayloadBufferSize is too small	Call ReceiveData () with PayloadBufferSize =10 & TCG command 0 to return security protocol info The return status should not be EFI Error Status
5.7.10.1.2	0x8e742768, 0x229a, 0x4aaa, 0xb5, 0x9d, 0xc9, 0xb2, 0x6e, 0x32, 0x44, 0x58	EFI_STORAGE_SECURITY_ COMMAND.ReceiveData – ReceiveData() should return EFI_MEDIA_CHANGED. When MediaID is not correct	Call ReceiveData () with Wrong MediaID & TCG command 0 to return security protocol info The return status should be EFI_MEDIA_CHANGED
5.7.10.1.3	0x2fe7a174, 0xa8a1, 0x45b3, 0x91, 0x5c, 0x1e, 0xd, 0x59, 0x13, 0x17, 0xa6	EFI_STORAGE_SECURITY_ COMMAND.ReceiveData – ReceiveData() should return EFI_INVALID_PARAMETER. When PayloadBuffer is NULL	Call ReceiveData () with NULL PayloadBuffer & TCG command 0 to return security protocol info The return status should be EFI_INVALID_PARAMETER.
5.7.10.1.4	0xa55d41c7, 0x0ca4, 0x4ab1, 0xaa, 0x7b, 0x82, 0xee, 0x74, 0x30, 0xcf, 0x9d	EFI_STORAGE_SECURITY_ COMMAND.ReceiveData – ReceiveData() should return EFI_INVALID_PARAMETER. When PayloadTransferSize is not NULL	Call ReceiveData () with NULL PayloadTransferSize & TCG command 0 to return security protocol info The return status should be EFI_INVALID_PARAMETER.
5.7.10.1.5	0xcc0223b7, 0xf088, 0x4ea1, 0xa6, 0xcb, 0x73, 0x93, 0x8, 0x4e, 0x9f, 0xf2	EFI_STORAGE_SECURITY_ COMMAND.ReceiveData – ReceiveData() should return EFI_NO_MEDIA. When There is no media present	Call ReceiveData () with TCG command 0 to return security protocol info on a no media device The return status should be EFI_NO_MEDIA.

9.10.2 SendData()

Number	GUID	Assertion	Test Description
5.7.10.2.1	0x2e6fddd2, 0xce3b, 0x49fb, 0xa1, 0xd4, 0x43, 0xa2, 0x99, 0xf7, 0xec, 0xc2	EFI_STORAGE_SECURITY_ COMMAND.SendData – SendData() should return EFI_MEDIA_CHANGED. When MediaID is not correct	Call SendData () with Wrong MediaID & TCG command 0 to return security protocol info The return status should be EFI_MEDIA_CHANGED

Number	GUID	Assertion	Test Description
5.7.10.2.2	0x2323be1a, 0xf73a, 0x46d5, 0xa2, 0x24, 0xbf, 0x9a, 0x7f, 0x6a, 0x53, 0x96	EFI_STORAGE_SECURITY_COMMAND.SendData – SendData() should return EFI_INVALID_PARAMETER. When PayloadBuffer is NULL	Call ReceiveData () with NULL PayloadBuffer& TCG command 0 to return security protocol info The return status should be EFI_INVALID_PARAMETER
5.7.10.2.3	0x68acfb97, 0xcec1, 0x4015, 0xac, 0xca, 0x64, 0xad, 0x8c, 0xde, 0x5e, 0x71	EFI_STORAGE_SECURITY_COMMAND.SendData – SendData() should return EFI_NO_MEDIA. When There is no media present	Call SendData () with TCG command 0 to return security protocol info on a no media device The return status should be EFI_NO_MEDIA.

9.11 EFI_DISK_IO2_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_DISK_IO2_PROTOCOL Section.

9.11.1 Cancel()

Number	GUID	Assertion	Test Description
5.7.11.1.1	0xd8cc30e4, 0xaac4, 0x415b, 0xb0, 0x12, 0x27, 0x13, 0x29, 0x8d, 0x0f, 0xfa	EFI_DISK_IO2_PROTOCOL.Cancel - Cancel () returns EFI_SUCCESS.	Call Cancel () ,the return status should be EFI_SUCCESS.

9.11.2 ReadDiskEx()

Number	GUID	Assertion	Test Description
5.7.11.2.1	0x9b457a7a, 0x9f63, 0x4627, 0x80, 0x6a, 0xfe, 0x39, 0x30, 0x9e, 0x29, 0xec	EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI_SUCCESS when Async call with proper parameter.	Async Call ReadDiskEx() with proper parameter, the return status should be EFI_SUCCESS. If possible, the data should be same as the result from ReadDisk().

Number	GUID	Assertion	Test Description
5.7.11.2.2	0xac2b9d8c, 0xc35c, 0x4788, 0xa9, 0xcf, 0xb2, 0x93, 0x2e, 0x7c, 0xe2, 0x85	EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx() returns EFI_SUCCESS when Async call with proper parameter.	Async Call ReadDiskEx() with proper parameter, the return status should be EFI_SUCCESS. The ReadFaillist should be empty.
5.7.11.2.3	0xe3aa41fc, 0x1275, 0x4d74, 0x80, 0x97, 0x85, 0x27, 0x5c, 0xc6, 0x22, 0x5c	EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx() returns EFI_SUCCESS when Async call with proper parameter.	Async Call ReadDiskEx() with proper parameter, the return status should be EFI_SUCCESS. The ReadExecuteList should be empty.
5.7.11.2.4	0x979b7b0d, 0x22bb, 0x4507, 0x9d, 0x69, 0x21, 0xd1, 0xb8, 0x50, 0x6c, 0x9e	EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx() returns EFI_SUCCESS when Sync call with proper parameter.	Sync Call ReadDiskEx() with proper parameter, the return status should be EFI_SUCCESS. If possible, the data should be same as the result from ReadDisk().
5.7.11.2.5	0xe09f04a1, 0x00ee, 0x4a48, 0x90, 0x5f, 0x2d, 0x23, 0xd2, 0xa6, 0x71, 0x2e	EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx() returns EFI_SUCCESS when Batch Async call with proper parameter.	Batch Async Call ReadDiskEx() with proper parameter, the return status should be EFI_SUCCESS. If possible, the data should be same as the result from ReadDisk().
5.7.11.2.6	0xe6172d46, 0x3648, 0x4677, 0x8d, 0xde, 0xb1, 0xfd, 0x10, 0x6a, 0xe5, 0xe6	EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx() returns EFI_SUCCESS when Mixed Sync & Async call with proper parameter.	Mixed Sync & Async Call ReadDiskEx() with proper parameter, the return status should be EFI_SUCCESS
5.7.11.2.7	0x8048d7d8, 0x3e99, 0x4a32, 0x87, 0xac, 0x1f, 0x61, 0x3c, 0xf9, 0x13, 0x9c	EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx() returns EFI_SUCCESS when Mixed Sync & Async call with proper parameter. Async call failed.	Mixed Sync & Async Call ReadDiskEx() with proper parameter, the return status should be EFI_SUCCESS. The MixReadFaillist should be empty.

Number	GUID	Assertion	Test Description
5.7.11.2.8	0x6e6179d0, 0xfbe3, 0x4ef8, 0xb5, 0xed, 0x1b, 0xb1, 0xc0, 0xe7, 0x69, 0xb0	`EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx() returns EFI_SUCCESS` when Mixed Sync & Async call with proper parameter. Async Read Call failed	Mixed Sync & Async Call `ReadDiskEx()` with proper parameter, the return status should be `EFI_SUCCESS`. The MixReadExecutelist should be empty.
5.7.11.2.9	0x870cf02a, 0xb573, 0x40d6,0x91, 0x70, 0x8d, 0x83, 0x3d, 0x45, 0xf4, 0xc3	`EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx() returns EFI_SUCCESS when Mixed Sync & Async call with proper parameter.` Sync Read Call success.	Mixed Sync & Async Call `ReadDiskEx() with proper parameter, the return status should be EFI_SUCCESS`. The data in SyncReadList should be same as the output from Async read.
5.7.11.2.10	0xb491381b, 0xf841, 0x44fb, 0x94, 0x62, 0xd5, 0x5c, 0x30, 0xde, 0xb1, 0x85	`EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx()` returns `EFI_SUCCESS` when Mixed Sync & Async call with proper parameter. Sync Read Call failed.	Mixed Sync & Async Call `ReadDiskEx()` with proper parameter, the return status should be `EFI_SUCCESS`. The SyncReadFailList should be empty.
5.7.11.2.11	0xfb7b94af, 0x0368, 0x4a66, 0xaf, 0x52, 0x31, 0x00, 0x8a, 0xf1, 0xd5, 0xc7	`EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx()` returns `EFI_MEDIA_CHANGED` when Sync & Async Read disk with MediaId not being the id for the current media in the device.	Sync & Async call `ReadDiskEx()` with MediaId not being the id for the current media in the device, the return status should be `EFI_MEDIA_CHANGED.`
5.7.11.2.12	0x2e9486a6, 0x51c3, 0x4da7,0xa4, 0x81, 0xab, 0xae, 0x72, 0x35, 0xe9, 0x43	`EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx()` returns `EFI_MEDIA_CHANGED` when Sync & Async Read disk with invalid offset.	Sync & Async call `ReadDiskEx()` with invalid offset, the return status should be `EFI_INVALID_PARAMETERS.`

Number	GUID	Assertion	Test Description
5.7.11.2.13	0x8851b5ee, 0x51ea, 0x4241,0xb8, 0x52, 0x40, 0xbc, 0x49, 0x1c, 0x62, 0x31	`EFI_DISK_IO2_PROTOCOL.ReadDiskEx - ReadDiskEx ()` returns `EFI_MEDIA_CHANGED` when Sync & Async Read disk from device without media present in the device.	Sync & Async call `ReadDiskEx()` from device without media present in the device, the return status should be `EFI_NO_MEDIA`.

9.11.3 WriteDiskEx()

Number	GUID	Assertion	Test Description
5.7.11.3.1	0x3a74e001, 0x817a, 0x45b2, 0xb3, 0x12, 0x3d, 0x12, 0xbb, 0x36, 0x41, 0xc0	`EFI_DISK_IO2_PROTOCOL.WriteDiskEx - WriteDiskEx ()` returns `EFI_SUCCESS` when Async call with proper parameter.	Async Call `WriteDiskEx ()` with proper parameter, the return status should be `EFI_SUCCESS`. If possible, the date read from the same address should be same as the date written in.
5.7.11.3.2	0xeeb0a39d, 0x6c51, 0x4152, 0xb5, 0x74, 0xa6, 0xec, 0xda, 0x4c, 0xdf, 0x80	`EFI_DISK_IO2_PROTOCOL.WriteDiskEx - WriteDiskEx ()` returns `EFI_SUCCESS` when Async call with proper parameter. Async call failed.	Async Call `WriteDiskEx ()` with proper parameter, the return status should be `EFI_SUCCESS`. The WriteFailList should be empty.
5.7.11.3.3	0x70b3b8f6, 0x91cf, 0x47a5, 0xbc, 0x12, 0x09, 0xe7, 0xb8, 0x27, 0x5d, 0x41	`EFI_DISK_IO2_PROTOCOL.WriteDiskEx - WriteDiskEx ()` returns `EFI_SUCCESS` when Async call with proper parameter.	Async Call `WriteDiskEx ()` with proper parameter, the return status should be `EFI_SUCCESS`. The WriteExecuteList should be empty.
5.7.11.3.4	0x5107009f, 0xe732, 0x45ad, 0xbe, 0x8d, 0xe6, 0x79, 0xb8, 0x76, 0x6a, 0xf3	`EFI_DISK_IO2_PROTOCOL.WriteDiskEx - WriteDiskEx ()` returns `EFI_SUCCESS` when Sync call with proper parameter.	Sync Call `WriteDiskEx ()` with proper parameter, the return status should be `EFI_SUCCESS`. If possible, the date read from the same address should be same as the date written in.

Number	GUID	Assertion	Test Description
5.7.11.3.5	0x72023591, 0x1ad7, 0x468c, 0xb4, 0x75, 0x31, 0xa4, 0x1b, 0x9d, 0x0c, 0x78	EFI_DISK_IO2_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI_SUCCESS when Batch Async call with proper parameter.	Batch Async Call WriteDiskEx () with proper parameter, the return status should be EFI_SUCCESS.
5.7.11.3.6	0x75a4a0e7, 0x5d73, 0x4809, 0xa4, 0x0e, 0x20, 0x3a, 0x0f, 0xcf, 0x09, 0x94	EFI_DISK_IO2_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI_MEDIA_CHANGED when Sync & Async Write disk with MediaId not being the id for the current media in the device.	Sync & Async Call WriteDiskEx () with MediaId not being the id for the current media in the device, the return status should be EFI_MEDIA_CHANGED.
5.7.11.3.7	0xe0540275, 0x032e, 0x4507, 0xb3, 0x03, 0x01, 0xc5, 0xbf, 0x9c, 0xe1, 0x56	EFI_DISK_IO2_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI_INVALID_PARAMETERS when Sync & Async Write disk with invalid Offset & BufferSize.	Sync & Async Call WriteDiskEx () with invalid Offset & BufferSize, the return status should be EFI_INVALID_PARAMETERS.
5.7.11.3.8	0x8688e7ad, 0x4f3e, 0x432e, 0xaf, 0x3b, 0x03, 0x93, 0x6b, 0xe3, 0xe5, 0xa6	EFI_DISK_IO2_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI_WRITE_PROTECTED when Sync & Async Write disk to a write-protected device.	Sync & Async Call WriteDiskEx () to a write-protected device, the return status should be EFI_WRITE_PROTECTED.

Number	GUID	Assertion	Test Description
5.7.11.3.9	0xee9fa363, 0x2009, 0x429f, 0x92, 0x0b, 0x60, 0x3b, 0xc4, 0xdc, 0x6e, 0x64	EFI_DISK_IO2_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI_NO_MEDIA when Sync & Async Write disk without media present in the device.	Sync & Async Call WriteDiskEx () without media present in the device, the return status should be EFI_NO_MEDIA.

9.11.4 FlushDiskEx()

Number	GUID	Assertion	Test Description
5.7.11.4.1	0x681169b1, 0xb5eb, 0x4cb0, 0x91, 0xc6, 0xfd, 0x2d, 0x9f, 0xe8, 0x24, 0x50	EFI_DISK_IO2_PROTOCOL.FlushDiskEx - FlushDiskEx () returns EFI_SUCCESS when Async call with proper parameter.	Async Call FlushDiskEx () with proper parameter, the return status should be EFI_SUCCESS.
5.7.11.4.2	0x2cf71e16, 0xa399, 0x4a8c, 0xa2, 0xf8, 0x09, 0x5c, 0x9d, 0xcd, 0x25, 0xbd	EFI_DISK_IO2_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI_SUCCESS when Async call with proper parameter. Async call failed.	Async Call FlushDiskEx () with proper parameter, the return status should be EFI_SUCCESS. The FlushFailList should be empty.
5.7.11.4.3	0x48a3fb9b, 0xd65f, 0x44fe, 0x94, 0x29, 0x14, 0xa6, 0x7b, 0x94, 0x0d, 0xda	EFI_DISK_IO2_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI_SUCCESS when Async call with proper parameter.	Async Call FlushDiskEx () with proper parameter, the return status should be EFI_SUCCESS. The FlushExecuteList should be empty.
5.7.11.4.4	0x0003470c, 0x15a7, 0x468a, 0xa2, 0xb1, 0xd1, 0x03, 0x8c, 0x81, 0x70, 0xb5	EFI_DISK_IO2_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI_SUCCESS when Sync call with proper parameter.	Sync Call FlushDiskEx () with proper parameter, the return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.7.11.4.5	0x14525c4c, 0x213e, 0x4985, 0xa6, 0x42, 0x75, 0x6f, 0x0a, 0x8b, 0x2e, 0xf1	EFI_DISK_IO2_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI_SUCCESS when Batch Async call with proper parameter.	Batch Async call FlushDiskEx () with proper parameter, the return status should be EFI_SUCCESS.
5.7.11.4.6	0x2f6c3f4b, 0x5e09, 0x4ada, 0xb0, 0xea, 0xb2, 0x99, 0xe1, 0xf3, 0xd3, 0x50	EFI_DISK_IO2_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI_MEDIA_CHANGED when Sync & Async flush disk with MediaId not being the id for the current media in the device.	Sync & Async call FlushDiskEx () with MediaId not being the id for the current media in the device, the return status should be EFI_MEDIA_CHANGED.
5.7.11.4.7	0x5243f002, 0x6d2e, 0x4267, 0xa5, 0x7b, 0x1f, 0xff, 0xb0, 0x98, 0x8c, 0x5f	EFI_DISK_IO2_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI_INVALID_PARAMETERS when Sync & Async flush disk with invalid Offset.	Sync & Async call FlushDiskEx () with invalid Offset, the return status should be EFI_INVALID_PARAMETERS.
5.7.11.4.8	0x0c0c5c6d, 0xd082, 0x4b2b, 0x9e, 0x6b, 0x8f, 0xaa, 0x5d, 0x72, 0xe8, 0xd4	EFI_DISK_IO2_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI_WRITE_PROTECTED when Sync & Async flush disk to a write-protected device.	Sync & Async call FlushDiskEx () to a write-protected device, the return status should be EFI_WRITE_PROTECTED.

Number	GUID	Assertion	Test Description
5.7.11.4.9	0x28882b47, 0x5bb8, 0x4d8c, 0x84, 0x5c, 0x33, 0xf7, 0x66, 0x32, 0x44, 0x25	**EFI_DISK_IO2_PROTOCOL.FlushDiskEx - FlushDiskEx ()** returns **EFI_NO_MEDIA** when Sync & Async flush disk without media present in the device.	Sync & Async call **FlushDiskEx ()** without media present in the device, the return status should be **EFI_NO_MEDIA.**

9.11.5 EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL Section.

9.11.6 PassThru()

Number	GUID	Assertion	Test Description
5.7.12.1.1	0x85ee4a17, 0xd2a1, 0x4857, 0x9d, 0xa1, 0xc, 0xa8, 0x2d, 0x45, 0x70, 0x19	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. PassThru() - PassThru() returns EFI_INVALID_PARAMETER when TransferBuffer does not meet the alignment requirement specified by the IoAlign field of the EFI_NVM_EXPRESS_PASS_THRU_MODE.	1. Call PassThru() when TransferBuffer does not meet the alignment requirement specified by the IoAlign field of the EFI_NVM_EXPRESS_PASS_THRU_MODE , the return status should be EFI_INVALID_PARAMETER.
5.7.12.1.2	0xd6366b2c, 0x437c, 0x48c5, 0x9b, 0xcd, 0x9f, 0x17, 0x6d, 0xf8, 0x61, 0x93	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. PassThru() - PassThru() returns EFI_INVALID_PARAMETER when QueueType is not 0 (Admin Submission Queue) or 1 (I/O Submission Queue).	1. Call PassThru() when QueueType is not 0 (Admin Submission Queue) or 1 (I/O Submission Queue), the return status should be EFI_INVALID_PARAMETER.
5.7.12.1.3	0xeed32c13, 0x9232, 0x48aa, 0xb0, 0x44, 0xc9, 0xdc, 0x18, 0x47, 0x77, 0xc0	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. Mode – Mode check returns Failure with neither EFI_NVM_EXPRESS_PASS_THRU_ATTRIBUTES_LOGICAL nor EFI_NVM_EXPRESS_PASS_THRU_ATTRIBUTES_PHYSICAL set is an illegal configuration.	1. An EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL with neither EFI_NVM_EXPRESS_PASS_THRU_ATTRIBUTES_LOGICAL nor EFI_NVM_EXPRESS_PASS_THRU_ATTRIBUTES_PHYSICAL set in Mode.Attributes is an illegal configuration.
5.7.12.1.4	0xe22b3a66, 0xb9c8, 0x479a, 0x9c, 0x80, 0x9, 0xa4, 0x44, 0x9c, 0xaf, 0x2e	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. Mode – Mode check returns Failure When Mode.IoAlign is neither the power of 2 nor 0.	1. Mode.IoAlign is neither the power of 2 nor 0.
5.7.12.1.5	0x976d1926, 0x862, 0x4f41, 0x84, 0x42, 0xa5, 0x23, 0xf3, 0xc7, 0x9e, 0x4b	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. PassThru() - PassThru() returns EFI_SUCCESS with the valid Identify Command and NULL Event.	1. Call PassThru() with the valid Identify Command and NULL Event, the return status should be EFI_INVALID_PARAMETER.
5.7.12.1.6	0x9c88d95c, 0x228a, 0x48e0, 0xbd, 0x17, 0xd1, 0x87, 0x31, 0x9, 0xf1, 0xfc	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. PassThru() - PassThru() returns EFI_SUCCESS with the valid Identify Command and Event.	1. Call PassThru() with the valid Identify Command and Event, the return status should be EFI_INVALID_PARAMETER and the corresponding notification function should be signaled if the NON_BLOCKIO is supported.

9.11.7 GetNextNamespace()

Number	GUID	Assertion	Test Description
5.7.12.2.1	0xd516e8e4, 0x2d06, 0x40b4, 0xb5, 0x36, 0x65, 0xf0, 0x1c, 0x59, 0x28, 0xf9	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. GetNextNamespace() - GetNextNamespace() returns EFI_INVALID_PARAMETER with invalid NameSpaceId.	1. Call GetNextNamespace() with invalid NameSpaceId, the return status should be EFI_INVALID_PARAMETER.
5.7.12.2.2	0x6f1c4115, 0x1ef7, 0x4ae9, 0x8e, 0x9, 0x85, 0xce, 0xe5, 0x4a, 0xd9, 0xb6	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. GetNextNamespace() - GetNextNamespace() returns EFI_NOT_FOUND when no more namespaces are defined on this controller.	1. Call GetNextNamespace()when no more namespaces are defined on this controller, the return status should be EFI_NOT_FOUND.

9.11.8 BuildDevicePath()

Number	GUID	Assertion	Test Description
5.7.12.3.1	0x6f45fc1, 0xa9cd, 0x4889, 0x88, 0x1d, 0x5e, 0x34, 0xb8, 0x12, 0xfa, 0x3d	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. BuildDevicePath() - BuildDevicePath() returns EFI_INVALID_PARAMETER with NULL DevicePath.	1. Call BuildDevicePath() with NULL DevicePath, the return status should be EFI_INVALID_PARAMETER.
5.7.12.3.2	0x2b9446e8, 0xea00, 0x49ee, 0x97, 0x2d, 0xcf, 0x2a, 0xa4, 0x9e, 0xa, 0xd3	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. BuildDevicePath() - BuildDevicePath() returns EFI_NOT_FOUND with invalid NameSpaceId.	1. Call BuildDevicePath() with invalid NameSpaceId, the return status should be EFI_NOT_FOUND.
5.7.12.3.3	0xa11dede9, 0xe13d, 0x4096, 0x90, 0xc8, 0xa6, 0x2e, 0x16, 0xc5, 0x76, 0xaf	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. BuildDevicePath() - BuildDevicePath() returns EFI_SUCCESS with valid NameSpaceId.	1. Call GetNextNamespace() with valid NameSpaceId, the return status should be EFI_SUCCESS. The member NameSpaceId in the DevicePath should be same as the NameSpaceId.

9.11.9 GetNamespace()

Number	GUID	Assertion	Test Description
5.7.12.4.1	0xbefdcd7a, 0xf32d, 0x4423, 0x87, 0x7e, 0xf8, 0xc4, 0x56, 0x38, 0xd6, 0xd8	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. GetNamespace() - GetNamespace() returns EFI_INVALID_PARAMETER with NULL NamespaceId.	1. Call GetNamespace() with NULL NamespaceId, the return status should be EFI_INVALID_PARAMETER.
5.7.12.4.2	0x38ae6f88, 0x2cf9, 0x497b, 0x94, 0x59, 0x7c, 0xaa, 0x34, 0xb7, 0xed, 0x7f	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. GetNamespace() - GetNamespace() returns EFI_INVALID_PARAMETER with NULL DevicePath.	1. Call GetNamespace() with NULL DevicePath, the return status should be EFI_INVALID_PARAMETER.
5.7.12.4.3	0x365f8fba, 0x3314, 0x4502, 0x89, 0x3e, 0x8e, 0x63, 0xc1, 0xda, 0xfe, 0xbc	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. GetNamespace() - GetNamespace() returns EFI_UNSUPPORTED with unsupported device path node.	1. Call GetNamespace() with unsupported device path node, the return status should be EFI_UNSUPPORTED.
5.7.12.4.4	0xe864012d, 0x12b0, 0x4467, 0xa9, 0x7b, 0x5f, 0x72, 0xb4, 0xa9, 0x50, 0x27	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. GetNamespace() - GetNamespace() returns EFI_NOT_FOUND with NVME device path node, but the translation from DevicePath to namespace ID failed.	1. Call GetNamespace() with NVME device path node, but translation from DevicePath to namespace ID failed, the return status should be EFI_NOT_FOUND.
5.7.12.4.5	0xc72a5f58, 0x742a, 0x4c7f, 0xbc, 0xc1, 0x35, 0xf9, 0xd0, 0x31, 0x32, 0xd	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. GetNamespace() - GetNamespace() returns EFI_SUCCESS with valid parameters.	1. Call GetNamespace() with valid parameters, the return status should be EFI_SUCCESS.

9.12 EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL Section.

9.12.1 PassThru()

Number	GUID	Assertion	Test Description
5.7.12.1.1	0x85ee4a17, 0xd2a1, 0x4857, 0x9d, 0xa1, 0xc, 0xa8, 0x2d, 0x45, 0x70, 0x19	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. PassThru() - PassThru() returns EFI_INVALID_PARAMETER when TransferBuffer does not meet the alignment requirement specified by the IoAlign field of the EFI_NVM_EXPRESS_PASS_THRU_MODE.	1. Call PassThru() when TransferBuffer does not meet the alignment requirement specified by the IoAlign field of the EFI_NVM_EXPRESS_PASS_THRU_MODE, the return status should be EFI_INVALID_PARAMETER.
5.7.12.1.2	0xd6366b2c, 0x437c, 0x48c5, 0x9b, 0xcd, 0x9f, 0x17, 0x6d, 0xf8, 0x61, 0x93	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. PassThru() - PassThru() returns EFI_INVALID_PARAMETER when QueueType is not 0 (Admin Submission Queue) or 1 (I/O Submission Queue).	1. Call PassThru() when QueueType is not 0 (Admin Submission Queue) or 1 (I/O Submission Queue), the return status should be EFI_INVALID_PARAMETER.
5.7.12.1.3	0xeed32c13, 0x9232, 0x48aa, 0xb0, 0x44, 0xc9, 0xdc, 0x18, 0x47, 0x77, 0xc0	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. Mode – Mode check returns Failure with neither EFI_NVM_EXPRESS_PASS_THRU_ATTRIBUTES_LOGICAL nor EFI_NVM_EXPRESS_PASS_THRU_ATTRIBUTES_PHYSICAL set is an illegal configuration.	1. An EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL with neither EFI_NVM_EXPRESS_PASS_THRU_ATTRIBUTES_LOGICAL nor EFI_NVM_EXPRESS_PASS_THRU_ATTRIBUTES_PHYSICAL set in Mode.Attributes is an illegal configuration.
5.7.12.1.4	0xe22b3a66, 0xb9c8, 0x479a, 0x9c, 0x80, 0x9, 0xa4, 0x44, 0x9c, 0xaf, 0x2e	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. Mode – Mode check returns Failure When Mode.IoAlign is neither the power of 2 nor 0.	1. Mode.IoAlign is neither the power of 2 nor 0.

Number	GUID	Assertion	Test Description
5.7.12.1.5	0x976d1926, 0x862, 0x4f41, 0x84, 0x42, 0xa5, 0x23, 0xf3, 0xc7, 0x9e, 0x4b	EFI_NVM_EXPRESS_PASS _THRU_PROTOCOL. PassThru() - PassThru() returns EFI_SUCCESS with the valid Identify Command and NULL Event.	1. Call PassThru() with the valid Identify Command and NULL Event, the return status should be EFI_INVALID_PARAMETER.
5.7.12.1.6	0x9c88d95c, 0x228a, 0x48e0, 0xbd, 0x17, 0xd1, 0x87, 0x31, 0x9, 0xf1, 0xfc	EFI_NVM_EXPRESS_PASS _THRU_PROTOCOL. PassThru() - PassThru() returns EFI_SUCCESS with the valid Identify Command and Event.	1. Call PassThru() with the valid Identify Command and Event, the return status should be EFI_INVALID_PARAMETER and the corresponding notification function should be signaled if the NON_BLOCKIO is supported.

9.12.2 GetNextNamespace()

Number	GUID	Assertion	Test Description
5.7.12.2.1	0xd516e8e4, 0x2d06, 0x40b4, 0xb5, 0x36, 0x65, 0xf0, 0x1c, 0x59, 0x28, 0xf9	EFI_NVM_EXPRESS_PASS_ THRU_PROTOCOL. GetNextNamespace() - GetNextNamespace() returns EFI_INVALID_PARAMETER with invalid NameSpaceId.	1. Call GetNextNamespace() with invalid NameSpaceId, the return status should be EFI_INVALID_PARAMETER.
5.7.12.2.2	0x6f1c4115, 0x1ef7, 0x4ae9, 0x8e, 0x9, 0x85, 0xce, 0xe5, 0x4a, 0xd9, 0xb6	EFI_NVM_EXPRESS_PASS_ THRU_PROTOCOL. GetNextNamespace() - GetNextNamespace() returns EFI_NOT_FOUND when no more namespaces are defined on this controller.	1. Call GetNextNamespace()when no more namespaces are defined on this controller, the return status should be EFI_NOT_FOUND.

9.12.3 BuildDevicePath()

Number	GUID	Assertion	Test Description
5.7.12.3.1	0x6f45fc1, 0xa9cd, 0x4889, 0x88, 0x1d, 0x5e, 0x34, 0xb8, 0x12, 0xfa, 0x3d	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. BuildDevicePath() - BuildDevicePath() returns EFI_INVALID_PARAMETER with NULL DevicePath.	1. Call BuildDevicePath() with NULL DevicePath, the return status should be EFI_INVALID_PARAMETER.
5.7.12.3.2	0x2b9446e8, 0xea00, 0x49ee, 0x97, 0x2d, 0xcf, 0x2a, 0xa4, 0x9e, 0xa, 0xd3	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. BuildDevicePath() - BuildDevicePath() returns EFI_NOT_FOUND with invalid NameSpaceId.	1. Call BuildDevicePath() with invalid NameSpaceId, the return status should be EFI_NOT_FOUND.
5.7.12.3.3	0xa11dede9, 0xe13d, 0x4096, 0x90, 0xc8, 0xa6, 0x2e, 0x16, 0xc5, 0x76, 0xaf	EFI_NVM_EXPRESS_PASS_THRU_PROTOCOL. BuildDevicePath() - BuildDevicePath() returns EFI_SUCCESS with valid NameSpaceId.	1. Call GetNextNamespace() with valid NameSpaceId, the return status should be EFI_SUCCESS. The member NameSpaceId in the DevicePath should be same as the NameSpaceId.

9.12.4 GetNamespace()

Number	GUID	Assertion	Test Description
5.7.12.4.1	0xbefdcd7a, 0xf32d, 0x4423, 0x87, 0x7e, 0xf8, 0xc4, 0x56, 0x38, 0xd6, 0xd8	EFI_NVM_EXPRESS_PASS _THRU_PROTOCOL. GetNamespace() - GetNamespace() returns EFI_INVALID_PARAMETER with NULL NamespaceId.	1. Call GetNamespace() with NULL NamespaceId, the return status should be EFI_INVALID_PARAMETER.
5.7.12.4.2	0x38ae6f88, 0x2cf9, 0x497b, 0x94, 0x59, 0x7c, 0xaa, 0x34, 0xb7, 0xed, 0x7f	EFI_NVM_EXPRESS_PASS _THRU_PROTOCOL. GetNamespace() - GetNamespace() returns EFI_INVALID_PARAMETER with NULL DevicePath.	1. Call GetNamespace() with NULL DevicePath, the return status should be EFI_INVALID_PARAMETER.
5.7.12.4.3	0x365f8fba, 0x3314, 0x4502, 0x89, 0x3e, 0x8e, 0x63, 0xc1, 0xda, 0xfe, 0xbc	EFI_NVM_EXPRESS_PASS _THRU_PROTOCOL. GetNamespace() - GetNamespace() returns EFI_UNSUPPORTED with unsupported device path node.	1. Call GetNamespace() with unsupported device path node, the return status should be EFI_UNSUPPORTED.
5.7.12.4.4	0xe864012d, 0x12b0, 0x4467, 0xa9, 0x7b, 0x5f, 0x72, 0xb4, 0xa9, 0x50, 0x27	EFI_NVM_EXPRESS_PASS _THRU_PROTOCOL. GetNamespace() - GetNamespace() returns EFI_NOT_FOUND with NVME device path node, but the translation from DevicePath to namespace ID failed.	1. Call GetNamespace() with NVME device path node, but translation from DevicePath to namespace ID failed, the return status should be EFI_NOT_FOUND.
5.7.12.4.5	0xc72a5f58, 0x742a, 0x4c7f, 0xbc, 0xc1, 0x35, 0xf9, 0xd0, 0x31, 0x32, 0xd	EFI_NVM_EXPRESS_PASS _THRU_PROTOCOL. GetNamespace() - GetNamespace() returns EFI_SUCCESS with valid parameters.	1. Call GetNamespace() with valid parameters, the return status should be EFI_SUCCESS.

9.13 EFI_ERASE_BLOCK_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_ERASE_BLOCK_PROTOCOL Section.

9.13.1 EraseBlocks()

Number	GUID	Assertion	Test Description
5.7.13.1.1	0xf62e99e3, 0xcda2, 0x4e44, 0x89, 0xa2, 0x47, 0x3b, 0xd8, 0x61, 0x90, 0xf8	FI_ERASE_BLOCK_PROTOCOL. EraseBlocks() - EraseBlocks(). This optional protocol should be installed on the same handle as the EFI_BLOCK_IO_PROTOCOL or EFI_BLOCK_IO2_PROTOCOL.	1. EFI_ERASE_BLOCK_PROTOCOL should be installed on the same handle as the EFI_BLOCK_IO_PROTOCOL or EFI_BLOCK_IO2_PROTOCOL.
5.7.13.1.2	0x4cfed8bb, 0xb9b1, 0x4c21, 0xb3, 0xb6, 0xa7, 0x5, 0x38, 0x6c, 0xf1, 0xe5	EFI_ERASE_BLOCK_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI_NO_MEDIA when there is no media in the device.	1. Call EraseBlocks() when there is no media in the device, the return status should be EFI_NO_MEDIA.
5.7.13.1.3	0x9877f323, 0x8812, 0x40bc, 0xbd, 0x41, 0x71, 0xe, 0x8b, 0xbe, 0xb6, 0x69	EFI_ERASE_BLOCK_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI_NO_MEDIA when there is no media in the device, even if LBA is invalid.	1. Call EraseBlocks() when there is no media in the device, even if LBA is invalid, the return status should be EFI_NO_MEDIA.
5.7.13.1.4	0x9877cf0d, 0x3d1b, 0x4ac5, 0x8a, 0x3f, 0x8c, 0xba, 0x95, 0x62, 0xb7, 0x53	EFI_ERASE_BLOCK_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI_NO_MEDIA when there is no media in the device, even if Size is invalid.	1. Call EraseBlocks() when there is no media in the device, even if Size is invalid, the return status should be EFI_NO_MEDIA.
5.7.13.1.5	0x61c0575e, 0x742f, 0x4094, 0xa8, 0x73, 0x2, 0x11, 0x4, 0xdb, 0x45, 0x1d	EFI_ERASE_BLOCK_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI_WRITE_PROTECTED when there is media in the device, but with the read only attribute.	1. Call EraseBlocks() when there is media in the device, but with the read only attribute, the return status should be EFI_WRITE_PROTECTED.
5.7.13.1.6	0x2176fd0d, 0xb211, 0x426d, 0xbf, 0xc, 0x84, 0x65, 0x5f, 0x3e, 0x3c, 0xcd	EFI_ERASE_BLOCK_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI_MEDIA_CHANGED when the MediaId is not for the current media.	1. Call EraseBlocks() when the MediaId is not for the current media, the return status should be EFI_MEDIA_CHANGED.
5.7.13.1.7	0x5d60ba1c, 0x42da, 0x4a50, 0x82, 0xbc, 0xe5, 0xbe, 0xe2, 0x3f, 0x41, 0x4f	EFI_ERASE_BLOCK_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI_NO_MEDIA when the MediaId is not for the current media, even if LBA is invalid.	1. Call EraseBlocks() when the MediaId is not for the current media, even if LBA is invalid, the return status should be EFI_NO_MEDIA.

Number	GUID	Assertion	Test Description
5.7.13.1.8	0x702c5141, 0xc1a8, 0x42ee, 0x8f, 0x9c, 0xe6, 0x8, 0x8e, 0x33, 0x2a, 0xe6	EFI_ERASE_BLOCK_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI_NO_MEDIA when the MediaId is not for the current media, even if Size is invalid.	1. Call EraseBlocks() when the MediaId is not for the current media, even if Size is invalid, the return status should be EFI_NO_MEDIA.
5.7.13.1.9	0x2864536a, 0x9aa4, 0x44ac, 0xa9, 0x60, 0x3b, 0x6e, 0x4e, 0x93, 0x47, 0xb5	EFI_ERASE_BLOCK_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI_INVALID_PARAMETER when the LBA is invalid.	1. Call EraseBlocks() when the LBA is invalid, the return status should be EFI_INVALID_PARAMETER.
5.7.13.1.10	0xb9ec66f1, 0x41ae, 0x44dc, 0xa6, 0xcc, 0x55, 0xde, 0x3b, 0x0, 0x37, 0xca	EFI_ERASE_BLOCK_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI_SUCCESS with the valid parameters.	1. Call BlockIo->ReadBlocks() to read the data from the specified area. 2. Call EraseBlocks() to erase the same area, the return status should be EFI_SUCCESS. 3. Call BlockIo->ReadBlocks() to read the same area, the content should be zero. 4. Call BlockIo->WriteBlocks() to restore the original data back.
5.7.13.1.11	0x2af1346c, 0xf3d8, 0x48d9, 0x94, 0x61, 0x6e, 0xef, 0xf6, 0xb2, 0x48, 0x3c	EFI_ERASE_BLOCK_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI_SUCCESS with the valid parameters.	1. Call BlockIo2->ReadBlocks() to read the data from the specified area. 2. Call EraseBlocks() to erase the same area, the return status should be EFI_SUCCESS. 3. Call BlockIo2->ReadBlocks() to read the same area, the content should be zero. 4. Call BlockIo2->WriteBlocks() to restore the original data back.

9.14 EFI_SD_MMC_PASS_THRU_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_SD_MMC_PASS_THRU_PROTOCOL Section.

9.14.1 PassThru()

Number	GUID	Assertion	Test Description
5.7.14.1.1	0x572e13de, 0xcd2e, 0x43ef, 0xa6, 0x41, 0x37, 0x1, 0x28, 0x18, 0xf8, 0xe4	EFI_SD_MMC_PASS_THRU_PROTOCOL. PassThru() - PassThru() returns EFI_INVALID_PARAMETER when Packet is NULL.	1. Call PassThru() when Packet is NULL, the return status should be EFI_INVALID_PARAMETER.
5.7.14.1.2	0x2df7228c, 0x94b9, 0x4a93, 0x90, 0x21, 0xff, 0xdc, 0xae, 0xa, 0x29, 0x53	EFI_SD_MMC_PASS_THRU_PROTOCOL. PassThru() - PassThru() returns EFI_INVALID_PARAMETER when the content of Packet is NULL.	1. Call PassThru() when the content of Packet is NULL, the return status should be EFI_INVALID_PARAMETER.
5.7.14.1.3	0x52b9c6df, 0xb7f6, 0x4cca, 0x9a, 0x70, 0xd6, 0x21, 0x72, 0x60, 0xdd, 0x0	EFI_SD_MMC_PASS_THRU_PROTOCOL. PassThru() - PassThru() returns EFI_INVALID_PARAMETER when Packet defines a data command but both InDataBuffer and OutDataBuffer are NULL.	1. Call PassThru() when Packet defines a data command but both InDataBuffer and OutDataBuffer are NULL, the return status should be EFI_INVALID_PARAMETER.
5.7.14.1.4	0x516deffa, 0x25ef, 0x4cb6, 0x95, 0xdf, 0xe0, 0x71, 0x93, 0xf0, 0xc4, 0xb5	EFI_SD_MMC_PASS_THRU_PROTOCOL. PassThru() - PassThru() returns EFI_INVALID_PARAMETER when Slot is invalid.	1. Call PassThru() when Slot is invalid, the return status should be EFI_INVALID_PARAMETER.
5.7.14.1.5	0x205e3e70, 0x92b1, 0x4534, 0x80, 0x21, 0xf2, 0x39, 0xcc, 0x21, 0xb5, 0x78	EFI_SD_MMC_PASS_THRU_PROTOCOL. PassThru() - PassThru(). The IoAlign should be 0, 1 or the power of 2.	1. The IoAlign should be 0, 1 or the power of 2.
5.7.14.1.6	0xd481f4ac, 0xed73, 0x4bd9, 0xab, 0xa1, 0x4f, 0xcc, 0xa5, 0x40, 0x95, 0x8e	EFI_SD_MMC_PASS_THRU_PROTOCOL. PassThru() - PassThru() returns EFI_SUCCESS when the SD Command Packet was sent by the host.	1. Call PassThru() when the SD Command Packet was sent by the host, the return status should be EFI_INVALID_PARAMETER.

9.14.2 GetNextSlot()

Number	GUID	Assertion	Test Description
5.7.14.2.1	0xcd9e89de, 0x9765, 0x4930, 0xa1, 0x88, 0xbc, 0x30, 0xd4, 0x9, 0xa0, 0x92	EFI_SD_MMC_PASS_THRU_PROTOCOL. GetNextSlot() - GetNextSlot() returns EFI_INVALID_PARAMETER when Slot is not 0xFF and Slot was not returned on a previous call.	1. Call GetNextSlot() when Slot is not 0xFF and Slot was not returned on a previous call, the return status should be EFI_INVALID_PARAMETER.
5.7.14.2.2	0x8f6d644f, 0x2d1e, 0x40b3, 0x91, 0x4a, 0xc6, 0xda, 0x21, 0x3, 0x82, 0x44	EFI_SD_MMC_PASS_THRU_PROTOCOL. GetNextSlot() - GetNextSlot() returns EFI_NOT_FOUND when there are no more slots on this SD controller.	1. Call GetNextSlot() when there are no more slots on this SD controller, the return status should be EFI_NOT_FOUND.

9.14.3 BuildDevicePath()

Number	GUID	Assertion	Test Description
5.7.14.3.1	0x962accdc, 0x5808, 0x450d, 0xba, 0xea, 0xe3, 0xb7, 0x1a, 0x34, 0x76, 0x22	EFI_SD_MMC_PASS_THRU_PROTOCOL. BuildDevicePath() - BuildDevicePath () returns EFI_INVALID_PARAMETER when DevicePath is NULL.	1. Call BuildDevicePath() when DevicePath is NULL, the return status should be EFI_INVALID_PARAMETER.
5.7.14.3.2	0x2597450b, 0xab3d, 0x49d6, 0x9c, 0x3f, 0xec, 0xcd, 0x24, 0xcc, 0xb5, 0xf5	EFI_SD_MMC_PASS_THRU_PROTOCOL. BuildDevicePath() - BuildDevicePath () returns EFI_NOT_FOUND when the SD card specified by Slot does not exist on the SD controller.	1. Call BuildDevicePath() when the SD card specified by Slot does not exist on the SD controller, the return status should be EFI_NOT_FOUND.
5.7.14.3.3	0x871efb1e, 0xdfbe, 0x4a0c, 0x83, 0xc4, 0x21, 0x9c, 0x20, 0x91, 0x8e, 0x91	EFI_SD_MMC_PASS_THRU_PROTOCOL. BuildDevicePath() - BuildDevicePath () returns EFI_SUCCESS when the device path node that describes the SD card specified by Slot was allocated and returned in DevicePath.	1. Call BuildDevicePath() when the device path node that describes the SD card specified by Slot was allocated and returned in DevicePath, the return status should be EFI_SUCCESS.

9.14.4 GetSlotNumber()

Number	GUID	Assertion	Test Description
5.7.14.4.1	0xab2880b3, 0x9ac3, 0x4ca4, 0x94, 0x75, 0x4e, 0xbd, 0xd1, 0xbe, 0xa, 0xd8	EFI_SD_MMC_PASS_THRU_PROTOCOL. GetSlotNumber() - GetSlotNumber() returns EFI_INVALID_PARAMETER when DevicePath is NULL.	1. Call GetSlotNumber() when DevicePath is NULL, the return status should be EFI_INVALID_PARAMETER.
5.7.14.4.2	0xff66737b, 0xad5c, 0x4383, 0xbe, 0x96, 0x9a, 0xff, 0xd7, 0xe2, 0xb3, 0x7a	EFI_SD_MMC_PASS_THRU_PROTOCOL. GetSlotNumber() - GetSlotNumber() returns EFI_INVALID_PARAMETER when Slot is NULL.	1. Call GetSlotNumber() when Slot is NULL, the return status should be EFI_INVALID_PARAMETER.
5.7.14.4.3	0x240951b8, 0xaa03, 0x4517, 0xb0, 0xa7, 0x3a, 0xbc, 0x57, 0x5a, 0xc, 0x3e	EFI_SD_MMC_PASS_THRU_PROTOCOL. GetSlotNumber() - GetSlotNumber() returns EFI_UNSUPPORTED when DevicePath is not a device path node type that the SD PassThru driver supports.	1. Call GetSlotNumber() when DevicePath is not a device path node type that the SD PassThru driver supports, the return status should be EFI_UNSUPPORTED.
5.7.14.4.4	0xb0631fb9, 0xd1f9, 0x41e6, 0xb1, 0x74, 0x18, 0xea, 0x2, 0x59, 0xd4, 0x7a	EFI_SD_MMC_PASS_THRU_PROTOCOL. GetSlotNumber() - GetSlotNumber() returns EFI_SUCCESS when SD card slot number is returned in Slot.	1. Call GetSlotNumber() when SD card slot number is returned in Slot, the return status should be EFI_SUCCESS.

9.14.5 ResetDevice()

Number	GUID	Assertion	Test Description
5.7.14.5.1	0x2dbb3a26, 0xb27, 0x4333, 0xa2, 0xec, 0xc3, 0x48, 0xee, 0xf9, 0xc9, 0x3e	EFI_SD_MMC_PASS_THRU_PROTOCOL. ResetDevice() - ResetDevice() returns EFI_INVALID_PARAMETER when Slot number is invalid or the SD controller does not support a device reset operation.	1. Call ResetDevice() when Slot number is invalid or the SD controller does not support a device reset operation, the return status should be EFI_INVALID_PARAMETER.
5.7.14.5.2	0x70c428ae, 0xf1a6, 0x4d02, 0xa1, 0x26, 0x47, 0x89, 0x14, 0xf5, 0xb5, 0xa2	EFI_ERASE_BLOCK_PROTOCOL. ResetDevice() - ResetDevice() returns EFI_SUCCESS when the SD card specified by the Slot is reset.	1. Call ResetDevice() when the SD card specified by the Slot is reset, the return status should be EFI_SUCCESS.

9.15 EFI_RAM_DISK_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_RAM_DISK_PROTOCOL Section

9.15.1 Register()

Number	GUID	Assertion	Test Description
5.7.15.1.1	0xf57e3b87, 0x2b93, 0x4645, 0x86, 0x56, 0x9a, 0x59, 0x53, 0x34, 0x58, 0x4b	EFI_RAM_DISK_PROTOCOL.Register() - Register() returns EFI_INVALID_PARAMETER when RamDiskSize is 0.	1. Call Register() when RamDiskSize is 0, the return status should be EFI_INVALID_PARAMETER.
5.7.15.1.2	0x309c7941, 0x13be, 0x43f6, 0x83, 0x33, 0x1c, 0x49, 0x5e, 0x7d, 0xf3, 0x56	EFI_RAM_DISK_PROTOCOL.Register() - Register() returns EFI_INVALID_PARAMETER when RamDiskType is NULL.	1. Call Register() when RamDiskType is NULL, the return status should be EFI_INVALID_PARAMETER.
5.7.15.1.3	0x35c6688b, 0x7eb9, 0x4446, 0x94, 0x7f, 0x34, 0x39, 0x16, 0xc5, 0xb9, 0x65	EFI_RAM_DISK_PROTOCOL.Register() - Register() returns EFI_INVALID_PARAMETER when DevicePath is NULL.	1. Call Register() when DevicePath is NULL, the return status should be EFI_INVALID_PARAMETER.
5.7.15.1.4	0xbf0432c4, 0x5b9b, 0x42f9, 0x94, 0x62, 0x49, 0x57, 0xb, 0x86, 0x83, 0xe1	EFI_RAM_DISK_PROTOCOL.Register() - Register() returns EFI_ALREADY_STARTED when the created DevicePath instance is already present in the handle database.	1. Call Register() to register one RAM disk with specified address, size and type. 2. Call Register() with the same parameters again, the return status should be EFI_ALREADY_STARTED.
5.7.15.1.5	0xb5b749af, 0x5ad3, 0x4e79, 0x88, 0x68, 0x58, 0x64, 0x65, 0x24, 0x91, 0x5f	EFI_RAM_DISK_PROTOCOL.Register() - Register() returns EFI_SUCCESS with valid parameters.	1. Call Register() with valid parameters, the return status should be EFI_SUCCESS.

9.15.2 Unregister()

Number	GUID	Assertion	Test Description
5.7.15.2.1	0xf05eae55, 0x1dd7, 0x4a10, 0xba, 0x57, 0x38, 0x8d, 0x38, 0x5, 0x51, 0x10	EFI_RAM_DISK_PROTOCOL. Unregister() - Unregister() returns EFI_NOT_FOUND when DevicePath is not existed.	1. Call Unregister() when DevicePath is not existed, the return status should be EFI_NOT_FOUND.
5.7.15.2.2	0x6919f770, 0xf418, 0x4873, 0x81, 0x38, 0xc1, 0x45, 0x36, 0x80, 0x1d, 0x77	EFI_RAM_DISK_PROTOCOL. Unregister() - Unregister() returns EFI_INVALID_PARAMETER when DevicePath is NULL.	1. Call Unregister() when DevicePath is NULL, the return status should be EFI_INVALID_PARAMETER.
5.7.15.2.3	0xbc90d7f7, 0x275d, 0x424f, 0x9c, 0x95, 0x14, 0x6e, 0x24, 0xbd, 0xc3, 0xe6	EFI_RAM_DISK_PROTOCOL. Unregister() - Unregister() returns EFI_UNSUPPORTED when DevicePath is not the valid Ramdisk device path.	1. Call Unregister() when DevicePath is not the valid Ramdisk device path, the return status should be EFI_UNSUPPORTED.
5.7.15.2.4	0xa85e1978, 0x216f, 0x4f52, 0xad, 0x7c, 0x70, 0xc2, 0x65, 0xe6, 0xf7, 0xee	EFI_RAM_DISK_PROTOCOL. Unregister() - Unregister() returns EFI_SUCCESS with valid parameters.	1. Call Unregister() with valid parameters, the return status should be EFI_SUCCESS.

10 Protocols PCI Bus Support Test

10.1 EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_PCI_ROOT_BRIDGE_IO_ PROTOCOL Section.

Configuration

Some checkpoints in the **EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL** test are device related. If the user needs to check the protocol on the specified device, the related profile needs to be updated to provide the specified information about this device.

For the format of the profile, please refer to A.2.

10.1.1 PollMem()

Number	GUID	Assertion	Test Description
5.8.1.1.1	0xa10d3292, 0x6908, 0x446f, 0x9b, 0xfa, 0x38, 0x67, 0x75, 0xc6, 0x3e, 0x2e	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollMem – PollMem()** with the correct value written to the destination address before delay time out returns **EFI_SUCCESS**	1. Call **Mem.Write()** to write specific value to destination address before the **PollMem()** delay times out. 2. Call **PollMem()** to poll the specific value on destination address. It should return **EFI_SUCCESS** when required value is written to destination address.
5.8.1.1.2	0xec6af458, 0x3dc1, 0x4022, 0xae, 0x0a, 0x7a, 0xd5, 0x61, 0x58, 0xdc, 0x5c	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollMem – PollMem()** returns **EFI_SUCCESS** immediately when required value has been written to destination address.	1. Call **Mem.Write()** to write specific value to destination address before call of **PollMem()**. 2. Call **PollMem()** to poll the specific value on destination address. It should return **EFI_SUCCESS** immediately.

Number	GUID	Assertion	Test Description
5.8.1.1.3	0x6f82fa28, 0x8c61, 0x4af9, 0x8b, 0x77, 0xc9, 0xab, 0x26, 0x64, 0x10, 0x30	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollMem – PollMem()** with delay as 0 returns **EFI_SUCCESS** immediately.	1. Call **PollMem()** to poll the specific value on destination address with delay as 0. It should return **EFI_SUCCESS** immediately.
5.8.1.1.4	0x2f0c1ddc, 0x53f3, 0x4053, 0xa8, 0xce, 0x37, 0x0f, 0xff, 0xac, 0x56, 0x05	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollMem – PollMem()** with the invalid value written to the destination address before delay time out returns **EFI_TIME_OUT**	1. Call **Mem.Write()** to write specific value to destination address before the **PollMem()** delay time out. 2. Call **PollMem()** to poll the different value on destination address. The return code should be **EFI_TIME_OUT** after delay time out.
5.8.1.1.5	0x1d028ad2, 0xd563, 0x445e, 0x8c, 0x68, 0x92, 0x6f, 0x66, 0x35, 0x12, 0xa5	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollMem – PollMem()** with *Width* as **EfiPciWidthMaximum** returns **EFI_INVALID_PARAMETER**.	1. Call **PollMem()** with *Width* as **EfiPciWidthMaximum**. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.1.6	0x78d809be, 0xa958, 0x4c16, 0xb7, 0xbc, 0xbd, 0xb0, 0x26, 0xa0, 0x10, 0x48	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollMem – PollMem()** with *Width* as **EfiPciWidthFifoUintX** returns **EFI_INVALID_PARAMETER**.	1. Call **PollMem()** with *Width* as **EfiPciWidthFifoUintX**. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.1.7	0x87dc296a, 0xa156, 0x4601, 0x8c, 0xfb, 0x25, 0xd5, 0xa5, 0xcb, 0x64, 0x11	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollMem – PollMem()** with *Width* as **EfiPciWidthFillUintX** returns **EFI_INVALID_PARAMETER**.	1. Call **PollMem()** with *Width* as **EfiPciWidthFillUintX**. The return code should be **EFI_INVALID_PARAMETER**

Number	GUID	Assertion	Test Description
5.8.1.1.8	0x4e02eeec, 0x660d, 0x4782, 0xb2, 0xec, 0x2f, 0x5a, 0x66, 0x6c, 0xf2, 0xb7	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollMem – PollMem()** with *Width* as -1 returns **EFI_INVALID_PARAMETER**.	1. Call **PollMem()** with *Width* as -1. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.1.9	0x438d7bdd, 0x3e1b, 0x44dc, 0xb3, 0x53, 0x54, 0xf1, 0x9f, 0x02, 0x2d, 0x88	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollMem – PollMem()** with *Result* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **PollMem()** with *Result* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**

10.1.2 PollIo()

Number	GUID	Assertion	Test Description
5.8.1.2.1	0x7f89a139, 0x7bba, 0x41da, 0xaa, 0x92, 0x1c, 0xe3, 0xc4, 0x77, 0x97, 0x68	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollIo – PollIo()** with the correct value written to the destination Io address before delay time out returns **EFI_SUCCESS**	1. Call **Io.Write()** to write specific value to destination Io address before the **PollIo()** delay time out. 2. Call **PollIo()** to poll the specific value on destination Io address. It should return **EFI_SUCCESS** when required value is written to destination address.
5.8.1.2.2	0xf6882063, 0xc841, 0x4822, 0xa9, 0x86, 0x16, 0x7e, 0xce, 0x5b, 0x2c, 0x76	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollIo – PollIo()** returns **EFI_SUCCESS** immediately when required value has been written to destination address.	1. Call **Io.Write()** to write specific value to destination address before call of **PollIo()**. 2. Call **PollIo()** to poll the specific value on destination address. It should return **EFI_SUCCESS** immediately.
5.8.1.2.3	0x2ba92ffe, 0x557b, 0x4e2e, 0xa1, 0x22, 0x7c, 0x12, 0x36, 0x87, 0xdf, 0x6a	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollIo – PollIo()** with delay as 0 returns **EFI_SUCCESS** immediately.	1. Call **PollIo()** to poll the specific value on destination address with delay as 0. It should return **EFI_SUCCESS** immediately.

Number	GUID	Assertion	Test Description
5.8.1.2.4	0x424cfc17, 0x7335, 0x49d5, 0xb7, 0x9f, 0xa5, 0xfd, 0x90, 0xf2, 0xc5, 0x5e	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollIo – PollIo() with the invalid value written to the destination address before delay time out returns EFI_TIME_OUT	1. Call Io.Write() to write specific value to destination address before the PollIo() delay time out. 2. Call PollIo() to poll the different value on destination address. The return code should be EFI_TIME_OUT after delay time out.
5.8.1.2.5	0xb46d5e49, 0xe908, 0x4874, 0x96, 0x2f, 0xf8, 0x4e, 0x21, 0x6d, 0xcb, 0x54	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollIo – PollIo() with *Width* as EfiPciWidthMaximum returns EFI_INVALID_PARAMETER.	1. Call PollIo() with *Width* as EfiPciWidthMaximum. The return code should be EFI_INVALID_PARAMETER
5.8.1.2.6	0x90f1257b, 0x115e, 0x4d5d, 0xa1, 0x83, 0x09, 0xed, 0xc9, 0x5c, 0x18, 0x08	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollIo – PollIo() with *Width* as EfiPciWidthFifoUintX returns EFI_INVALID_PARAMETER.	1. Call PollIo() with *Width* as EfiPciWidthFifoUintX. The return code should be EFI_INVALID_PARAMETER
5.8.1.2.7	0xf557d70d, 0x4418, 0x4903, 0x8a, 0xb7, 0x66, 0x6f, 0x11, 0x1a, 0xd3, 0x37	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollIo – PollIo() with *Width* as EfiPciWidthFillUintX returns EFI_INVALID_PARAMETER.	1. Call PollIo() with *Width* as EfiPciWidthFillUintX. The return code should be EFI_INVALID_PARAMETER
5.8.1.2.8	0xd00129f5, 0x35d4, 0x4c01, 0xa7, 0x41, 0x00, 0xc7, 0xd5, 0xa5, 0x19, 0x0f	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollIo – PollIo() with *Width* as -1 returns EFI_INVALID_PARAMETER.	1. Call PollIo() with *Width* as -1. The return code should be EFI_INVALID_PARAMETER
5.8.1.2.9	0x7465fa90, 0xa357, 0x442f, 0xa8, 0xec, 0xf8, 0x86, 0x5f, 0xb6, 0xe2, 0xca	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.PollIo – PollIo() with *Result* as NULL returns EFI_INVALID_PARAMETER.	1. Call PollIo() with *Result* as NULL. The return code should be EFI_INVALID_PARAMETER

10.1.3 Mem.Read()

Number	GUID	Assertion	Test Description
5.8.1.3.1	0x122320b0, 0x435d, 0x449b, 0x9c, 0xc0, 0x99, 0xd5, 0x95, 0xc9, 0xd2, 0x3d	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Read – Mem.Read()` with `EfiPciWidthUintX` returns `EFI_SUCCESS`.	1. Call `Mem.Read()` with data width as `EfiPciWidthUintX`(X=8,16,32) to read Mem address contents to buffer. The return code should be `EFI_SUCCESS`.
5.8.1.3.2	0xc29f3981, 0x0a68, 0x48f0, 0x99, 0xfe, 0xc2, 0xe4, 0x84, 0xe8, 0xd2, 0x9d	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Read – Mem.Read()` with `EfiPciWidthUintX` returns `EFI_SUCCESS`.	1. Call `Mem.Read()` with data width as `EfiPciWidthUintX`(X=8,16,32) to read Mem address contents to backup buffer. 2. Call `Mem.Write()` to write backup buffer contents to Mem address. 3. Call `Mem.Read()` again to read Mem address contents to another buffer. The return code should be `EFI_SUCCESS`.
5.8.1.3.3	0x57e2d8b2, 0xed4c, 0x4856, 0x82, 0xb6, 0xa0, 0xfd, 0x80, 0xd0, 0xb2, 0x55	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Read – Mem.Read()` with `EfiPciWidthUintX` returns the contents written by `Mem.Write()`.	1. Call `Mem.Read()` with data width as `EfiPciWidthUintX`(X=8,16,32) to read Mem address contents to backup buffer. 2. Call `Mem.Write()` to write backup buffer contents to Mem address. 3. Call `Mem.Read()` again to read Mem address contents to another buffer. The read contents in buffer should be the same as backup buffer.
5.8.1.3.4	0x729ba46d, 0x7962, 0x4a2b, 0xb5, 0x20, 0xbf, 0x52, 0xa2, 0x02, 0x3c, 0xbe	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Read – Mem.Read()` with `EfiPciWidthFifoUintX` returns `EFI_SUCCESS`.	1. Call `Mem.Read()` with data width as `EfiPciWidthFifoUintX`(X=8,16,32) to read Mem address contents to buffer. The return code should be `EFI_SUCCESS`.
5.8.1.3.5	0x701e90f7, 0xd218, 0x411f, 0xba, 0x7d, 0xb5, 0xab, 0x92, 0x2a, 0xcb, 0x93	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Read – Mem.Read()` with `EfiPciWidthFifoUintX` only increases buffer for each of the count operations performed.	1. Call `Mem.Write()` with `EfiPciWidthUintX` to write *`Buffer1`* to memory address. 2. Call `Mem.Read()` with data width as `EfiPciWidthFifoUintX` from the same memory address to *`Buffer2`*. All units of *`Buffer2`* should be the first unit of *`Buffer1`*.

Number	GUID	Assertion	Test Description
5.8.1.3.6	0x383c6e62, 0xf92f, 0x4719, 0x9a, 0x11, 0x70, 0x95, 0x08, 0x31, 0x19, 0xad	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Read – Mem.Read()` with `EfiPciWidthFillUintX` returns `EFI_SUCCESS`.	1. Call `Mem.Read()` with data width as `EfiPciWidthFillUintX`(X=8,16,32) to read Mem address contents to buffer. The return code should be `EFI_SUCCESS`.
5.8.1.3.7	0x596a5971, 0x11d4, 0x43b0, 0x82, 0x4d, 0xe5, 0xcc, 0x41, 0x81, 0x9e, 0x14	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Read – Mem.Read()` with `EfiPciWidthFillUintX` only increases address for each of the count operations performed.	1. Call `Mem.Write()` with `EfiPciWidthUintX` to write *`Buffer1`* to memory address. 2. Set all units of *`Buffer2`* with the same value. 2. Call `Mem.Read()` with data width as `EfiPciWidthFillUintX` from the same memory address to *`Buffer2`*. The first unit of *`Buffer2`* should be same as the last unit of *`Buffer1`* and other units of *`Buffer2`* should remain unchanged.
5.8.1.3.8	0x28ba919b, 0xbc04, 0x464a, 0xbb, 0xa0, 0x87, 0xee, 0xda, 0xc1, 0x0f, 0x33	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Read – Mem.Read()` with *`Width`* as `EfiPciWidthMaximum` returns `EFI_INVALID_PARAMETER`.	1. Call `Mem.Read()` with *`Width`* as `EfiPciWidthMaximum`. The return code should be `EFI_INVALID_PARAMETER`
5.8.1.3.9	0xbc884213, 0xe80e, 0x41e6, 0x81, 0x69, 0xbc, 0x46, 0x7d, 0x53, 0x40, 0x86	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Read – Mem.Read()` with *`Width`* as -1 returns `EFI_INVALID_PARAMETER`.	1. Call `Mem.Read()` with *`Width`* as -1. The return code should be `EFI_INVALID_PARAMETER`
5.8.1.3.10	0x8cc49d7f, 0x87be, 0x4a2e, 0x82, 0xc0, 0xce, 0xc2, 0xbf, 0xcb, 0xb1, 0x3d	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Read – Mem.Read()` with buffer as `NULL` returns `EFI_INVALID_PARAMETER`.	1. Call `Mem.Read()` with buffer as `NULL`. The return code should be `EFI_INVALID_PARAMETER`
5.8.1.3.11	0xbbf33c06, 0xa3a0, 0x4e13, 0xa3, 0xc7, 0x49, 0x23, 0x37, 0x07, 0xc9, 0x0d	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Read – Mem.Read()` with unsupported *`Width`* from profile returns `EFI_INVALID_PARAMETER`.	1. Call `Mem.Read()` with unsupported *`Width`* from profile. The return code should be `EFI_INVALID_PARAMETER`

10.1.4 Mem.Write()

Number	GUID	Assertion	Test Description
5.8.1.4.1	0x9dac86c8, 0xb700, 0x47ec, 0x95, 0x27, 0x9e, 0xf2, 0x39, 0x56, 0xbc, 0xca	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write – Mem.Write()** with **EfiPciWidthUintX** returns **EFI_SUCCESS**.	1. Call **Mem.Write()** with data width as **EfiPciWidthUintX**(X=8,16,32) to write buffer to Mem address. The return code should be **EFI_SUCCESS**.
5.8.1.4.2	0x1ed536a0, 0x7dbb, 0x4f97, 0xa7, 0xcd, 0xeb, 0xb4, 0xc4, 0x84, 0xab, 0x2b	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write – Mem.Read()** with **EfiPciWidthUintX** returns the contents written by **Mem.Write()**.	1. Call **Mem.Read()** to read Mem address contents to backup buffer. 2. Call **Mem.Write()** with data width as **EfiPciWidthUintX**(X=8,16,32) to write backup buffer contents to Mem address. 3. Call **Mem.Read()** again to read Mem address contents to another buffer. The read contents in buffer should be the same as backup buffer.
5.8.1.4.3	0xd2f05d14, 0xff03, 0x4b2d, 0x94, 0xbc, 0x11, 0xd7, 0x7a, 0x56, 0x20, 0x5e	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write – Mem.Write()** with **EfiPciWidthFifoUintX** returns **EFI_SUCCESS**.	1. Call **Mem.Write()** with data width as **EfiPciWidthFifoUintX**(X=8,16,32) to write buffer contents to Mem address. The return code should be **EFI_SUCCESS**.
5.8.1.4.4	0x2e0a75e3, 0x04f3, 0x47f4, 0x85, 0x8f, 0x75, 0x1a, 0x29, 0xcf, 0x1c, 0x6a	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write – Mem.Write()** with **EfiPciWidthFifoUintX** only increases buffer for each of the count operations performed.	1. Call **Mem.Read()** with **EfiPciWidthUintX** to read memory address contents to *Buffer1*. 2. Call **Mem.Write()** with **EfiPciWidthFifoUintX** to write *Buffer1* to memory address. 3. Call **Mem.Read()** with data width as **EfiPciWidthUintX** from the same memory address to *Buffer2*. The first unit of *Buffer2* should be the same as the last unit of *Buffer1*, and other units of *Buffer2* should be the same as corresponding units of *Buffer1*.

Number	GUID	Assertion	Test Description
5.8.1.4.5	0xd220d6da, 0xa7b9, 0x477f, 0xa6, 0xfb, 0xc1, 0x52, 0x43, 0xe9, 0x52, 0x5e	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write – Mem.Write()` with `EfiPciWidthFillUintX` returns `EFI_SUCCESS`.	1. Call `Mem.Write()` with data width as `EfiPciWidthFillUintX`(X=8,16,32) to write buffer contents to Mem address. The return code should be `EFI_SUCCESS`.
5.8.1.4.6	0x8283aeec, 0x2896, 0x460b, 0x9e, 0xf1, 0xe7, 0xa6, 0x89, 0xa4, 0x8c, 0x86	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write – Mem.Read()` after Mem.Write the data using `EfiPciIoWidthFillUintX` return `EFI_SUCCESS`.	1. Call `Mem.Write()` with data width as `EfiPciWidthFillUintX`(X=8,16,32) to write buffer contents to Mem address. 2. Call `Mem.Read()` with data width as `EfiPciWidthUintX` to read Mem address contents to buffer. The return code should be `EFI_SUCCESS`.
5.8.1.4.7	0xcabf0b57, 0x7e2b, 0x40f6, 0x96, 0xa6, 0x3d, 0x4e, 0x92, 0xca, 0x5b, 0x55	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write – Mem.Write()` with `EfiPciWidthFillUintX` only increases address for each of the count operations performed.	1. Call `Mem.Write()` with data width as `EfiPciWidthFillUintX`(X=8,16,32) to write *`Buffer1`* contents to Mem address. 2. Call `Mem.Read()` with data width as `EfiPciWidthUintX` to read Mem address contents to *`Buffer2`*. All the units of *`Buffer2`* should be the same as the first unit of *`Buffer1`*.
5.8.1.4.8	0xaa2e8dd7, 0x501e, 0x4210, 0x8f, 0x10, 0xd0, 0x30, 0x78, 0x30, 0x75, 0x64	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write – Mem.Write()` with `EfiPciWidthUintX` returns `EFI_SUCCESS`.	1. Call `Mem.Read()` with data width as `EfiPciWidthUintX`(X=8,16,32) to write buffer back to Mem address. The return code should be `EFI_SUCCESS`.
5.8.1.4.9	0x26aa2144, 0x1c21, 0x4499, 0xb4, 0xdb, 0xda, 0xf4, 0x80, 0x07, 0xfa, 0xd9	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write – Mem.Write()` with *`Width`* as `EfiPciWidthMaximum` returns `EFI_INVALID_PARAMETER`.	1. Call `Mem.Write()` with *`Width`* as `EfiPciWidthMaximum`. The return code should be `EFI_INVALID_PARAMETER`

Number	GUID	Assertion	Test Description
5.8.1.4.10	0x71b8a5d8, 0xf464, 0x416d, 0xb9, 0x73, 0x4e, 0xb0, 0xc1, 0x06, 0x94, 0x07	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write` – `Mem.Write()` with *`Width`* as -1 returns `EFI_INVALID_PARAMETER`.	1. Call `Mem.Write()` with *`Width`* as -1. The return code should be `EFI_INVALID_PARAMETER`
5.8.1.4.11	0x2b698420, 0x82b3, 0x43b3, 0xaa, 0x39, 0x53, 0xc2, 0x9d, 0x1d, 0x91, 0x13	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write` – `Mem.Write()` with buffer as `NULL` returns `EFI_INVALID_PARAMETER`.	1. Call `Mem.Write()` with buffer as `NULL`. The return code should be `EFI_INVALID_PARAMETER`
5.8.1.4.12	0xcf2417f3, 0x1491, 0x44ea, 0x93, 0xec, 0xad, 0x0b, 0x5b, 0xc0, 0x2b, 0xc6	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Mem.Write` – `Mem.Write()` with unsupported *`Width`* from profile returns `EFI_INVALID_PARAMETER`.	1. Call `Mem.Write()` with unsupported *`Width`* from profile. The return code should be `EFI_INVALID_PARAMETER`

10.1.5 Io.Read()

Number	GUID	Assertion	Test Description
5.8.1.5.1	0xf6d5c145, 0x15c9, 0x4bc5, 0xa5, 0x1c, 0xd5, 0xfd, 0xba, 0xf0, 0x73, 0xe9	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Read` – `Io.Read()` with `EfiPciWidthUintX` returns `EFI_SUCCESS`.	1. Call `Io.Read()` with data width as `EfiPciWidthUintX`(X=8,16,32) to read Io address contents to buffer. The return code should be `EFI_SUCCESS`.
5.8.1.5.2	0x12a1078a, 0xc78a, 0x446d, 0x90, 0x37, 0x22, 0xd8, 0xd0, 0x88, 0xfb, 0x2d	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Read` – `Io.Read()` with `EfiPciWidthUintX` returns `EFI_SUCCESS`.	1. Call `Io.Read()` with data width as `EfiPciWidthUintX`(X=8,16,32) to read Io address contents to backup buffer. 2. Call `Io.Write()` to write backup buffer contents to Io address. 3. Call `Io.Read()` again to read Io address contents to another buffer. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.8.1.5.3	0xcc985605, 0x262d, 0x4954, 0xb4, 0x1c, 0xa9, 0x4c, 0xd0, 0x15, 0x7b, 0x96	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Read – Io.Read()** with **EfiPciWidthUintX** returns the contents written by **Io.Write()**.	1. Call **Io.Read()** with data width as **EfiPciWidthUintX**(X=8,16,32) to read Io address contents to backup buffer. 2. Call **Io.Write()** to write backup buffer contents to Io address. 3. Call **Io.Read()** again to read Io address contents to another buffer. The read contents in buffer should be the same as backup buffer.
5.8.1.5.4	0x0d6630e0, 0x4a9e, 0x4720, 0xa2, 0xe1, 0x4e, 0xf3, 0xef, 0x81, 0x5f, 0x41	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Read – Io.Read()** with **EfiPciWidthFifoUintX** returns **EFI_SUCCESS**.	1. Call **Io.Read()** with data width as **EfiPciWidthFifoUintX**(X=8,16,32) to read Io address contents to buffer. The return code should be **EFI_SUCCESS**.
5.8.1.5.5	0xddb273f7, 0xd3d7, 0x4ab2, 0xa2, 0x41, 0xcb, 0x78, 0x05, 0x76, 0x79, 0xe0	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Read – Io.Read()** with **EfiPciWidthFifoUintX** only increases buffer for each of the count operations performed.	1. Call **Io.Write()** with **EfiPciWidthUintX** to write *Buffer1* to Io address. 2. Call **Io.Read()** with data width as **EfiPciWidthFifoUintX** from the same Io address to *Buffer2*. All units of *Buffer2* should be the first unit of *Buffer1*.
5.8.1.5.6	0x349eb44d, 0x2db1, 0x4fa7, 0xa3, 0xf2, 0x1a, 0x08, 0x8d, 0xa9, 0x0e, 0x3c	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Read – Io.Read()** with **EfiPciWidthFillUintX** returns **EFI_SUCCESS**.	1. Call **Io.Read()** with data width as **EfiPciWidthFillUintX**(X=8,16,32) to read Io address contents to buffer. The return code should be **EFI_SUCCESS**.
5.8.1.5.7	0x3dcc7e09, 0x598c, 0x4fdb, 0xbb, 0x03, 0xda, 0xa6, 0x1a, 0xc9, 0x9f, 0x28	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Read – Io.Read()** with **EfiPciWidthFillUintX** only increases address for each of the count operations performed.	1. Call **Io.Write()** with **EfiPciWidthUintX** to write *Buffer1* to Io address. 2. Set all units of *Buffer2* with the same value. 2. Call **Io.Read()** with data width as **EfiPciWidthFillUintX** from the same Io address to *Buffer2*. The first unit of *Buffer2* should be same as the last unit of *Buffer1* and other units of *Buffer2* should remain unchanged.

Number	GUID	Assertion	Test Description
5.8.1.5.8	0xb7153211, 0xaf3b, 0x4a10, 0x85, 0x16, 0x5d, 0x5b, 0x13, 0x1d, 0x9e, 0x67	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Read – Io.Read() with *Width* as EfiPciWidthMaximum returns EFI_INVALID_PARAMETER.	1. Call Io.Read() with *Width* as EfiPciWidthMaximum. The return code should be EFI_INVALID_PARAMETER
5.8.1.5.9	0x8578f6de, 0xc396, 0x42f7, 0x92, 0x42, 0x74, 0x37, 0x13, 0xdb, 0xbf, 0x6d	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Read – Io.Read() with *Width* as -1 returns EFI_INVALID_PARAMETER.	1. Call Io.Read() with *Width* as -1. The return code should be EFI_INVALID_PARAMETER
5.8.1.5.10	0x50b7d46a, 0x73b5, 0x4bba, 0xa7, 0x36, 0x8a, 0xae, 0x97, 0x5c, 0x42, 0x6b	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Read – Io.Read() with buffer as NULL returns EFI_INVALID_PARAMETER.	1. Call Io.Read() with buffer as NULL. The return code should be EFI_INVALID_PARAMETER
5.8.1.5.11	0xb24b8daa, 0x5ea2, 0x47d0, 0x88, 0xc0, 0x32, 0x3b, 0x26, 0x43, 0x2f, 0xbc	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Read – Io.Read() with unsupported *Width* from profile returns EFI_INVALID_PARAMETER.	1. Call Io.Read() with unsupported *Width* from profile. The return code should be EFI_INVALID_PARAMETER

10.1.6 Io.Write()

Number	GUID	Assertion	Test Description
5.8.1.6.1	0xa0954c3a, 0x86d9, 0x43a8, 0xb0, 0xcb, 0x13, 0xcf, 0x13, 0xe2, 0x82, 0x50	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Write – Io.Write() with EfiPciWidthUintX returns EFI_SUCCESS.	1. Call Io.Write() with data width as EfiPciWidthUintX(X=8,16,32) to write buffer to Io address. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.8.1.6.2	0xe401d5de, 0x3a4e, 0x4e21, 0xb1, 0x4c, 0x34, 0x90, 0xc6, 0xe8, 0xf3, 0xd8	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Write – Io.Read() with EfiPciWidthUintX returns the contents written by Io.Write().	1. Call Io.Read() to read Io address contents to backup buffer. 2. Call Io.Write() with data width as EfiPciWidthUintX(X=8,16,32) to write backup buffer contents to Io address. 3. Call Io.Read() again to read Io address contents to another buffer. The read contents in buffer should be the same as backup buffer.
5.8.1.6.3	0xef5142b5, 0xe421, 0x43b8, 0xb1, 0xd5, 0x17, 0x60, 0x46, 0x60, 0x72, 0x3a	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Write – Io.Write() with EfiPciWidthFifoUintX returns EFI_SUCCESS.	1. Call Io.Write() with data width as EfiPciWidthFifoUintX(X=8,16,32) to write buffer contents to Io address. The return code should be EFI_SUCCESS.
5.8.1.6.4	0xd2f5dadf, 0x82f7, 0x4d25, 0x9a, 0x96, 0x50, 0xd5, 0xb6, 0xfe, 0x86, 0xbf	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Write – Io.Write() with EfiPciWidthFifoUintX only increases buffer for each of the count operations performed.	1. Call Io.Read() with EfiPciWidthUintX to read Io address contents to *Buffer1*. 2. Call Io.Write() with EfiPciWidthFifoUintX to write *Buffer1* to Io address. 3. Call Io.Read() with data width as EfiPciWidthUintX from the same Io address to *Buffer2*. The first unit of *Buffer2* should be the same as the last unit of *Buffer1*, and other units of *Buffer2* should be the same as corresponding units of *Buffer1*.
5.8.1.6.5	0xf6433206, 0xe359, 0x4a42, 0x82, 0x68, 0xb6, 0xbb, 0x68, 0x90, 0x6a, 0x3a	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Write – Io.Write() with EfiPciWidthFillUintX returns EFI_SUCCESS.	1. Call Io.Write() with data width as EfiPciWidthFillUintX(X=8,16,32) to write buffer contents to Io address. The return code should be EFI_SUCCESS.
5.8.1.6.6	0x8912391c, 0xf457, 0x4e51, 0x82, 0xb4, 0xe8, 0xaf, 0x1c, 0x5a, 0x18, 0xc2	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Write – Io.Write() with EfiPciWidthFillUintX only increases address for each of the count operations performed.	1. Call Io.Write() with data width as EfiPciWidthFillUintX(X=8,16,32) to write *Buffer1* contents to Io address. 2. Call Io.Read() with data width as EfiPciWidthUintX to read Io address contents to *Buffer2*. All the units of *Buffer2* should be the same as the first unit of *Buffer1*.

Number	GUID	Assertion	Test Description
5.8.1.6.7	0xe347d0ed, 0x8fbd, 0x46c4, 0xbd, 0xfe, 0x27, 0x2f, 0x81, 0x3a, 0x84, 0x85	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Write – Io.Write()** with **EfiPciWidthUintX** returns **EFI_SUCCESS**.	1. Call **Io.Read()** with data width as **EfiPciWidthUintX**(X=8,16,32) to write buffer back to Io address. The return code should be **EFI_SUCCESS**.
5.8.1.6.8	0x21d34064, 0x9df8, 0x4edf, 0x81, 0xd8, 0xeb, 0x90, 0x9c, 0xe7, 0x53, 0xd5	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Write – Io.Write()** with *Width* as **EfiPciWidthMaximum** returns **EFI_INVALID_PARAMETER**.	1. Call **Io.Write()** with *Width* as **EfiPciWidthMaximum**. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.6.9	0x9174967b, 0x1639, 0x46b0, 0xab, 0x66, 0x70, 0x59, 0x4e, 0x5a, 0x3f, 0x57	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Write – Io.Write()** with *Width* as -1 returns **EFI_INVALID_PARAMETER**.	1. Call **Io.Write()** with *Width* as -1. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.6.10	0x429ab4d0, 0x8d64, 0x4308, 0xa3, 0x08, 0x3e, 0x48, 0xa5, 0x66, 0x70, 0x4b	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Write – Io.Write()** with buffer as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **Io.Write()** with buffer as **NULL**. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.6.11	0x3d761cee, 0x9d62, 0x4942, 0x91, 0xde, 0xa9, 0xca, 0x93, 0xe4, 0xd5, 0x31	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Io.Write – Io.Write()** with unsupported *Width* from profile returns **EFI_INVALID_PARAMETER**.	1. Call **Io.Write()** with unsupported *Width* from profile. The return code should be **EFI_INVALID_PARAMETER**

10.1.7 Pci.Read()

Number	GUID	Assertion	Test Description
5.8.1.7.1	0x0a24c289, 0xe2b2, 0x465e, 0x93, 0x03, 0x20, 0x4e, 0xae, 0x23, 0x88, 0xd5	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Read – Pci.Read()** with **EfiPciWidthUintX** returns **EFI_SUCCESS**.	1. Call **Pci.Read()** with data width as **EfiPciWidthUintX**(X=8,16,32) to read Pci address contents to buffer. The return code should be **EFI_SUCCESS**.
5.8.1.7.2	0x6a0884db, 0x48e2, 0x4330, 0x97, 0xa7, 0xf5, 0x26, 0x92, 0x4a, 0xf5, 0xea	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Read – Pci.Read()** with **EfiPciWidthUintX** returns **EFI_SUCCESS**.	1. Call **Pci.Read()** with data width as **EfiPciWidthUintX**(X=8,16,32) to read Pci address contents to backup buffer. 2. Call **Pci.Write()** to write backup buffer contents to Pci address. 3. Call **Pci.Read()** again to read Pci address contents to another buffer. The return code should be **EFI_SUCCESS**.
5.8.1.7.3	0x34b35b73, 0xdb30, 0x4343, 0x85, 0x9a, 0x13, 0xb9, 0xac, 0x6e, 0x88, 0x9a	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Read – Pci.Read()** with **EfiPciWidthUintX** returns the contents written by **Pci.Write()**.	1. Call **Pci.Read()** with data width as **EfiPciWidthUintX**(X=8,16,32) to read Pci address contents to backup buffer. 2. Call **Pci.Write()** to write backup buffer contents to Pci address. 3. Call **Pci.Read()** again to read Pci address contents to another buffer. The read contents in buffer should be the same as backup buffer.
5.8.1.7.4	0x0cb1fa0c, 0xfb2d, 0x4eed, 0x8d, 0x72, 0xb1, 0x65, 0x14, 0xcf, 0x95, 0xee	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Read – Pci.Read()** with **EfiPciWidthFifoUintX** returns **EFI_SUCCESS**.	1. Call **Pci.Read()** with data width as **EfiPciWidthFifoUintX**(X=8,16,32) to read Pci address contents to buffer. The return code should be **EFI_SUCCESS**.
5.8.1.7.5	0x95094926, 0x51ab, 0x43c1, 0xb6, 0xb3, 0x77, 0xba, 0x39, 0x8b, 0x4a, 0x94	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Read – Pci.Read()** with **EfiPciWidthFifoUintX** only increases buffer for each of the count operations performed.	1. Call **Pci.Write()** with **EfiPciWidthUintX** to write *Buffer1* to Pci address. 2. Call **Pci.Read()** with data width as **EfiPciWidthFifoUintX** from the same Pci address to *Buffer2*. All units of *Buffer2* should be the first unit of *Buffer1*.
5.8.1.7.6	0xfb4b5e93, 0x494b, 0x4865, 0x9e, 0xb0, 0x8c, 0xb5, 0xeb, 0x0d, 0x86, 0x64	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Read – Pci.Read()** with **EfiPciWidthFillUintX** returns **EFI_SUCCESS**.	1. Call **Pci.Read()** with data width as **EfiPciWidthFillUintX**(X=8,16,32) to read Pci address contents to buffer. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.8.1.7.7	0x711d56d9, 0x90d4, 0x422b, 0xad, 0x2b, 0xfe, 0xe9, 0x01, 0x2c, 0xfd, 0x7a	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Read – Pci.Read()** with **EfiPciWidthFillUintX** only increases address for each of the count operations performed.	1. Call **Pci.Write()** with **EfiPciWidthUintX** to write *Buffer1* to Pci address. 2. Set all units of *Buffer2* with the same value. 2. Call **Pci.Read()** with data width as **EfiPciWidthFillUintX** from the same Pci address to *Buffer2*. The first unit of *Buffer2* should be same as the last unit of *Buffer1* and other units of *Buffer2* should remain unchanged.
5.8.1.7.8	0xbeed4e4f, 0xf7aa, 0x480e, 0x97, 0xfd, 0x3d, 0xd8, 0x83, 0x5f, 0x47, 0x09	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Read – Pci.Read()** with *Width* as **EfiPciWidthMaximum** returns **EFI_INVALID_PARAMETER**.	1. Call **Pci.Read()** with *Width* as **EfiPciWidthMaximum**. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.7.9	0x1698aaaf, 0x8a6e, 0x4a56, 0xb6, 0xd5, 0x4e, 0xa4, 0x1d, 0x12, 0x2c, 0xb3	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Read – Pci.Read()** with *Width* as -1 returns **EFI_INVALID_PARAMETER**.	1. Call **Pci.Read()** with *Width* as -1. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.7.10	0x201fdef9, 0xdc84, 0x4c9d, 0x85, 0x98, 0x86, 0xf7, 0xca, 0x3f, 0xef, 0x81	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Read – Pci.Read()** with buffer as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **Pci.Read()** with buffer as **NULL**. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.7.11	0xe0a36a5f, 0x3be9, 0x4b11, 0x9e, 0xfb, 0x90, 0x07, 0x1c, 0x73, 0x99, 0xc9	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Read – Pci.Read()** with unsupported *Width* from profile returns **EFI_INVALID_PARAMETER**.	1. Call **Pci.Read()** with unsupported *Width* from profile. The return code should be **EFI_INVALID_PARAMETER**

10.1.8 Pci.Write()

Number	GUID	Assertion	Test Description
5.8.1.8.1	0x22abcbe1, 0x5a58, 0x47d0, 0xb7, 0x3a, 0x6d, 0x3c, 0x55, 0x7a, 0xe9, 0x7c	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Write – Pci.Write()` with `EfiPciWidthUintX` returns `EFI_SUCCESS`.	1. Call `Pci.Write()` with data width as `EfiPciWidthUintX`(X=8,16,32) to write buffer to Pci address. The return code should be `EFI_SUCCESS`.
5.8.1.8.2	0xb4e49e1b, 0xbe09, 0x4cdc, 0xbb, 0x56, 0xaa, 0x44, 0x4b, 0x86, 0xa6, 0x4a	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Write – Pci.Read()` with `EfiPciWidthUintX` returns the contents written by `Pci.Write()`.	1. Call `Pci.Read()` to read Pci address contents to backup buffer. 2. Call `Pci.Write()` with data width as `EfiPciWidthUintX`(X=8,16,32) to write backup buffer contents to Pci address. 3. Call `Pci.Read()` again to read Pci address contents to another buffer. The read contents in buffer should be the same as backup buffer.
5.8.1.8.3	0xd753202a, 0xbe16, 0x4a58, 0x88, 0x3a, 0xcb, 0x5b, 0x82, 0xdf, 0xb8, 0xe8	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Write – Pci.Write()` with `EfiPciWidthFifoUintX` returns `EFI_SUCCESS`.	1. Call `Pci.Write()` with data width as `EfiPciWidthFifoUintX`(X=8,16,32) to write buffer contents to Pci address. The return code should be `EFI_SUCCESS`.
5.8.1.8.4	0x241e4d94, 0xa5a2, 0x4192, 0x93, 0x66, 0x6d, 0x25, 0x8b, 0x20, 0x9b, 0xfc	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Write – Pci.Write()` with `EfiPciWidthFifoUintX` only increases buffer for each of the count operations performed.	1. Call `Pci.Read()` with `EfiPciWidthUintX` to read Pci address contents to *Buffer1*. 2. Call `Pci.Write()` with `EfiPciWidthFifoUintX` to write *Buffer1* to Pci address. 3. Call `Pci.Read()` with data width as `EfiPciWidthUintX` from the same Pci address to *Buffer2*. The first unit of *Buffer2* should be the same as the last unit of *Buffer1*, and other units of *Buffer2* should be the same as corresponding units of *Buffer1*.
5.8.1.8.5	0xadff8bd8, 0x7efd, 0x4368, 0x9b, 0x72, 0x0e, 0x9b, 0x10, 0xca, 0x13, 0x39	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Write – Pci.Write()` with `EfiPciWidthFillUintX` returns `EFI_SUCCESS`.	1. Call `Pci.Write()` with data width as `EfiPciWidthFillUintX`(X=8,16,32) to write buffer contents to Pci address. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.8.1.8.6	0xe9a41aa8, 0xd9be, 0x4b34, 0x99, 0xab, 0x40, 0x89, 0x08, 0x76, 0xc4, 0xe0	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Write – Pci.Write()** with **EfiPciWidthFillUintX** only increases address for each of the count operations performed.	1. Call **Pci.Write()** with data width as **EfiPciWidthFillUintX**(X=8,16,32) to write *Buffer1* contents to Pci address. 2. Call **Pci.Read()** with data width as **EfiPciWidthUintX** to read Pci address contents to *Buffer2*. All the units of *Buffer2* should be the same as the first unit of *Buffer1*.
5.8.1.8.7	0x91076895, 0x66a6, 0x4d26, 0x84, 0xca, 0x8d, 0x38, 0xeb, 0x96, 0xd7, 0x5f	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Write – Pci.Write()** with **EfiPciWidthUintX** returns **EFI_SUCCESS**.	1. Call **Pci.Read()** with data width as **EfiPciWidthUintX**(X=8,16,32) to write buffer back to Pci address. The return code should be **EFI_SUCCESS**.
5.8.1.8.8	0x7ff7a44c, 0x8647, 0x46de, 0x94, 0xe9, 0xe4, 0x0d, 0x30, 0xd1, 0x52, 0x41	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Write – Pci.Write()** with *Width* as **EfiPciWidthMaximum** returns **EFI_INVALID_PARAMETER**.	1. Call **Pci.Write()** with *Width* as **EfiPciWidthMaximum**. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.8.9	0x5928ba78, 0x13d0, 0x48bd, 0x8f, 0xf7, 0xa6, 0xee, 0x82, 0x79, 0xef, 0xea	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Write – Pci.Write()** with *Width* as -1 returns **EFI_INVALID_PARAMETER**.	1. Call **Pci.Write()** with *Width* as -1. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.8.10	0xb04a41bf, 0xa881, 0x4f93, 0xb6, 0x81, 0x14, 0x5c, 0xea, 0xaf, 0xa6, 0xa8	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Write – Pci.Write()** with buffer as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **Pci.Write()** with buffer as **NULL**. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.8.11	0x009e4d36, 0xdc7e, 0x45a6, 0xa7, 0xa5, 0xfa, 0x8b, 0x79, 0x11, 0xfb, 0x0c	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Pci.Write – Pci.Write()** with unsupported *Width* from profile returns **EFI_INVALID_PARAMETER**.	1. Call **Pci.Write()** with unsupported *Width* from profile. The return code should be **EFI_INVALID_PARAMETER**

10.1.9 CopyMem()

Number	GUID	Assertion	Test Description
5.8.1.9.1	0x73a0ec23, 0x176e, 0x4560, 0xb2, 0xa3, 0x77, 0x13, 0xae, 0x8e, 0x42, 0xd2	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem – CopyMem() between non-overlapping regions regions returns EFI_SUCCESS.	1. Set *Buffer1* with specific value. Call Mem.Write() to write *Buffer1* to *Address1* with count units. 2. Call CopyMem() to copy Mem from *Address1* to *Address1*+ BufferSize with count units. The return code should be EFI_SUCCESS.
5.8.1.9.2	0x6fd31187, 0xf3e6, 0x4b1d, 0x90, 0x61, 0xdc, 0xd8, 0x36, 0x98, 0xe6, 0xfc	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem – The data in destination address should be the same as the source address after call of CopyMem() between non-overlapping regions.	1. Set *Buffer1* with specific value. Call Mem.Write() to write *Buffer1* to Address1 with count units. 2. Call CopyMem() to copy Mem from *Address1* to *Address1*+ BufferSize with count units. 3. Call Mem.Read() to read data of *Address1*+BufferSize to *Buffer2*. All units of *Buffer2* should be the same as *Buffer1*.
5.8.1.9.3	0x4110b651, 0xb45e, 0x4684, 0xae, 0x38, 0x72, 0x8d, 0x01, 0xbb, 0x00, 0x97	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem – CopyMem() between overlapping regions with destination address > source address returns EFI_SUCCESS.	1. Set Buffer1 with specific value. Call Mem.Write() to write *Buffer1* to *Address1* with count units. 2. Call CopyMem() to copy Mem from *Address1* to *Address1*+ BufferSize/2 with count units. The return code should be EFI_SUCCESS.
5.8.1.9.4	0x2f84ec07, 0xa38a, 0x4db2, 0xac, 0x0f, 0x66, 0x4f, 0x91, 0x3b, 0xb3, 0xea	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem – After call of CopyMem() between overlapping regions, the data in destination address should be the same as the buffer contents written to the source address.	1. Set *Buffer1* with specific value. Call Mem.Write() to write *Buffer1* to *Address1* with count units. 2. Call CopyMem() to copy Mem from *Address1* to *Address1*+ BufferSize/2 with count units. 3. Call Mem.Read() to read data of *Address1*+BufferSize/2 to *Buffer2*. All units of *Buffer2* should be the same as *Buffer1*.
5.8.1.9.5	0x4081f6bf, 0xf332, 0x44de, 0xb8, 0x62, 0x19, 0xe5, 0xaa, 0xdb, 0x43, 0x7e	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem – CopyMem() between overlapping regions with destination address < source address returns EFI_SUCCESS.	1. Set *Buffer1* with specific value. Call Mem.Write() to write *Buffer1* to *Address1*+ BufferSize/2 with count units. 2. Call CopyMem() to copy Mem from *Address1*+ BufferSize/2 to *Address1* with count units. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.8.1.9.6	0x8fb4d613, 0x2bde, 0x4f40, 0x9c, 0x70, 0xe1, 0x60, 0x34, 0xdc, 0x3b, 0xbc	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem** – After call of **CopyMem()** between overlapping regions, the data in destination address should be the same as the buffer contents written to the source address.	1. Set *Buffer1* with specific value. Call **Mem.Write()** to write *Buffer1* to *Address1*+ **BufferSize**/2 with count units. 2. Call **CopyMem()** to copy Mem from *Address1*+ **BufferSize**/2 to *Address1* with count units. 3. Call **Mem.Read()** to read data of *Address1* to *Buffer2*. All units of *Buffer2* should be the same as *Buffer1*.
5.8.1.9.7	0x0bcb82fb, 0x7052, 0x4d0f, 0xad, 0x73, 0xd3, 0xe7, 0x25, 0xae, 0x46, 0xb5	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem – CopyMem()** with *Width* as **EfiPciWidthMaximum** returns **EFI_INVALID_PARAMETER**.	1. Call **CopyMem()** with *Width* as **EfiPciWidthMaximum**. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.9.8	0x9f7bf606, 0xf898, 0x42f2, 0xb7, 0x7f, 0xc1, 0x39, 0xa5, 0x90, 0x65, 0x6c	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem – CopyMem()** with *Width* as -1 returns **EFI_INVALID_PARAMETER**.	1. Call **CopyMem()** with *Width* as -1. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.9.9	0x5762a830, 0x4fd5, 0x4858, 0x82, 0x1f, 0x76, 0xab, 0x12, 0xe9, 0xa9, 0x80	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem – CopyMem()** with *Width* as **EfiPciWidthFifoUintX** returns **EFI_INVALID_PARAMETER**.	1. Call **CopyMem()** with *Width* as **EfiPciWidthFifoUintX**. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.9.10	0x09154449, 0xd6bc, 0x47b3, 0x8a, 0x47, 0x25, 0xd3, 0x08, 0x81, 0xa5, 0x0f	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem – CopyMem()** with *Width* as **EfiPciWidthFillUintX** returns **EFI_INVALID_PARAMETER**.	1. Call **CopyMem()** with *Width* as **EfiPciWidthFillUintX**. The return code should be **EFI_INVALID_PARAMETER**

Number	GUID	Assertion	Test Description
5.8.1.9.11	0x6ea5136c, 0x0060, 0x4e70, 0xa1, 0x7a, 0xc1, 0xf0, 0xbf, 0x9c, 0x74, 0x89	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem – CopyMem() with unsupported *Width* from profile returns EFI_INVALID_PARAMETER.	1. Call CopyMem() with unsupported *Width* from profile. The return code should be EFI_INVALID_PARAMETER.

10.1.10 Map()

Number	GUID	Assertion	Test Description
5.8.1.10.1	0xb5eadff4, 0x6bbc, 0x45a2, 0xb9, 0x05, 0x85, 0x49, 0x78, 0xf3, 0xa6, 0x27	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map– Map with EfiPciOperationBusMasterRead returns EFI_SUCCESS.	1. Allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterRead to map the address of *Buffer* to device address. The return code should be EFI_SUCCESS.
5.8.1.10.2	0x93950131, 0x0bc3, 0x429d, 0xad, 0x2d, 0x10, 0x47, 0x70, 0x76, 0x6c, 0xce	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map– Map with EfiPciOperationBusMasterRead returns non-0 *NumberOfBytes*.	1. Allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterRead to map the address of *Buffer* to device address. The return value of *NumberOfBytes* should not be 0.
5.8.1.10.3	0x1a041b96, 0x79ea, 0x4732, 0xb9, 0xaa, 0x1c, 0xd4, 0x3b, 0x8c, 0x36, 0xcc	[DELETED]	
5.8.1.10.4	0x11e33211, 0xbc86, 0x4d69, 0xb9, 0xdf, 0x2d, 0x0a, 0xb5, 0xa0, 0x94, 0x46	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map– Map() with EfiPciOperationBusMasterRead64 returns EFI_SUCCESS.	1. Allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterRead64 to map the address of *Buffer* to device address. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.8.1.10.5	0x42e6a8c6, 0x0b28, 0x422d, 0xae, 0x3d, 0x86, 0x4d, 0xbf, 0x7b, 0x55, 0xee	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map – Map() with EfiPciOperationBusMasterRead64 returns non-0 *NumberOfBytes*.	1. Allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterRead64 to map the address of *Buffer* to device address. The return value of *NumberOfBytes* should not be 0.
5.8.1.10.6	0x84f186ad, 0x3c1e, 0x46c4, 0x95, 0x52, 0xff, 0xd9, 0xdc, 0xbf, 0x80, 0x9d	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map – After Map() with EfiPciOperationBusMasterRead64, the data read from device address is the same as original data.	1. Allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterRead64 to map the address of *Buffer* to device address. The data read from device address must be the same as original data.
5.8.1.10.7	0xe10594a2, 0xfd97, 0x4383, 0x82, 0x5c, 0x62, 0x14, 0x54, 0x62, 0xd9, 0x5e	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map – Map() with EfiPciOperationBusMasterWrite returns EFI_SUCCESS.	1. Allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterWrite to map the address of *Buffer* to device address. The return code should be EFI_SUCCESS.
5.8.1.10.8	0x07e366fc, 0x5d2e, 0x474f, 0xba, 0xd3, 0xf8, 0xe4, 0x0a, 0x50, 0xf1, 0xd9	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map – Map() with EfiPciOperationBusMasterWrite returns non-0 *NumberOfBytes*.	1. Allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterWrite to map the address of *Buffer* to device address. The return value of *NumberOfBytes* should not be 0.
5.8.1.10.9	0xbceb0ddc, 0x1145, 0x4fcd, 0x89, 0x1c, 0x53, 0x2f, 0x71, 0xb1, 0xf4, 0xe7	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map – Map() with EfiPciOperationBusMasterWrite does not change data in host address.	1. Allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterWrite to map the address of *Buffer* to device address. Data in *Buffer* should not be changed.
5.8.1.10.10	0x5288b979, 0x9a17, 0x474a, 0xaf, 0xa0, 0x68, 0x61, 0x88, 0x48, 0xb3, 0xc1	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map – Map() with EfiPciOperationBusMasterWrite64 returns EFI_SUCCESS.	1. Allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterWrite64 to map the address of *Buffer* to device address. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.8.1.10.11	0x65d95c94, 0xd3b9, 0x4e4b, 0x88, 0x38, 0x49, 0x96, 0x0d, 0xb8, 0xfb, 0x24	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map** – **Map()** with **EfiPciOperationBusMasterWrite** returns non-0 *NumberOfBytes*.	1. Allocate memory to *Buffer*. 2. Call **Map()** with **EfiPciOperationBusMasterWrite**64 to map the address of *Buffer* to device address. The return value of *NumberOfBytes* should not be 0.
5.8.1.10.12	0x29fc59bc, 0x9f0d, 0x463d, 0xb4, 0x4a, 0x5a, 0xd2, 0x2d, 0x11, 0xa2, 0x26	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map** – **Map()** with **EfiPciOperationBusMasterWrite**64 does not change data in host address.	1. Allocate memory to *Buffer*. 2. Call **Map()** with **EfiPciOperationBusMasterWrite**64 to map the address of *Buffer* to device address. Data in *Buffer* should not be changed.
5.8.1.10.13	0xb674ab5a, 0xc030, 0x4832, 0x9d, 0x69, 0xbb, 0x18, 0x27, 0xb3, 0x39, 0x8e	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map** – **Map()** with **EfiPciOperationBusMasterCommonBuffer** returns **EFI_SUCCESS**.	1. Call **AllocateBuffer()** to allocate memory to *Buffer*. 2. Call **Map()** with **EfiPciOperationBusMasterCommonBuffer** to map the address of *Buffer* to device address. The return code should be **EFI_SUCCESS**.
5.8.1.10.14	0xebb4be23, 0x25c7, 0x46ce, 0xb8, 0x52, 0xde, 0xc7, 0x18, 0x2a, 0xc2, 0x07	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map** – **Map()** with **EfiPciOperationBusMasterCommonBuffer** returns non-0 *NumberOfBytes*.	1. Call **AllocateBuffer()** to allocate memory to *Buffer*. 2. Call **Map()** with **EfiPciOperationBusMasterCommonBuffer** to map the address of *Buffer* to device address. The return value of *NumberOfBytes* should not be 0.
5.8.1.10.18	0x8120df74, 0xae1e, 0x47f9, 0xaa, 0x45, 0x8e, 0x70, 0xa7, 0xe3, 0x31, 0x19	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map** – **Map()** with **EfiPciOperationBusMasterCommonBuffer**64 returns **EFI_SUCCESS**.	1. Call **AllocateBuffer()** to allocate memory to *Buffer*. 2. Call **Map()** with **EfiPciOperationBusMasterCommonBuffer**64 to map the address of *Buffer* to device address. The return code should be **EFI_SUCCESS**.
5.8.1.10.19	0xb93854ce, 0x5237, 0x492f, 0xbd, 0x55, 0x27, 0xd3, 0x82, 0xc1, 0xce, 0x53	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map** – **Map()** with **EfiPciOperationBusMasterCommonBuffer**64 returns non-0 *NumberOfBytes*.	1. Call **AllocateBuffer()** to allocate memory to *Buffer*. 2. Call **Map()** with **EfiPciOperationBusMasterCommonBuffer**64 to map the address of *Buffer* to device address. The return value of *NumberOfBytes* should not be 0.

Number	GUID	Assertion	Test Description
5.8.1.10.20	0x3ec7dc5b, 0x3c99, 0x47e1, 0x87, 0xff, 0xb2, 0x4d, 0x08, 0x95, 0x04, 0x96	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map – After Map() with EfiPciOperationBusMasterCommonBuffer64, the data read from device address is the same as original data.	1. Call AllocateBuffer() to allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterCommonBuffer64 to map the address of *Buffer* to device address. The data read from device address must be the same as original data.
5.8.1.10.21	0xb4df6e6e, 0x4e30, 0x457e, 0xa1, 0xf8, 0x39, 0xf4, 0x52, 0xf6, 0x11, 0x2f	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map – After Map() with EfiPciOperationBusMasterCommonBuffer64, the data in original host address remains in sync with mapped device address.	1. Call AllocateBuffer() to allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterCommonBuffer64 to map the address of *Buffer* to device address. 3. Call BS.SetMem() to change contents of mapped device address. Data in host address should change also and be equal to data in device address.
5.8.1.10.22	0xc4451e9d, 0x538e, 0x4cda, 0xa7, 0xa6, 0x0c, 0xa1, 0x50, 0x06, 0x03, 0x87	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map – After Map() with EfiPciOperationBusMasterCommonBuffer64, the data in mapped device address remains in sync with original host address.	1. Call AllocateBuffer() to allocate memory to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterCommonBuffer64 to map the address of *Buffer* to device address. 3. Call BS.SetMem() to change contents of host address. Data in mapped device address should change also and be equal to data in device address.
5.8.1.10.23	0xc79ed36f, 0xe0b3, 0x426c, 0x85, 0xc1, 0x7d, 0xfe, 0xb8, 0xcf, 0xdf, 0x07	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map – Map() with invalid Operation as EfiPciOperationMaximum returns EFI_INVALID_PARAMETER.	1. Call Map() with invalid Operation: EfiPciOperationMaximum. The return code should be EFI_INVALID_PARAMETER.
5.8.1.10.24	0x04b07426, 0x3d17, 0x4f18, 0x8b, 0x1c, 0xbd, 0x59, 0xae, 0x99, 0xe5, 0xf8	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map – Map() with invalid Operation as -1 returns EFI_INVALID_PARAMETER.	1. Call Map() with invalid Operation: -1. The return code should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.8.1.10.25	0xf8a42643, 0x912a, 0x4731, 0xb9, 0x04, 0x47, 0xbc, 0x87, 0x7f, 0xdd, 0xcf	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map**– **Map()** with *HostAddress* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **Map()** with *HostAddress* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.8.1.10.26	0x13513dbf, 0xc4da, 0x4952, 0xa4, 0x37, 0x44, 0x22, 0x28, 0x13, 0xdb, 0xfd	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map**– **Map()** with *NumberOfBytes* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **Map()** with *NumberOfBytes* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.8.1.10.27	0x8bfb7a69, 0xd816, 0x4315, 0xbe, 0x27, 0xe2, 0xa9, 0x03, 0x44, 0x69, 0x8e	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map**– **Map()** with *DeviceAddress* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **Map()** with *DeviceAddress* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.8.1.10.28	0x6fe65b18, 0x7638, 0x4584, 0xb9, 0x5f, 0x90, 0x2c, 0x0f, 0x80, 0xf6, 0x9b	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map**– **Map()** with *Mapping* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **Map()** with *Mapping* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.8.1.10.29	0xd6b631c7, 0xd459, 0x40cd, 0xa1, 0xca, 0x6d, 0x28, 0x7b, 0x61, 0xaa, 0xd9	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map**– **Map()** with **EfiPciOperationBusMasterCommonBuffer** and *HostAddress* + *NumberofBytes* > 4GB returns **EFI_UNSUPPORTED**.	1. Call **Map()** with *HostAddress* + *NumberofBytes* > 4GB. The return code should be **EFI_UNSUPPORTED**.
5.8.1.10.30	0x04030971, 0xedb2, 0x498b, 0x84, 0x94, 0xf0, 0x19, 0x24, 0x28, 0xd4, 0x14	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Map**– **Map()** with **EfiPciOperationBusMasterCommonBuffer**64 and *HostAddress* + *NumberofBytes* > 4GB returns **EFI_UNSUPPORTED**.	1. Call **Map()** with *HostAddress* + *NumberofBytes* > 4GB. The return code should be **EFI_UNSUPPORTED**.

10.1.11 Unmap()

Number	GUID	Assertion	Test Description
5.8.1.11.1	0xb4a084d7, 0x48de, 0x48de, 0x97, 0xa0, 0x27, 0x10, 0x07, 0x9f, 0xcc, 0x04	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of EfiPciOperationBusMasterRead returns EFI_SUCCESS.	1. Call Map() with EfiPciOperationBusMasterRead to map the address of *Buffer* to device address.. 2. Call Unmap() to release resources of mapping. The return code should be EFI_SUCCESS.
5.8.1.11.2	0xa4ef56f6, 0x597b, 0x47a4, 0xa3, 0xed, 0x00, 0xba, 0x87, 0xcd, 0x47, 0xd8	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of EfiPciOperationBusMasterRead does not change contents in host address.	1. Set specific value to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterRead to map the address of *Buffer* to device address. 3. Call Unmap with mapping value gotten from Map(). The data in *Buffer* should remain unchanged.
5.8.1.11.3	0xd211369e, 0x2b2d, 0x4d95, 0xa7, 0x30, 0x7c, 0x7c, 0xf5, 0xd6, 0xfc, 0x13	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of EfiPciOperationBusMasterRead64 returns EFI_SUCCESS.	1. Call Map() with EfiPciOperationBusMasterRead 64 to map the address of *Buffer* to device address.. 2. Call Unmap() to release resources of mapping. The return code should be EFI_SUCCESS.
5.8.1.11.4	0xa32ec004, 0x1e89, 0x4553, 0xac, 0x80, 0x9d, 0x3b, 0x14, 0xe6, 0x09, 0x49	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of EfiPciOperationBusMasterRead64 does not change contents in host address.	1. Set specific value to *Buffer*. 2. Call Map() with EfiPciOperationBusMasterRead 64 to map the address of *Buffer* to device address. 3. Call Unmap with mapping value gotten from Map(). The data in *Buffer* should remain unchanged.
5.8.1.11.5	0x8a2ffff4, 0x186b, 0x4624, 0xa5, 0x4a, 0x1a, 0x8f, 0xaf, 0xe4, 0x06, 0x2a	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of EfiPciOperationBusMasterWrite returns EFI_SUCCESS.	1. Call Map() with EfiPciOperationBusMasterWrite to map the address of *Buffer* to device address. 2. Call Unmap() to release resources of mapping. The return code should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.8.1.11.6	0x8874b727, 0x7a35, 0x4e6e, 0x96, 0x19, 0x7e, 0x5b, 0x22, 0xcb, 0x3f, 0xf8	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap()** with mapping value gotten from **Map()** of **EfiPciOperationBusMasterWrite** does not change contents in host address.	1. Set specific value to *Buffer*. 2. Call **Map()** with **EfiPciOperationBusMasterWrite** to map the address of *Buffer* to device address. 3. Call Unmap with mapping value gotten from **Map()**. The data in *Buffer* should remain unchanged.
5.8.1.11.7	0xffd39873, 0xa3da, 0x49fd, 0xae, 0x87, 0x5c, 0x09, 0xb5, 0xa1, 0x01, 0x73	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap()** with mapping value gotten from **Map()** of **EfiPciOperationBusMasterWrite** returns **EFI_SUCCESS**.	1. Call **Map()** with **EfiPciOperationBusMasterWrite** to map the address of *Buffer* to device address. 2. Call **Unmap()** to release resources of mapping. The return code should be **EFI_SUCCESS**.
5.8.1.11.8	0xd8eedc25, 0xea92, 0x4d1b, 0x8f, 0xe7, 0x7c, 0xb1, 0x87, 0xb2, 0xc0, 0xa6	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap()** with mapping value gotten from **Map()** of **EfiPciOperationBusMasterWrite**, does not change contents in host address.	1. Set specific value to the Buffer. 2. Call **Map()** with **EfiPciOperationBusMasterWrite** to map the address of the Buffer to the device address. 3. Call **Unmap()** with mapping value gotten from **Map()**. The data in the Buffer should remain unchanged.
5.8.1.11.9	0xe543e036, 0x3948, 0x4773, 0xa8, 0x0e, 0x89, 0x2c, 0xd3, 0xcc, 0xf0, 0xdf	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap()** with mapping value gotten from **Map()** of **EfiPciOperationBusMasterCommonBuffer** returns **EFI_SUCCESS**.	1. Call **Map()** with **EfiPciOperationBusMasterCommonBuffer** to map the address of the Buffer to the device address. 2. Call **Unmap()** to release resources of mapping. The return code should be **EFI_SUCCESS**.
5.8.1.11.10	0xd2368593, 0x122a, 0x41e7, 0x83, 0x34, 0x65, 0x7e, 0x78, 0xed, 0x12, 0xbc	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap()** with mapping value gotten from **Map()** of **EfiPciOperationBusMasterCommonBuffer** does not change contents in host address.	1. Call **AllocateBuffer()** to allocate memory to the Buffer. 2. Call **Map()** with **EfiPciOperationBusMasterCommonBuffer** to map the address of the Buffer to the device address. 3. Call **Unmap()** with mapping value gotten from **Map()**. The data in the Buffer should remain unchanged.

Number	GUID	Assertion	Test Description
5.8.1.11.11	0x9356285b, 0x21b2, 0x40a3, 0x95, 0xed, 0xd6, 0xfe, 0x27, 0x5a, 0x2b, 0xba	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of EfiPciOperationBusMasterCommonBuffer64 returns EFI_SUCCESS.	1. Call Map() with EfiPciOperationBusMasterCommonBuffer64 to map the address of the Buffer to the device address. 2. Call Unmap() to release resources of mapping. The return code should be EFI_SUCCESS.
5.8.1.11.12	0x0c44017c, 0x078d, 0x475c, 0x90, 0x0c, 0x4a, 0x36, 0xe6, 0x8b, 0x72, 0x04	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Unmap – Unmap() with mapping value gotten from Map() of EfiPciOperationBusMasterCommonBuffer64 does not change contents in host address	1. Call AllocateBuffer() to allocate memory to the Buffer. 2. Call Map() with EfiPciOperationBusMasterCommonBuffer64 to map the address of the Buffer to the device address. 3. Call Unmap with mapping value gotten from Map(). The data in the Buffer should remain unchanged.

10.1.12 AllocateBuffer()

Number	GUID	Assertion	Test Description
5.8.1.12.1	0x58a99166, 0xfdbe, 0x4963, 0xb9, 0x56, 0x00, 0x4f, 0x97, 0xcc, 0xe5, 0x20	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.AllocateBuffer – AllocateBuffer() with valid parameter returns EFI_SUCCESS.	1. Call AllocateBuffer() with valid parameter. The return code should be EFI_SUCCESS.
5.8.1.12.2	0x193efb14, 0x0c2a, 0x494d, 0xa2, 0xfc, 0xe1, 0x28, 0xb0, 0xe7, 0xb6, 0x5c	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.AllocateBuffer – AllocateBuffer() with invalid memory types -1 returns EFI_INVALID_PARAMETER.	1. Call AllocateBuffer() with invalid memory types -1. The return code should be EFI_INVALID_PARAMETER
5.8.1.12.3	0x08d81bb3, 0x1db0, 0x4ce3, 0x8e, 0xe0, 0xa6, 0x7c, 0x46, 0xf1, 0xa8, 0x9b	EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.AllocateBuffer – AllocateBuffer() with invalid memory types returns EFI_INVALID_PARAMETER.	1. Call AllocateBuffer() with invalid memory types. The return code should be EFI_INVALID_PARAMETER

Number	GUID	Assertion	Test Description
5.8.1.12.4	0x66bd765c, 0x6b86, 0x4a29, 0xbe, 0x88, 0x10, 0xab, 0xfe, 0x5a, 0xef, 0xbd	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.AllocateBuffer – AllocateBuffer()** with *HostAddress* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **AllocateBuffer()** with *HostAddress* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**
5.8.1.12.5	0xf2e8d30e, 0x40d8, 0x4823, 0x97, 0xb2, 0x08, 0x32, 0x11, 0x9f, 0x78, 0xd3	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.AllocateBuffer – AllocateBuffer()** with unsupported *Attributes* returns **EFI_UNSUPPORTED**.	1. Call **AllocateBuffer()** with unsupported *Attributes*. The return code should be **EFI_UNSUPPORTED**.

10.1.13 FreeBuffer()

Number	GUID	Assertion	Test Description
5.8.1.13.1	0xf2ec6740, 0x6416, 0x4890, 0xaf, 0xe6, 0xad, 0x67, 0x91, 0xf0, 0x22, 0xaf	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.FreeBuffer – FreeBuffer()** with valid parameter returns **EFI_SUCCESS**.	1. Call **AllocateBuffer()** to allocate memory to buffer. 2. Call **FreeBuffer()** to free buffer memory. The return code should be **EFI_SUCCESS**.

10.1.14 Flush()

Number	GUID	Assertion	Test Description
5.8.1.14.1	0x8ce74cd6, 0x0409, 0x4513, 0x98, 0xdd, 0x3d, 0x0f, 0x96, 0x97, 0x4f, 0xe8	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Flush – Flush()** with valid parameter returns **EFI_SUCCESS**.	1. Call **Flush()** with valid parameter. The return code should be **EFI_SUCCESS**.

10.1.15 GetAttributes()

Number	GUID	Assertion	Test Description
5.8.1.15.1	0x8e661c40, 0xf56f, 0x4ce8, 0x8e, 0x7e, 0xf4, 0x07, 0x28, 0x57, 0xf9, 0x5b	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.GetAttributes` – `GetAttributes()` to get current attributes and supported attributes returns `EFI_SUCCESS`.	1. Call `GetAttributes()` to get current attributes and supported attributes. The return code should be `EFI_SUCCESS`.
5.8.1.15.2	0x54d94c0e, 0x70d7, 0x4a7a, 0x9e, 0x81, 0xf5, 0xb1, 0x63, 0x05, 0x93, 0xbe	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.GetAttributes` – Current attributes must within Supported attributes.	1. Call `GetAttributes()` to get current attributes and supported attributes. 2. Current attributes must within Supported attributes.
5.8.1.15.3	0x727cabec, 0x1a1b, 0x4e9d, 0xb1, 0xde, 0x3b, 0x3e, 0xda, 0x55, 0x84, 0x44	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.GetAttributes` – `GetAttributes()` to only get current attributes returns `EFI_SUCCESS`.	1. Call `GetAttributes()` to only get current attributes. The return code should be `EFI_SUCCESS`.
5.8.1.15.4	0x66fb3230, 0xa799, 0x4efe, 0x89, 0xfa, 0xbf, 0x86, 0xdf, 0x23, 0xb0, 0xf7	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.GetAttributes` – The second call of `GetAttributes()` returns the same current attributes as the first time.	1. Call `GetAttributes()` to get current attributes and supported attributes. 2. Call `GetAttributes()` for the second time to only get current attributes. It should return the same current attribute as the first time.
5.8.1.15.5	0x2176073a, 0x7dfa, 0x463a, 0xa2, 0xf1, 0xab, 0xba, 0x92, 0x42, 0xe0, 0xea	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.GetAttributes` – `GetAttributes()` to only get supported attributes returns `EFI_SUCCESS`.	1. Call `GetAttributes()` to only get supported attributes. The return code should be `EFI_SUCCESS`.
5.8.1.15.6	0x5a5c6253, 0x1202, 0x4abd, 0x95, 0x6f, 0x23, 0x0a, 0x1b, 0x2f, 0x45, 0xc0	`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.GetAttributes` – The second call of `GetAttributes()` returns the same supported attributes as the first time.	1. Call `GetAttributes()` to get current attributes and supported attributes. 2. Call `GetAttributes()` for the second time to only get supported attributes. It should return the same supported attribute as the first time.

Number	GUID	Assertion	Test Description
5.8.1.15.7	0x8f25b1c3, 0x4571, 0x4101, 0x95, 0xf1, 0x36, 0xc1, 0xe5, 0x83, 0xc0, 0x23	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.GetAttributes – GetAttributes()** with both *Attributes* and *Supports* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **GetAttributes()** with both *Attributes* and *Supports* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**

10.1.16 SetAttributes()

Number	GUID	Assertion	Test Description
5.8.1.16.1	0xb9ee4bd9, 0x5a92, 0x4521, 0xbf, 0xaa, 0x80, 0x7f, 0x8b, 0x20, 0xac, 0xaa	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** to set supported attributes returns **EFI_SUCCESS**.	1. Call **GetAttributes()** to get supported attributes. 2. Call **SetAttributes()** to set supported attributes. The return code should be **EFI_SUCCESS**.
5.8.1.16.2	0x1dbb0bee, 0x7ebf, 0x4a3f, 0xa8, 0xaf, 0xb8, 0x24, 0x76, 0x29, 0xd6, 0x7c	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** to set supported attributes changes current attributes as expected.	1. Call **GetAttributes()** to get supported attributes. 2. Call **SetAttributes()** to set supported attributes. 3. Call **GetAttributes()** to get current attributes. The supported attributes bits should be set.
5.8.1.16.3	0x697e0d03, 0xca02, 0x4a21, 0x87, 0xf6, 0xd5, 0xd5, 0xeb, 0xb3, 0xab, 0xdb	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** to set supported attributes that require a resource returns **EFI_SUCCESS**.	1. Call **GetAttributes()** to get supported attributes. 2. Call **SetAttributes()** to set **MEMORY_WRITE_COMBINE**, **MEMORY_CACHED** or **MEMORY_DISABLE** if they are supported. The return code should be **EFI_SUCCESS**.
5.8.1.16.4	0x1f27d46e, 0x53b4, 0x4687, 0xaa, 0x9a, 0x5d, 0x46, 0xfb, 0x05, 0xa3, 0x65	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** to set supported attributes changes current attributes as expected.	1. Call **GetAttributes()** to get supported attributes. 2. Call **SetAttributes()** to set **MEMORY_WRITE_COMBINE**, **MEMORY_CACHED** or **MEMORY_DISABLE** if they are supported. 3. Call **GetAttributes()** to get current attributes. The supported attribute bits specified by **SetAttributes()** should be set.

Number	GUID	Assertion	Test Description
5.8.1.16.5	0x405511dd, 0x38b4, 0x4aed, 0x9a, 0x7e, 0x18, 0xaa, 0xd1, 0x21, 0x67, 0x68	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** with unsupported attributes that do not need resources returns **EFI_UNSUPPORTED**.	1. Call **GetAttributes()** to get current attributes and supported attributes. 2. Call **SetAttributes()** with unsupported attributes that do not need resources. The return code should be **EFI_UNSUPPORTED**.
5.8.1.16.6	0x0150f584, 0x775b, 0x422d, 0xb3, 0xd7, 0xb8, 0x0d, 0x34, 0x56, 0x26, 0x47	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** with unsupported attributes that need resources returns **EFI_UNSUPPORTED**.	1. Call **GetAttributes()** to get current attributes and supported attributes. 2. Call **SetAttributes()** with unsupported attributes that need resources. The return code should be **EFI_UNSUPPORTED**.
5.8.1.16.7	0xdbf3baef, 0x35e9, 0x4d10, 0x8a, 0xbb, 0xcc, 0xca, 0x70, 0x5e, 0x99, 0x86	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** with unsupported attributes does not change current attributes.	1. Call **GetAttributes()** to get current attributes and supported attributes. 2. Call **SetAttributes()** with unsupported attributes that not resource. 3. Call **GetAttributes()** to get current attributes. It should remain unchanged.
5.8.1.16.8	0x186fee52, 0x7b8d, 0x4589, 0x8d, 0x87, 0x8e, 0x4f, 0x6b, 0x67, 0x9c, 0x6c	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_WRITE_COMBINE** and *ResourceBase* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_WRITE_COMBINE** and *ResourceBase* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.8.1.16.9	0x5a06217c, 0xcbf1, 0x4faa, 0x94, 0x04, 0x3b, 0xaf, 0x39, 0x6d, 0x04, 0x1d	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_WRITE_COMBINE** and *ResourceLength* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_WRITE_COMBINE** and *ResourceLength* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.8.1.16.10	0x7d1e8194, 0x0732, 0x4ca0, 0xac, 0x50, 0xdb, 0x62, 0x18, 0xe0, 0x69, 0xdd	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_CACHED** and *ResourceBase* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_CACHED** and *ResourceBase* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.8.1.16.11	0x037c66ae, 0x79a4, 0x4909, 0x93, 0xa4, 0xa6, 0xb7, 0xb8, 0xee, 0x58, 0xd6	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_CACHED** and *ResourceLength* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_CACHED** and *ResourceLength* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.8.1.16.12	0x117de9ad, 0xbc79, 0x49c2, 0xa7, 0x0f, 0x80, 0xc8, 0x80, 0x48, 0x6c, 0x91	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_DISABLE** and *ResourceBase* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_DISABLE** and *ResourceBase* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.8.1.16.13	0x363d5f12, 0x4c82, 0x4117, 0xa7, 0x6c, 0xc3, 0xd3, 0x70, 0x8f, 0xdb, 0xda	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.SetAttributes – SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_DISABLE** and *ResourceLength* as **NULL** returns **EFI_INVALID_PARAMETER**.	1. Call **SetAttributes()** with **EFI_PCI_ATTRIBUTE_MEMORY_DISABLE** and *ResourceLength* as **NULL**. The return code should be **EFI_INVALID_PARAMETER**.

10.1.17 Configuration()

Number	GUID	Assertion	Test Description
5.8.1.17.1	0xe65742bb, 0x7693, 0x4de1, 0xb0, 0x7b, 0x74, 0xfd, 0x64, 0x43, 0x6b, 0xf5	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Configuration – Configuration()** to get the resource list returns **EFI_SUCCESS**.	1. Call **Configuration()** to get the resource list. The return code should be **EFI_SUCCESS**.
5.8.1.17.2	0xa5982933, 0x6b43, 0x4947, 0xb0, 0x29, 0xa8, 0xd5, 0x66, 0x72, 0xaa, 0xce	**EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.Configuration** – *Resource* returned by Configuration points to a valid ACPI 2.0 QWord descriptor.	1. Call **Configuration()** to get the *Resource* list. The return *Resource* should be a valid ACPI 2.0 QWord descriptor.

10.2 EFI_PCI_IO_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_PCI_ IO_ PROTOCOL Section.

Configuration

Some checkpoints in the **EFI_PCI_IO_PROTOCOL** test are device related. If the user needs to check the protocol on the specified device, the related profile needs to be updated to provide the specified information about this device.

For the format of the profile, please refer to EFI_PCI_IO_PROTOCOL Test Profile.

10.2.1 PollMem()

Number	GUID	Assertion	Test Description
5.8.2.1.1	0xaef16eb4, 0x40ad, 0x4dcf, 0x8c, 0x57, 0x20, 0x92, 0xa7, 0x43, 0xa9, 0x78	**EFI_PCI_IO_PROTOCOL.PollMem – PollMem()** with valid value returns **EFI_SUCCESS**.	1. Call **Mem.Write()** to set the Alternate Value on the address. 2. Start a 3 second timer event. The event handler writes the Target Value to the address. 3. Call **PollMem()** for the Target Value with *Delay* as 5 seconds on the address -- **PollMem()** must return **EFI_SUCCESS** with *Result* as the Target Value.

Number	GUID	Assertion	Test Description
5.8.2.1.2	0x6e8a67fe, 0x4ad1, 0x4317, 0xa6, 0xfe, 0x76, 0x88, 0x02, 0x49, 0x0f, 0xbc	`EFI_PCI_IO_PROTOCOL.PollMem - PollMem()` with valid value again returns `EFI_SUCCESS`.	1. Call `Mem.Write()` to set the Alternate Value on the address. 2. Start a 3 second timer event. The event handler writes the Target Value to the address. 3. Call `PollMem()` for the Target Value with *Delay* as 5 seconds on the address -- `PollMem()` must return `EFI_SUCCESS` with *Result* as the Target Value. 4. Call `PollMem()` for the Target Value again on the address. -- `PollMem()` must return `EFI_SUCCESS` with *Result* as the expected value.
5.8.2.1.3	0x3b2cfc3e, 0xf167, 0x4c1f, 0x99, 0x8e, 0x2b, 0xca, 0x0b, 0x17, 0x6d, 0x39	`EFI_PCI_IO_PROTOCOL.PollMem - PollMem()` with delay equals 0 and invalid destination address, returns `EFI_SUCCESS`.	1. Call `Mem.Write()` to set the Alternate Value on the address. 2. `PollMem()` for the Target Value on the address with *Delay* as 0. -- `PollMem()` must return `EFI_SUCCESS`, with *Result* as the Alternate Value.
5.8.2.1.4	0x600c99fb, 0x31d0, 0x4a94, 0x8e, 0xa3, 0xbd, 0x59, 0x54, 0xd0, 0xa5, 0x2b	`EFI_PCI_IO_PROTOCOL.PollMem - PollMem()` with 5 seconds delay and invalid destination address, returns `EFI_TIMEOUT`.	1. Call `Mem.Write()` to set the Alternate Value on the address. 2. `PollMem()` for the Target on the address with *Delay* as 5 seconds. – `PollMem()` must return `EFI_TIMEOUT`, with *Result* as the Alternate Value.
5.8.2.1.5	0x5a9e8b1e, 0xdc0d, 0x461f, 0x9f, 0xd5, 0xf4, 0x4c, 0xb9, 0x6e, 0xff, 0xfa	`EFI_PCI_IO_PROTOCOL.PollMem` - With *Width* as `EfiPciWidthMaximum`, the return status is `EFI_INVALID_PARAMETER`.	1. Call `PollMem()` with *Width* as `EfiPciWidthMaximum`. The return status must be `EFI_INVALID_PARAMETER`.
5.8.2.1.6	0x3c29ad4d, 0x8bad, 0x4862, 0xab, 0x3a, 0x9b, 0xde, 0xee, 0xd6, 0x2e, 0x19	`EFI_PCI_IO_PROTOCOL.PollMem` - With *Width* as `EfiPciWidthFifoUintX`, the return status is `EFI_INVALID_PARAMETER`.	1. Call `PollMem()` with *Width* as `EfiPciWidthFifoUintX`. The return status must be `EFI_INVALID_PARAMETER`.
5.8.2.1.7	0xb9b9ebdc, 0x09e9, 0x4cc6, 0xaf, 0x45, 0xf1, 0xae, 0x28, 0x06, 0x17, 0x70	`EFI_PCI_IO_PROTOCOL.PollMem` - With *Width* as `EfiPciWidthFillUintX`, the return status is `EFI_INVALID_PARAMETER`.	1. Call `PollMem()` with *Width* as `EfiPciWidthFillUintX`. The return status must be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.8.2.1.8	0x9007c300, 0x0782, 0x4f3e, 0xae, 0x40, 0xd5, 0x9d, 0x95, 0xce, 0x55, 0xf6	**EFI_PCI_IO_PROTOCOL.PollMem** – With *Width* as -1, the return status is **EFI_INVALID_PARAMETER**	1. Call **PollMem()** with *Width* as -1. The return status must be **EFI_INVALID_PARAMETER**
5.8.2.1.9	0xdb14a663, 0x3a39, 0x4cf1, 0x90, 0xe6, 0x7a, 0xfe, 0x00, 0x6c, 0x66, 0xe2	**EFI_PCI_IO_PROTOCOL.PollMem** – With *Result* as **NULL**, the return status is **EFI_INVALID_PARAMETER**.	1. Call **PollMem()** with *Result* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.1.10	0x47e2f242, 0xf876, 0x46ed, 0x9c, 0x91, 0x82, 0xd6, 0xd6, 0xb6, 0x7d, 0xb5	**EFI_PCI_IO_PROTOCOL.PollMem** – With *Offset* beyond the range of BAR, the return status is **EFI_UNSUPPORTED**.	1. Call **PollMem()** with *Offset* beyond the range of BAR. The return status must be **EFI_UNSUPPORTED**.
5.8.2.1.11	0x02b6ac92, 0x4984, 0x42d8, 0xab, 0xda, 0xb1, 0x87, 0x8e, 0xa0, 0xd6, 0xc8	**EFI_PCI_IO_PROTOCOL.PollMem** – With invalid BAR Index the return status is **EFI_UNSUPPORTED**.	1. Call **PollMem()** with invalid BAR Index. The return status must be **EFI_UNSUPPORTED**.
5.8.2.1.12	0x668ccc4e, 0xb0b2, 0x4980, 0xab, 0x43, 0xff, 0xfd, 0x11, 0x83, 0x91, 0x75	**EFI_PCI_IO_PROTOCOL.PollMem** – With Io BAR Index the return status is **EFI_UNSUPPORTED**.	1. Call **PollMem()** with Io BAR Index. The return status must be **EFI_UNSUPPORTED**.
5.8.2.1.13	0x47a63a3d, 0xa134, 0x4a04, 0xb0, 0xd2, 0x10, 0xf1, 0x64, 0x88, 0xb0, 0xfb	**EFI_PCI_IO_PROTOCOL.PollMem** – With invalid *Width* the return status is **EFI_INVALID_PARAMETER**.	1. Call **PollMem()** with invalid *Width*. The return status must be **EFI_INVALID_PARAMETER**.

10.2.2 PollIo()

Number	GUID	Assertion	Test Description
5.8.2.2.1	0x6dfeb4fd, 0xdd98, 0x40db, 0x8e, 0x42, 0x67, 0x8a, 0xfb, 0x92, 0x6a, 0xe9	**EFI_PCI_IO_PROTOCOL.PollIo** – **PollIo()** with valid value returns **EFI_SUCCESS**.	1. Call **Mem.Write()** to set the Alternate Value on the address. 2. Start a 3 second timer event. The event handler writes the Target Value to the address. 3. Call **PollMem()** for the Target Value with *Delay* as 5 seconds on the address -- **PollMem()** must return **EFI_SUCCESS** with *Result* as the Target Value.
5.8.2.2.2	0x427eb5db, 0x6e41, 0x4b01, 0xad, 0xb0, 0x31, 0xff, 0xd9, 0x99, 0x6a, 0x5b	**EFI_PCI_IO_PROTOCOL.PollIo** – **PollIo()** with valid value again returns **EFI_SUCCESS**.	1. Call **Io.Write()** to set the Alternate Value on the address. 2. Start a 3 second timer event. The event handler writes the Target Value to the address. 3. Call **PollIo()** for the Target Value with *Delay* as 5 seconds on the address -- **PollIo()** must return **EFI_SUCCESS** with *Result* as the Target Value. 4. Call **PollIo()** for the Target Value again on the address. -- **PollIo()** must return **EFI_SUCCESS** with *Result* as the expected value
5.8.2.2.3	0xdff400ef, 0x9e72, 0x448f, 0xad, 0x6b, 0xb1, 0x34, 0x25, 0x45, 0xc7, 0x02	**EFI_PCI_IO_PROTOCOL.PollIo** – **PollIo()** with delay equal 0 and invalid destination address returns **EFI_SUCCESS**.	1. Call **Io.Write()** to set the Alternate Value on the address. 2. **PollIo()** for the Target Value on the address with *Delay* as 0. -- **PollIo()** must return **EFI_SUCCESS**, with *Result* as the Alternate Value.
5.8.2.2.4	0x6071974c, 0x35c0, 0x4599, 0xa6, 0x53, 0xe4, 0xbe, 0xc7, 0x34, 0xf7, 0x2c	**EFI_PCI_IO_PROTOCOL.PollIo** – **PollIo()** with 5 seconds delay and invalid destination address returns **EFI_TIMEOUT**.	1. Call **Io.Write()** to set the Alternate Value on the address. 2. **PollIo()** for the Target on the address with *Delay* as 5 seconds. – **PollIo()** must return **EFI_TIMEOUT**, with *Result* as the Alternate Value.
5.8.2.2.5	0xc113fe3f, 0x0fae, 0x4266, 0xbf, 0xb4, 0xfd, 0x41, 0xed, 0x41, 0xea, 0x39	**EFI_PCI_IO_PROTOCOL.PollIo** – With *Width* as **EfiPciWidthMaximum** the return status is **EFI_INVALID_PARAMETER**.	1. Call **PollIo()** with *Width* as **EfiPciWidthMaximum**. The return status must be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.8.2.2.6	0x11466e1f, 0xd7e6, 0x4622, 0x84, 0x73, 0xfd, 0x57, 0xbf, 0x2f, 0x8f, 0x8e	**EFI_PCI_IO_PROTOCOL.PollIo** – With *Width* as **EfiPciWidthFifoUintX** the return status is **EFI_INVALID_PARAMETER**.	1. Call **PollIo()** with *Width* as **EfiPciWidthFifoUintX**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.2.7	0x251113eb, 0x968c, 0x4c70, 0xbf, 0xa0, 0x0d, 0xf6, 0x74, 0x7f, 0xfa, 0x9a	**EFI_PCI_IO_PROTOCOL.PollIo** – With *Width* as **EfiPciWidthFillUintX** the return status is **EFI_INVALID_PARAMETER**.	1. Call **PollIo()** with *Width* as **EfiPciWidthFillUintX**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.2.8	0xc6e532e8, 0xacc8, 0x4d48, 0x84, 0x69, 0xfd, 0xb0, 0xc1, 0xe0, 0xe5, 0x34	**EFI_PCI_IO_PROTOCOL.PollIo** – With *Width* as -1 the return status is **EFI_INVALID_PARAMETER**.	1. Call **PollIo()** with *Width* as -1. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.2.9	0xdd0e653a, 0x9da8, 0x4f32, 0x9d, 0x0a, 0xe3, 0x29, 0xe1, 0x17, 0x19, 0x0e	**EFI_PCI_IO_PROTOCOL.PollIo** – With *Result* as **NULL** the return status is **EFI_INVALID_PARAMETER**.	1. Call **PollIo()** with *Result* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.2.10	0xda044ef5, 0xe73b, 0x415c, 0xaf, 0x03, 0xaf, 0x3c, 0xb0, 0x00, 0x3f, 0x45	**EFI_PCI_IO_PROTOCOL.PollIo** – With *Offset* beyond the range of BAR the return status is **EFI_UNSUPPORTED**.	1. Call **PollIo()** with *Offset* beyond the range of BAR. The return status must be **EFI_UNSUPPORTED**.
5.8.2.2.11	0x0929e753, 0x7659, 0x4b6b, 0x80, 0x1a, 0x8b, 0xd6, 0xb6, 0x37, 0x4d, 0xf6	**EFI_PCI_IO_PROTOCOL.PollIo** – With invalid BAR Index the return status is **EFI_UNSUPPORTED**.	1. Call **PollIo()** with invalid BAR Index. The return status must be **EFI_UNSUPPORTED**.

Number	GUID	Assertion	Test Description
5.8.2.2.12	0x64e878f6, 0xa53d, 0x4b4f, 0xa3, 0xca, 0x18, 0x9e, 0x37, 0x23, 0x4a, 0x99	**EFI_PCI_IO_PROTOCOL.PollIo** – With Mem BAR Index the return status is **EFI_UNSUPPORTED**.	1. Call **PollIo()** with Mem BAR Index. The return status must be **EFI_UNSUPPORTED**.
5.8.2.2.13	0xf2e6563e, 0x0881, 0x4efc, 0xae, 0x69, 0x6d, 0x08, 0xdf, 0x1c, 0xb2, 0x80	**EFI_PCI_IO_PROTOCOL.PollIo** – With invalid *Width* the return status is **EFI_INVALID_PARAMETER**.	1. Call **PollIo()** with invalid *Width*. The return status must be **EFI_INVALID_PARAMETER**.

10.2.3 Mem.Read()

Number	GUID	Assertion	Test Description
5.8.2.3.1	0xa52d8d69, 0x77cb, 0x4012, 0x9d, 0x3f, 0xfa, 0x19, 0xe3, 0x2f, 0x17, 0x6c	**EFI_PCI_IO_PROTOCOL.Mem.Read** – **Mem.Read()** reads data out and returns **EFI_SUCCESS**.	1. Allocate a buffer that matches the size of the address range. 2. Call **Mem.Read()** to fill in the buffer with the predefined data units The return status should be **EFI_SUCCESS**.
5.8.2.3.2	0x44e5c09e, 0xce91, 0x419d, 0xbe, 0xaf, 0xd6, 0x60, 0x73, 0xdf, 0x4e, 0xe3	**EFI_PCI_IO_PROTOCOL.Mem.Read** – **Mem.Read()** read out the data from the address space and returns **EFI_SUCCESS**.	1. Allocate a buffer that matches the size of the address range. 2. Call **Mem.Read()** to fill in the buffer with the predefined data units. 3. Call **Mem.Write()** to write the buffer into the address range. 4. Call **Mem.Read()** to read out the data in destination address range. The return status should be **EFI_SUCCESS**.
5.8.2.3.3	0x8ac05fc7, 0x0378, 0x4b5e, 0xba, 0x48, 0xb8, 0x53, 0x3d, 0x9e, 0xf2, 0x4c	**EFI_PCI_IO_PROTOCOL.Mem.Read** - The data read out is the same as that written in.	1. Allocate a buffer that matches the size of the address range. 2. Call **Mem.Read()** to fill in the buffer with the predefined data units. 3. Call **Mem.Write()** to write the buffer into the address range. 4. Call **Mem.Read()** to read out the data in destination address range. 5. Compare the data read out with data written in.

Number	GUID	Assertion	Test Description
5.8.2.3.4	0xca3a1290, 0x652f, 0x490c, 0x8a, 0x3f, 0xea, 0x94, 0x45, 0xa4, 0xd3, 0x81	**EFI_PCI_IO_PROTOCOL.Mem.Read – Mem.Read()** reads out the data with **EfiPciIoWidthFifoX** returns **EFI_SUCCESS**.	1. Allocate a buffer that matches the size of the address range. 2. Call **Mem.Read()** to fill in the buffer with the predefined data units. 3. Call **Mem.Write()** to write the buffer into the address range. 4. Call **Mem.Read()** to read out the data using **EfiPciIoWidthFifoUintX**. The return status should be **EFI_SUCCESS**.
5.8.2.3.5	0x99bb7423, 0xa29c, 0x442e, 0x9a, 0x29, 0x7b, 0xf8, 0xf1, 0x88, 0xa8, 0x7e	**EFI_PCI_IO_PROTOCOL.Mem.Read** - With **EfiPciIoWidthFifoX**, the data read out is the same as the first data unit.	1. Allocate a buffer that matches the size of the address range. 2. Call **Mem.Read()** to fill in the buffer with the predefined data units. 3. Call **Mem.Write()** to write the buffer into the address range. 4. Call **Mem.Read()** to read out the data using **EfiPciIoWidthFifoUintX**. 5. Compare the each data unit in the buffer with the data at the Starting Address of the address range.
5.8.2.3.6	0x4b9fef07, 0x3a4f, 0x40a0, 0xad, 0x43, 0xd1, 0x6a, 0x59, 0x8c, 0x22, 0x04	**EFI_PCI_IO_PROTOCOL.Mem.Read – Mem.Read()** reads out the data with **EfiPciIoWidthFillX** and returns **EFI_SUCCESS**.	1. Allocate a bufferthat matches the size of the address range. 2. Call **Mem.Read()** to fill in the buffer with the predefined data units. 3. Call **Mem.Write()** to write the buffer into the address range. 4. Call **Mem.Read()** to read out the data using **EfiPciIoWidthFillUintX**. The return status should be **EFI_SUCCESS**.
5.8.2.3.7	0xd0bb89cc, 0x3838, 0x48bd, 0xb9, 0xd8, 0x1b, 0x8e, 0x3d, 0xef, 0x77, 0xd5	**EFI_PCI_IO_PROTOCOL.Mem.Read** - With **EfiPciIoWidthFillX**, the data read out from the first unit in buffer equals the last unit in the address space.	1. Allocate a bufferthat matches the size of the address range. 2. Call **Mem.Read()** to fill in the buffer with the predefined data units. 3. Call **Mem.Write()** to write the buffer into the address range. 4. Call **Mem.Read()** to read out the data using **EfiPciIoWidthFillUintX**. 5. Compare the first data unit in the output buffer with the last data unit in the address range. Compare other data units in the output buffer with the original data.

Number	GUID	Assertion	Test Description
5.8.2.3.8	0xd282dcc9, 0x004f, 0x4733, 0xb2, 0xa6, 0xb5, 0x56, 0x6b, 0x4c, 0xaf, 0x91	**EFI_PCI_IO_PROTOCOL.Mem.Read** - With *Width* as **EfiPciIoWidthMaximum** the return status is **EFI_INVALID_PARAMETER**.	1. Call **Mem.Read()** with *Width* as **EfiPciWidthMaximum**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.3.9	0x927ad37d, 0x5ee5, 0x4d7a, 0x9f, 0x2e, 0x49, 0x9d, 0x7a, 0x49, 0x87, 0xb9	**EFI_PCI_IO_PROTOCOL.Mem.Read** - With invalid *Width* type -1 the return status is **EFI_INVALID_PARAMETER**.	1. Call **Mem.Read()** with *Width* as -1. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.3.10	0x99d41dcf, 0x75ee, 0x48bf, 0xac, 0x3d, 0x86, 0xce, 0xe6, 0x67, 0x11, 0x1c	**EFI_PCI_IO_PROTOCOL.Mem.Read** - With *Buffer* as **NULL** the return status is **EFI_INVALID_PARAMETER**.	1. Call **Mem.Read()** with *Buffer* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.3.11	0xa6d04a84, 0x2808, 0x48c7, 0xa0, 0x0b, 0xc2, 0xd6, 0xab, 0xad, 0xd5, 0x91	**EFI_PCI_IO_PROTOCOL.Mem.Read** - With address out of BAR range the return status is **EFI_UNSUPPORTED**.	1. Call **Mem.Read()** with address out of BAR range. The return status must be **EFI_UNSUPPORTED**.
5.8.2.3.12	0xe8417927, 0xe158, 0x4094, 0x90, 0xf6, 0x03, 0x15, 0xf7, 0x2f, 0x61, 0xdc	**EFI_PCI_IO_PROTOCOL.Mem.Read** - With address out of BAR range the return status is **EFI_UNSUPPORTED**.	1. Call **Mem.Read()** with address out of BAR range. The return status must be **EFI_UNSUPPORTED**.
5.8.2.3.13	0x80720d1b, 0xa3dd, 0x465f, 0x8d, 0xe8, 0x9b, 0x6b, 0xb9, 0x64, 0x76, 0xda	**EFI_PCI_IO_PROTOCOL.Mem.Read** - With invalid BAR Index the return status is **EFI_UNSUPPORTED**.	1. Call **Mem.Read()** with invalid BAR Index. The return status must be **EFI_UNSUPPORTED**.

Number	GUID	Assertion	Test Description
5.8.2.3.14	0x3b9e11c1, 0x6fea, 0x4742, 0x81, 0xfd, 0xf2, 0xfb, 0xd6, 0x9c, 0xb6, 0xba	`EFI_PCI_IO_PROTOCOL.Mem.Read` - With Io `Type` BAR the return status is `EFI_UNSUPPORTED`.	1. Call `Mem.Read()` with Io `Type` BAR. The return status must be `EFI_UNSUPPORTED`.
5.8.2.3.15	0xa043ffdf, 0x568b, 0x4128, 0x80, 0xf8, 0x61, 0x29, 0x0c, 0xd8, 0x8d, 0x57	`EFI_PCI_IO_PROTOCOL.Mem.Read` - With invalid *`Width`* the return status is `EFI_INVALID_PARAMETER`.	1. Call `Mem.Read()` with invalid *`Width`*. The return status must be `EFI_INVALID_PARAMETER`.

10.2.4 Mem.Write()

Number	GUID	Assertion	Test Description
5.8.2.4.1	0x5847e586, 0x1f02, 0x466c, 0xa8, 0x33, 0x27, 0x23, 0x0d, 0x8d, 0xd9, 0xfd	`EFI_PCI_IO_PROTOCOL.Mem.Write` - `Mem.Write()` writes data to the memory address space and returns `EFI_SUCCESS`.	1. Allocate a buffer that matches the size of the address range. 2. Call `Mem.Read()` to fill in the buffer with the predefined data units. 3. Call `Mem.Write()` to write the buffer into the address range The return status should be `EFI_SUCCESS`.
5.8.2.4.2	0x6790de90, 0x56b2, 0x456e, 0x8e, 0x7a, 0xd1, 0x65, 0x77, 0xb9, 0xce, 0x39	`EFI_PCI_IO_PROTOCOL.Mem.Write` - The data read out is the same as that written in.	1. Allocate a buffer that matches the size of the address range. 2. Call `Mem.Read()` to fill in the buffer with the predefined data units. 3. Call `Mem.Write()` to write the buffer into the address range. 4. Call `Mem.Read()` to read out the data in destination address range. 5. Compare the data read out with data written in.
5.8.2.4.3	0x148a380b, 0xdbe0, 0x496b, 0xbd, 0x51, 0x56, 0xe6, 0xde, 0xcc, 0xf7, 0xca	`EFI_PCI_IO_PROTOCOL.Mem.Write` - `Mem.Write()` writes the data with `EfiPciIoWidthFifoX` and returns `EFI_SUCCESS`.	1. Allocate a buffer that matches the size of the address range. 2. Call `Mem.Read()` to fill in the buffer with the predefined data units. 3. Call `Mem.Write()` to write to the starting address of address range using `EfiPciIoWidthFifoUintX`. The return status should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.8.2.4.4	0xf641e745, 0x9f3c, 0x42bf, 0x94, 0x23, 0x04, 0x20, 0x56, 0x46, 0x4b, 0x6b	EFI_PCI_IO_PROTOCOL.Mem.Write - With EfiPciIoWidthFifoX, the first data unit is the same as the last data unit.	1. Allocate a buffer that matches the size of the address range. 2. Call Mem.Read() to fill in the buffer with the predefined data units 3. Call Mem.Write() to write to the starting address of address range using EfiPciIoWidthFifoUintX. 4. Call Mem.Read() to read out the data using EfiPciIoWidthFifoUintX. 5. Compare the data in the starting address with the last data unit. Compare other data units with original data.
5.8.2.4.5	0xbb3f0bad, 0x6680, 0x4aaa, 0xbe, 0x39, 0x70, 0xe4, 0x13, 0x02, 0xf8, 0x5d	EFI_PCI_IO_PROTOCOL.Mem.Write - Mem.Write() writes the data with EfiPciIoWidthFillX and returns EFI_SUCCESS.	1. Allocate a buffer that matches the size of the address range. 2. Call Mem.Read() to fill in the buffer with the predefined data units. 3. Call Mem.Write() to write to address range using EfiPciIoWidthFillUintX. The return status should be EFI_SUCCESS.
5.8.2.4.6	0x787dfda9, 0xcbfd, 0x4aae, 0x82, 0x98, 0xb1, 0xd4, 0x74, 0x15, 0x89, 0xb2	EFI_PCI_IO_PROTOCOL.Mem.Write - With EfiPciIoWidthFillX, all the data units read out are the same as the first data unit in the address space.	1. Allocate a buffer that matches the size of the address range. 2. Call Mem.Read() to fill in the buffer with the predefined data units. 3. Call Mem.Write() to write to address range using EfiPciIoWidthFillUintX. 4. Call Mem.Read() to read out the data using EfiPciIoWidthFillUintX. 5. Compare all the data units with the first data unit.
5.8.2.4.7	0x2d6920fd, 0x05a9, 0x480b, 0x8c, 0x74, 0x2a, 0xfc, 0x0f, 0xa7, 0x83, 0x3a	EFI_PCI_IO_PROTOCOL.Mem.Write - Mem.Write() writes back the Data and returns EFI_SUCCESS.	1. Allocate a buffer that matches the size of the address range. 2. Call Mem.Read() to fill in the buffer with the predefined data units. 3. Call Mem.Write() to write the buffer into the address range. 4. Call Mem.Read() to read out the data in destination address range. 5. Call Mem.Write() to write the data back. The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.8.2.4.8	0x4fe0f156, 0x0cb2, 0x464a, 0xb1, 0xbd, 0x23, 0x14, 0x9e, 0x3e, 0x09, 0x60	**EFI_PCI_IO_PROTOCOL.Mem.Write** - With *Width* as **EfiPciIoWidthMaximum** the return status is **EFI_INVALID_PARAMETER**.	1. Call **Mem.Write()** with *Width* as **EfiPciWidthMaximum**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.4.9	0xb868ce7a, 0xfff0, 0x4c3c, 0x98, 0x00, 0xf5, 0xc7, 0xc2, 0x13, 0xaa, 0x09	**EFI_PCI_IO_PROTOCOL.Mem.Write** - With invalid *Width* type -1 the return status is **EFI_INVALID_PARAMETER**.	1. Call **Mem.Write()** with *Width* as -1. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.4.10	0x2fe9804a, 0xa418, 0x40b7, 0xa6, 0x8c, 0xaa, 0x40, 0xc3, 0xe6, 0x2f, 0x84	**EFI_PCI_IO_PROTOCOL.Mem.Write** - With *Buffer* as **NULL** the return status is **EFI_INVALID_PARAMETER**.	1. Call **Mem.Write()** with *Buffer* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.4.11	0xdac9a8dc, 0x172e, 0x4c6d, 0xb2, 0xe7, 0xf1, 0x65, 0x94, 0xfe, 0x89, 0x39	**EFI_PCI_IO_PROTOCOL.Mem.Write** - With address out of BAR range the return status is **EFI_UNSUPPORTED**.	1. Call **Mem.Write()** with address out of BAR range. The return status must be **EFI_UNSUPPORTED**.
5.8.2.4.12	0x99fca122, 0xd9dc, 0x4d3b, 0xbb, 0xb0, 0x2a, 0xf5, 0x3d, 0xd1, 0x39, 0x0e	**EFI_PCI_IO_PROTOCOL.Mem.Write** - With address out of BAR range the return status is **EFI_UNSUPPORTED**.	1. Call **Mem.Write()** with address out of BAR range. The return status must be **EFI_UNSUPPORTED**.
5.8.2.4.13	0xd5c1f492, 0x5dbf, 0x4b4d, 0x9e, 0x09, 0xd5, 0x1a, 0x23, 0x47, 0x37, 0xcc	**EFI_PCI_IO_PROTOCOL.Mem.Write** - With invalid BAR Index the return status is **EFI_UNSUPPORTED**.	1. Call **Mem.Write()** with invalid BAR Index. The return status must be **EFI_UNSUPPORTED**.

Number	GUID	Assertion	Test Description
5.8.2.4.14	0x1af1b78c, 0x8ca2, 0x4146, 0x97, 0x69, 0x94, 0x29, 0xac, 0x48, 0x11, 0x65	EFI_PCI_IO_PROTOCOL.Mem.Write - With Io Type BAR the return status is EFI_UNSUPPORTED.	1. Call Mem.Write() with Io Type BAR. The return status must be EFI_UNSUPPORTED.
5.8.2.4.15	0xa154d373, 0xc12b, 0x4939, 0xa3, 0xb2, 0xc0, 0x14, 0xc1, 0x09, 0xd3, 0x68	EFI_PCI_IO_PROTOCOL.Mem.Write - With invalid *Width* the return status is EFI_INVALID_PARAMETER.	1. Call Mem.Write() with invalid *Width*. The return status must be EFI_INVALID_PARAMETER.

10.2.5 Io.Read()

Number	GUID	Assertion	Test Description
5.8.2.5.1	0x36e0b044, 0x2b2b, 0x484b, 0xb4, 0x80, 0x85, 0x99, 0xa9, 0x99, 0xa9, 0x35	EFI_PCI_IO_PROTOCOL.Io.Read - Io.Read() reads data out and returns EFI_SUCCESS.	1. Allocate a buffer that matches the size of the address range. 2. Call Io.Read() to fill in the buffer with the predefined data units. The return status should be EFI_SUCCESS.
5.8.2.5.2	0xe65f66cb, 0xb1cb, 0x4a7a, 0x8c, 0x68, 0xb2, 0x0c, 0x69, 0x58, 0xdd, 0x6a	EFI_PCI_IO_PROTOCOL.Io.Read - Io.Read() reads out the data from Io address space and returns EFI_SUCCESS.	1. Allocate a buffer that matches the size of the address range. 2. Call Io.Read() to fill in the buffer with the predefined data units. 3. Call Io.Write() to write the buffer into the address range. 4. Call Io.Read() to read out the data in destination address range. The return status should be EFI_SUCCESS.
5.8.2.5.3	0xec27b5c5, 0x59fb, 0x4954, 0x9c, 0x51, 0xad, 0xf4, 0x46, 0x7e, 0xe7, 0xe6	EFI_PCI_IO_PROTOCOL.Io.Read - The data read out is the same as that written in.	1. Allocate a buffer that matches the size of the address range. 2. Call Io.Read() to fill in the buffer with the predefined data units. 3. Call Io.Write() to write the buffer into the address range. 4. Call Io.Read() to read out the data in destination address range. 5. Compare the data read out with data written in.

Number	GUID	Assertion	Test Description
5.8.2.5.4	0x271e3b70, 0x6617, 0x4f5f, 0xb5, 0x12, 0x46, 0xb1, 0xe3, 0x1d, 0xe3, 0x79	**EFI_PCI_IO_PROTOCOL.Io.Read** – **Io.Read()** reads out the data with **EfiPciIoWidthFifoUintX** and returns **EFI_SUCCESS**.	1. Allocate a buffer that matches the size of the address range. 2. Call **Io.Read()** to fill in the buffer with the predefined data units. 3. Call **Io.Write()** to write the buffer into the address range. 4. Call **Io.Read()** to read out the data using **EfiPciIoWidthFifoUintX**. The return status should be **EFI_SUCCESS**.
5.8.2.5.5	0xccf3806e, 0x25fa, 0x4697, 0xb7, 0x08, 0x8d, 0xc1, 0x5b, 0x47, 0xba, 0x8d	**EFI_PCI_IO_PROTOCOL.Io.Read** – All the data read out with **EfiPciIoWidthFifoX** is equal with the first data unit.	1. Allocate a buffer that matches the size of the address range. 2. Call **Io.Read()** to fill in the buffer with the predefined data units. 3. Call **Io.Write()** to write the buffer into the address range. 4. Call **Io.Read()** to read out the data using **EfiPciIoWidthFifoUintX**. 5. Compare each data unit in the buffer with the data at the Starting Address of the address range.
5.8.2.5.6	0x080ea87f, 0xc265, 0x4a33, 0xab, 0x09, 0x78, 0xf8, 0x94, 0x58, 0x03, 0x2b	**EFI_PCI_IO_PROTOCOL.Io.Read** – **Io.Read()** reads out the data with **EfiPciIoWidthFillX** and returns **EFI_SUCCESS**.	1. Allocate a buffer that matches the size of the address range. 2. Call **Io.Read()** to fill in the buffer with the predefined data units. 3. Call **Io.Write()** to write the buffer into the address range. 4. Call **Io.Read()** to read out the data using **EfiPciIoWidthFillUintX**. The return status should be **EFI_SUCCESS**.
5.8.2.5.7	0x543fda6a, 0x651a, 0x4560, 0xaf, 0xfd, 0x6a, 0x95, 0x76, 0x54, 0x07, 0x30	**EFI_PCI_IO_PROTOCOL.Io.Read** – Reads out the data with **EfiPciIoWidthFillX**. The first data unit eqauls the last data unit in destination address.	1. Allocate a buffer that matches the size of the address range. 2. Call **Io.Read()** to fill in the buffer with the predefined data units. 3. Call **Io.Write()** to write the buffer into the address range. 4. Call **Io.Read()** to read out the data using **EfiPciIoWidthFillUintX**. 5. Compare the first data unit in the output buffer with the last data unit in the address range. Compare other data units in the output buffer with the original data.
5.8.2.5.8	0x65b3c515, 0x1fe1, 0x4021, 0xb2, 0x02, 0xdd, 0xc9, 0x7a, 0x0d, 0xb2, 0x11	**EFI_PCI_IO_PROTOCOL.Io.Read** – With *Width* as **EfiPciIoWidthMaximum**, the return status is **EFI_INVALID_PARAMETER**.	1. Call **Io.Read()** with *Width* as **EfiPciWidthMaximum**. The return status must be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.8.2.5.9	0x8ef36cf9, 0x84b7, 0x4961, 0xaa, 0xcc, 0xf7, 0x41, 0x21, 0x96, 0xc0, 0xdc	**EFI_PCI_IO_PROTOCOL.Io.Read** – With invalid *Width* type -1, the return status is **EFI_INVALID_PARAMETER**.	1. Call **Io.Read()** with *Width* as -1. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.5.10	0x4cac979d, 0x6b8c, 0x458c, 0xb3, 0xca, 0x75, 0x30, 0x6f, 0x59, 0xa9, 0xb7	**EFI_PCI_IO_PROTOCOL.Io.Read** – With *Buffer* as **NULL**, the return status is **EFI_INVALID_PARAMETER**.	1. Call **Io.Read()** with *Buffer* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.5.11	0xaf51e635, 0x89c8, 0x49db, 0xa7, 0x11, 0xb6, 0xc6, 0xb8, 0x96, 0xf9, 0x79	**EFI_PCI_IO_PROTOCOL.Io.Read** – With address out of BAR range, the return status is **EFI_UNSUPPORTED**.	1. Call **Io.Read()** with address out of BAR range. The return status must be **EFI_UNSUPPORTED**.
5.8.2.5.12	0x8d878934, 0x8270, 0x48a7, 0xad, 0x51, 0x65, 0xa8, 0x8d, 0xac, 0x36, 0x93	**EFI_PCI_IO_PROTOCOL.Io.Read** – With address out of BAR range, the return status is **EFI_UNSUPPORTED**.	1. Call **Io.Read()** with address out of BAR range. The return status must be **EFI_UNSUPPORTED**.
5.8.2.5.13	0x53cc0e1e, 0xf3aa, 0x4f15, 0xaf, 0xec, 0xc3, 0x04, 0x8f, 0x0f, 0xa5, 0xb8	**EFI_PCI_IO_PROTOCOL.Io.Read** – With invalid BAR Index, the return status is **EFI_UNSUPPORTED**.	1. Call **Io.Read()** with invalid BAR Index. The return status must be **EFI_UNSUPPORTED**.
5.8.2.5.14	0x2fb4dc13, 0xb3f5, 0x4e19, 0xba, 0xe2, 0x76, 0x47, 0x10, 0x4d, 0xf7, 0x79	**EFI_PCI_IO_PROTOCOL.Io.Read** – With Mem **Type** BAR, the return status is **EFI_UNSUPPORTED**.	1. Call **Io.Read()** with Mem **Type** BAR. The return status must be **EFI_UNSUPPORTED**.

Number	GUID	Assertion	Test Description
5.8.2.5.15	0x44b6de4e, 0xc968, 0x4d97, 0xbe, 0x01, 0x3f, 0xf3, 0xdd, 0xfc, 0x53, 0xe0	EFI_PCI_IO_PROTOCOL.Io.Read – With invalid *Width*, the return status is EFI_INVALID_PARAMETER.	1. Call Io.Read() with invalid *Width*. The return status must be EFI_INVALID_PARAMETER.

10.2.6 Io.Write()

Number	GUID	Assertion	Test Description
5.8.2.6.1	0x7b1ed2c6, 0xa84e, 0x4858, 0xa7, 0x8b, 0xa6, 0xd9, 0x32, 0x03, 0x22, 0xbe	EFI_PCI_IO_PROTOCOL.Io.Write – Io.Write() writes data to Io address space, returns EFI_SUCCESS.	1. Allocate a buffer that matches the size of the address range. 2. Call Io.Read() to fill in the buffer with the predefined data units. 3. Call Io.Write() to write the buffer into the address range. The return status should be EFI_SUCCESS.
5.8.2.6.2	0xd1704c13, 0xd0df, 0x4f7c, 0xb8, 0xb6, 0xd9, 0x5b, 0xe6, 0xdc, 0xea, 0x87	EFI_PCI_IO_PROTOCOL.Io.Write – The data read equals the data written in.	1. Allocate a buffer that matches the size of the address range. 2. Call Io.Read() to fill in the buffer with the predefined data units. 3. Call Io.Write() to write the buffer into the address range. 4. Call Io.Read() to read out the data in destination address range. 5. Compare the data read out with data written in.
5.8.2.6.3	0xafb5070c, 0x1d07, 0x4df3, 0x9a, 0xd5, 0x6f, 0x91, 0x7e, 0x48, 0xc5, 0xed	EFI_PCI_IO_PROTOCOL.Io.Write – Io.Write() writes the data with EfiPciIoWidthFifoX and returns EFI_SUCCESS.	1. Allocate a buffer that matches the size of the address range. 2. Call Io.Read() to fill in the buffer with the predefined data units. 3. Call Io.Write() to write to the starting address of address range using EfiPciIoWidthFifoUintX. The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.8.2.6.4	0xee8d1797, 0x1474, 0x4d80, 0x85, 0x82, 0x35, 0x78, 0x61, 0x2b, 0x26, 0x01	**EFI_PCI_IO_PROTOCOL.Io.Write** – With **EfiPciIoWidthFifoUintX**, the first data unit is the last data unit.	1. Allocate a buffer that matches the size of the address range. 2. Call **Io.Read()** to fill in the buffer with the predefined data units. 3. Call **Io.Write()** to write to the starting address of address range using **EfiPciIoWidthFifoUintX**. 4. Call **Io.Read()** to read out the data using **EfiPciIoWidthFifoUintX**. 5. Compare the data in the starting address with the last data unit. Compare other data units with original data.
5.8.2.6.5	0x4a6378ee, 0x5058, 0x42b2, 0x8a, 0x03, 0x1f, 0x1c, 0xff, 0x05, 0x15, 0xcc	**EFI_PCI_IO_PROTOCOL.Io.Write** – **Io.Write()** writes the data with **EfiPciIoWidthFillX** and returns **EFI_SUCCESS**.	1. Allocate a buffer that matches the size of the address range. 2. Call **Io.Read()** to fill in the buffer with the predefined data units. 3. Call **Io.Write()** to write to address range using **EfiPciIoWidthFillUintX**. The return status should be **EFI_SUCCESS**.
5.8.2.6.6	0x15b81460, 0xbc5e, 0x4be3, 0x9c, 0xc5, 0xb6, 0x59, 0x06, 0x83, 0x28, 0x5e	**EFI_PCI_IO_PROTOCOL.Io.Write** – With **EfiPciIoWidthFillUintX**, all the data units read out are the same as the first data units in the address space.	1. Allocate a buffer that matches the size of the address range. 2. Call **Io.Read()** to fill in the buffer with the predefined data units. 3. Call **Io.Write()** to write to address range using **EfiPciIoWidthFillUintX**. 4. Call **Io.Read()** to read out the data using **EfiPciIoWidthFillUintX**. 5. Compare all the data units with the first data unit.
5.8.2.6.7	0x8e854d61, 0x2048, 0x446f, 0xb6, 0x47, 0x3c, 0x37, 0x17, 0x14, 0xac, 0xf6	**EFI_PCI_IO_PROTOCOL.Io.Write** – **Io.Write()** writes back the data and returns **EFI_SUCCESS**.	1. Allocate a buffer that matches the size of the address range. 2. Call **Io.Read()** to fill in the buffer with the predefined data units. 3. Call **Io.Write()** to write the buffer into the address range. 4. Call **Io.Read()** to read out the data in destination address range. 5. Call **Io.Write()** to write the data back. The return status should be **EFI_SUCCESS**.
5.8.2.6.8	0xb96af4e4, 0x988f, 0x4362, 0x8c, 0x63, 0x1f, 0x08, 0xeb, 0xfd, 0xa3, 0x5f	**EFI_PCI_IO_PROTOCOL.Io.Write** – With *Width* as **EfiPciIoWidthMaximum**, return status is **EFI_INVALID_PARAMETER**.	1. Call **Io.Write()** with *Width* as **EfiPciWidthMaximum**. The return status must be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.8.2.6.9	0x8cb298d4, 0x5831, 0x48ce, 0x87, 0x8d, 0xf3, 0xf8, 0x20, 0x62, 0xea, 0xf3	**EFI_PCI_IO_PROTOCOL.Io.Write** – With invalid *Width* type -1, the return status is **EFI_INVALID_PARAMETER**.	1. Call **Io.Write()** with *Width* as -1. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.6.10	0x175943ee, 0x4d2d, 0x480f, 0xa3, 0xf1, 0x88, 0xc9, 0x7c, 0x6b, 0x04, 0x77	**EFI_PCI_IO_PROTOCOL.Io.Write** – With *Buffer* as **NULL**, the return status is **EFI_INVALID_PARAMETER**.	1. Call **Io.Write()** with *Buffer* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.6.11	0x4617468a, 0xd228, 0x4a84, 0x88, 0x56, 0x21, 0x8c, 0x3f, 0x39, 0x46, 0xd1	**EFI_PCI_IO_PROTOCOL.Io.Write** – With address out of BAR range the return status is **EFI_UNSUPPORTED**.	1. Call **Io.Write()** with address out of BAR range. The return status must be **EFI_UNSUPPORTED**.
5.8.2.6.12	0x03dd4807, 0xe461, 0x4e97, 0x9d, 0xf9, 0xea, 0x73, 0x38, 0x15, 0xd5, 0x62	**EFI_PCI_IO_PROTOCOL.Io.Write** – With address out of BAR range the return status is **EFI_UNSUPPORTED**.	1. Call **Io.Write()** with address out of BAR range. The return status must be **EFI_UNSUPPORTED**.
5.8.2.6.13	0xd6b9d51d, 0x2676, 0x4449, 0xa4, 0xd6, 0x3d, 0xa0, 0x17, 0x36, 0x2e, 0xa6	**EFI_PCI_IO_PROTOCOL.Io.Write** – With invalid BAR Index the return status is **EFI_UNSUPPORTED**.	1. Call **Io.Write()** with invalid BAR Index. The return status must be **EFI_UNSUPPORTED**.
5.8.2.6.14	0x648a859d, 0x3b72, 0x41a6, 0x86, 0xad, 0x3f, 0xff, 0x66, 0xd8, 0x61, 0x2f	**EFI_PCI_IO_PROTOCOL.Io.Write** – With Mem **Type** BAR the return status is **EFI_UNSUPPORTED**.	1. Call **Io.Write()** with Mem **Type** BAR. The return status must be **EFI_UNSUPPORTED**.

Number	GUID	Assertion	Test Description
5.8.2.6.15	0xfdc9b3f3, 0x2b80, 0x4a99, 0xa9, 0xba, 0xa5, 0x5e, 0xf9, 0xf8, 0x26, 0x19	**EFI_PCI_IO_PROTOCOL.Io.Write** – With invalid *Width* the return status is **EFI_INVALID_PARAMETER**.	1. Call **Io.Write()** with invalid *Width*. The return status must be **EFI_INVALID_PARAMETER**.

10.2.7 Pci.Read()

Number	GUID	Assertion	Test Description
5.8.2.7.1	0xea2a44d0, 0xc8d1, 0x465b, 0xb5, 0x50, 0x58, 0xd6, 0xef, 0x4e, 0x38, 0xd4	**EFI_PCI_IO_PROTOCOL.Pci.Read** – **Pci.Read()** reads data out into backup buffer and returns **EFI_SUCCESS**.	1. Allocate a buffer that matches the size of the address range. 2. Call **Pci.Read()** to fill in the buffer with the predefined data units. The return status should be **EFI_SUCCESS**.
5.8.2.7.2	0xe30bb837, 0x1d06, 0x4ee2, 0x80, 0x85, 0x18, 0xd4, 0x6b, 0x1c, 0x99, 0x66	**EFI_PCI_IO_PROTOCOL.Pci.Read** – **Pci.Read()** reads out the data from PCI configuration space and returns **EFI_SUCCESS**.	1. Allocate a buffer that matches the size of the address range. 2. Call **Pci.Read()** to fill in the buffer with the predefined data units. 3. Call **Pci.Write()** to write the buffer into the address range. 4. Call **Pci.Read()** to read out the data in destination address range. The return status should be **EFI_SUCCESS**.
5.8.2.7.3	0x2f9274d9, 0x7a14, 0x492f, 0x87, 0xc0, 0x40, 0x81, 0x4f, 0x66, 0x1b, 0xb4	**EFI_PCI_IO_PROTOCOL.Pci.Read** – The data read out from the PCI configuration space with **PciIoWidthUintX** equals the data written in.	1. Allocate a buffer that matches the size of the address range. 2. Call **Pci.Read()** to fill in the buffer with the predefined data units. 3. Call **Pci.Write()** to write the buffer into the address range. 4. Call **Pci.Read()** to read out the data in destination address range. 5. Compare the data read out with data written in.

Number	GUID	Assertion	Test Description
5.8.2.7.4	0x59ba5b67, 0x9e17, 0x4b60, 0xb5, 0x79, 0x5f, 0xd3, 0x26, 0x16, 0xe6, 0x6a	EFI_PCI_IO_PROTOCOL.Pci.Read - Pci.Read() reads out the data with EfiPciIoWidthFifoX and returns EFI_SUCCESS.	1. Allocate a buffer that matches the size of the address range. 2. Call Pci.Read() to fill in the buffer with the predefined data units. 3. Call Pci.Write() to write the buffer into the address range. 4. Call Pci.Read() to read out the data using EfiPciIoWidthFifoUintX. The return status should be EFI_SUCCESS.
5.8.2.7.5	0xd3b49ee4, 0x131a, 0x4fa3, 0xab, 0x81, 0x9f, 0x86, 0x33, 0xdf, 0x2d, 0xc7	EFI_PCI_IO_PROTOCOL.Pci.Read - Reads out the data with EfiPciIoWidthFifoX. The data read out is the same as the first data unit.	1. Allocate a buffer that matches the size of the address range. 2. Call Pci.Read() to fill in the buffer with the predefined data units. 3. Call Pci.Write() to write the buffer into the address range. 4. Call Pci.Read() to read out the data using EfiPciIoWidthFifoUintX. 5. Compare the each data unit in the buffer with the data at the Starting Address of the address range.
5.8.2.7.6	0x6e5881b2, 0x262d, 0x41ec, 0xa8, 0xd4, 0xcf, 0x28, 0x71, 0x1e, 0x5c, 0x15	EFI_PCI_IO_PROTOCOL.Pci.Read - Pci.Read() reads out the data with EfiPciIoWidthFillX and returns EFI_SUCCESS.	1. Allocate a buffer that matches the size of the address range. 2. Call Pci.Read() to fill in the buffer with the predefined data units. 3. Call Pci.Write() to write the buffer into the address range. 4. Call Pci.Read() to read out the data using EfiPciIoWidthFillUintX. The return status should be EFI_SUCCESS.
5.8.2.7.7	0x4595bbca, 0xbad7, 0x417f, 0xaf, 0x8d, 0x37, 0xec, 0x32, 0xf8, 0x03, 0x80	EFI_PCI_IO_PROTOCOL.Pci.Read - Reads out the data with EfiPciIoWidthFillX. The first data unit equals the last data unit in Destination address range.	1. Allocate a buffer that matches the size of the address range. 2. Call Pci.Read() to fill in the buffer with the predefined data units. 3. Call Pci.Write() to write the buffer into the address range. 4. Call Pci.Read() to read out the data using EfiPciIoWidthFillUintX. 5. Compare the first data unit in the output buffer with the last data unit in the address range. Compare other data units in the output buffer with the original data.
5.8.2.7.8	0x94d0a3d8, 0x7b61, 0x4147, 0xad, 0x9a, 0xea, 0xbb, 0x5f, 0x30, 0x59, 0xc2	EFI_PCI_IO_PROTOCOL.Pci.Read - With *Width* as EfiPciIoWidthMaximum, the return status is EFI_INVALID_PARAMETER.	1. Call Pci.Read() with *Width* as EfiPciWidthMaximum. The return status must be EFI_INVALID_PARAMETER

Number	GUID	Assertion	Test Description
5.8.2.7.9	0x18cf01fe, 0xa703, 0x4639, 0xb8, 0xe0, 0x8e, 0xd7, 0x3c, 0xbe, 0xa0, 0xb6	**EFI_PCI_IO_PROTOCOL.Pci.Read** – With invalid *Width* type -1, the return status is **EFI_INVALID_PARAMETER**.	1. Call **Pci.Read()** with *Width* as -1. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.7.10	0xa7710b95, 0x114d, 0x4096, 0xa8, 0x3c, 0xf6, 0x5f, 0x63, 0x00, 0xbd, 0xab	**EFI_PCI_IO_PROTOCOL.Pci.Read** – With *Buffer* as **NULL** the return status is **EFI_INVALID_PARAMETER**.	1. Call **Pci.Read()** with *Buffer* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.7.11	0x147279d7, 0xf685, 0x4658, 0xb8, 0x09, 0xdf, 0xd1, 0xd7, 0x75, 0xe0, 0xb5	**EFI_PCI_IO_PROTOCOL.Pci.Read** – With *Offset* + *Count* * *Width* > 255, the return status is **EFI_UNSUPPORTED**.	1. Call **Pci.Read()** with *Offset* + *Count* * *Width* > 255. The return status must be **EFI_UNSUPPORTED**.
5.8.2.7.12	0xf070aeda, 0x2e6b, 0x4911, 0xae, 0x80, 0x1b, 0x21, 0xcc, 0xef, 0x30, 0x50	**EFI_PCI_IO_PROTOCOL.Pci.Read** – With *Offset* + *Count* * *Width* > 255 the return status is **EFI_UNSUPPORTED**.	1. Call **Pci.Read()** with *Offset* + *Count* * *Width* > 255. The return status must be **EFI_UNSUPPORTED**.
5.8.2.7.13	0x85111b07, 0x5d78, 0x4e62, 0x90, 0x48, 0x69, 0xca, 0x37, 0x4a, 0xdc, 0xb3	**EFI_PCI_IO_PROTOCOL.Pci.Read** – With invalid *Width* the return status is **EFI_INVALID_PARAMETER**.	1. Call **Pci.Read()** with invalid *Width*. The return status must be **EFI_INVALID_PARAMETER**.

10.2.8 Pci.Write()

Number	GUID	Assertion	Test Description
5.8.2.8.1	0x1c65f03c, 0x6d87, 0x435e, 0x94, 0x2e, 0x41, 0x4f, 0xfa, 0x1d, 0x69, 0xb8	**EFI_PCI_IO_PROTOCOL.Pci.Write** – **Pci.Write()** writes data to the PCI configuration space and returns **EFI_SUCCESS**.	1. Allocate a buffer that matches the size of the address range. 2. Call **Pci.Read()** to fill in the buffer with the predefined data units. 3. Call **Pci.Write()** to write the buffer into the address range The return status should be **EFI_SUCCESS**.
5.8.2.8.2	0xb175434f, 0xf038, 0x43a2, 0xa1, 0xa8, 0xef, 0xab, 0x71, 0x57, 0x7f, 0xac	**EFI_PCI_IO_PROTOCOL.Pci.Write** – Data read out from PCI configuration space with **PciIoWidthUintX** equals the data written in.	1. Allocate a buffer that matches the size of the address range 2. Call **Pci.Read()** to fill in the buffer with the predefined data units 3. Call **Pci.Write()** to write the buffer into the address range 4. Call **Pci.Read()** to read out the data in destination address range. 5. Compare the data read out with data written in.
5.8.2.8.3	0xfbc65a77, 0xd113, 0x4584, 0xa6, 0xe0, 0x40, 0x6d, 0xc7, 0xd9, 0x24, 0x1f	**EFI_PCI_IO_PROTOCOL.Pci.Write** – **Pci.Write()** writes the data with **EfiPciIoWidthFifoX** and returns **EFI_SUCCESS**	1. Allocate a buffer that matches the size of the address range. 2. Call **Pci.Read()** to fill in the buffer with the predefined data units. 3. Call **Pci.Write()** to write to the starting address of address range using **EfiPciIoWidthFifoUintX**. The return status should be **EFI_SUCCESS**.
5.8.2.8.4	0x1dd97ca1, 0x6920, 0x41db, 0xa2, 0x0c, 0xcf, 0x62, 0x78, 0xbd, 0x07, 0x47	**EFI_PCI_IO_PROTOCOL.Pci.Write** – With **PciIoWidthFifoUintX**, the first data unit is equal to the last data unit, and the other data units are unchanged.	1. Allocate a buffer that matches the size of the address range. 2. Call **Pci.Read()** to fill in the buffer with the predefined data units. 3. Call **Pci.Write()** to write to the starting address of address range using **EfiPciIoWidthFifoUintX**. 4. Call **Pci.Read()** to read out the data using **EfiPciIoWidthFifoUintX**. 5. Compare the data in the starting address with the last data unit. Compare other data units with original data.
5.8.2.8.5	0x3ea04425, 0xbf3d, 0x465a, 0xbd, 0x5b, 0xf7, 0x77, 0xc5, 0x41, 0x21, 0x6a	**EFI_PCI_IO_PROTOCOL.Pci.Write** – **Pci.Write()** writes the data with **EfiPciIoWidthFillX** and returns **EFI_SUCCESS**.	1. Allocate a buffer that matches the size of the address range. 2. Call **Pci.Read()** to fill in the buffer with the predefined data units. 3. Call **Pci.Write()** to write to address range using **EfiPciIoWidthFillUintX**. The return status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.8.2.8.6	0x74ff6a17, 0xdf28, 0x434a, 0x8a, 0xd7, 0xbf, 0xa3, 0xe9, 0xc5, 0x1f, 0x12	EFI_PCI_IO_PROTOCOL.Pci.Write - With PciIoWidthFillX, all the data units read out are equal to the first data unit in the address space.	1. Allocate a buffer that matches the size of the address range. 2. Call Pci.Read() to fill in the buffer with the predefined data units. 3. Call Pci.Write() to write to address range using EfiPciIoWidthFillUintX. 4. Call Pci.Read() to read out the data using EfiPciIoWidthFillUintX. 5. Compare all the data units with the first data unit.
5.8.2.8.7	0xc355e57b, 0x93ef, 0x4ca6, 0x91, 0x2f, 0x65, 0x6e, 0x4f, 0x2e, 0x2a, 0x13	EFI_PCI_IO_PROTOCOL.Pci.Write - Pci.Write() writes data back with EfiPciIoWidthX and returns EFI_SUCCESS	1. Allocate a buffer that matches the size of the address range 2. Call Pci.Read() to fill in the buffer with the predefined data units 3. Call Pci.Write() to write the buffer into the address range 4. Call Pci.Read() to read out the data in destination address range. 5. Call Pci.Write() to write the data back. The return status should be EFI_SUCCESS
5.8.2.8.8	0x8a26f93b, 0xc0a3, 0x4e08, 0x9f, 0xf1, 0xd6, 0xf1, 0xac, 0x2e, 0x63, 0x9a	EFI_PCI_IO_PROTOCOL.Pci.Write - With *Width* as EfiPciIoWidthMaximum the return status is EFI_INVALID_PARAMETER	1. Call Pci.Write() with *Width* as EfiPciWidthMaximum. The return status must be EFI_INVALID_PARAMETER
5.8.2.8.9	0xfeab0187, 0x541b, 0x45da, 0x92, 0x1f, 0x49, 0x01, 0x00, 0xb7, 0xdd, 0x7a	EFI_PCI_IO_PROTOCOL.Pci.Write - With invalid *Width* type -1 the return status is EFI_INVALID_PARAMETER	1. Call Pci.Write() with *Width* as -1. The return status must be EFI_INVALID_PARAMETER
5.8.2.8.10	0x686732db, 0xa12b, 0x4ed7, 0x90, 0xfb, 0x66, 0x92, 0xbb, 0xd7, 0xe8, 0x4c	EFI_PCI_IO_PROTOCOL.Pci.Write - With *Buffer* as NULL the return status is EFI_INVALID_PARAMETER	1. Call Pci.Write() with *Buffer* as NULL. The return status must be EFI_INVALID_PARAMETER

Number	GUID	Assertion	Test Description
5.8.2.8.11	0x11cf0b51, 0x6f50, 0x4bba, 0xa9, 0xd7, 0x3e, 0x53, 0x28, 0xb3, 0x1f, 0x30	**EFI_PCI_IO_PROTOCOL.Pci.Write** – With *Offset* + *Count* * *Width* > 255 the return status is **EFI_UNSUPPORTED**	1. Call **Pci.Write()** with *Offset* + *Count* * *Width* > 255. The return status must be **EFI_UNSUPPORTED**
5.8.2.8.12	0x4e4617a2, 0x4e8a, 0x46c8, 0xb2, 0x4b, 0xa4, 0x91, 0x55, 0xf2, 0x3a, 0x0d	**EFI_PCI_IO_PROTOCOL.Pci.Write** – With *Offset* + *Count* * *Width* > 255 the return status is **EFI_UNSUPPORTED**	1. Call **Pci.Write()** with *Offset* + *Count* * *Width* > 255. The return status must be **EFI_UNSUPPORTED**
5.8.2.8.13	0xc6dbb28e, 0xbf42, 0x40e3, 0xbc, 0x93, 0x5f, 0x9b, 0x11, 0xa2, 0x46, 0x5f	**EFI_PCI_IO_PROTOCOL.Pci.Write** – With invalid *Width* the return status is **EFI_INVALID_PARAMETER**	1. Call **Pci.Write()** with invalid *Width*. The return status must be **EFI_INVALID_PARAMETER**

10.2.9 CopyMem()

Number	GUID	Assertion	Test Description
5.8.2.9.1	0x8d728b05, 0xc64e, 0x4ef0, 0x80, 0x68, 0x51, 0xbc, 0xe3, 0x9f, 0xc5, 0x0c	**EFI_PCI_IO_PROTOCOL.CopyMem** – **CopyMem()** copying Data between non-overlapping regions returns **EFI_SUCCESS**.	1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / *Width* / 2 * *Width*. 2. Call **Mem.Write()** to write the buffer into the beginning address. 3. Call **CopyMem()** to copy Data between non-overlapping regions. The return status should be **EFI_SUCCESS**.
5.8.2.9.2	0x73f80e2c, 0xe2d9, 0x4c6b, 0xbe, 0xc0, 0x85, 0xd7, 0xa4, 0x27, 0x07, 0xd0	**EFI_PCI_IO_PROTOCOL.CopyMem** – Data copied between non-overlapping regions is equal.	1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / *Width* / 2 * *Width*. 2. Call **Mem.Write()** to write the buffer into the beginning address. 3. Call **CopyMem()** to copy Data between non-overlapping regions. 4. Call **Mem.Read()** to read out the data. 5. Compare the data read out with the data written in.

Number	GUID	Assertion	Test Description
5.8.2.9.3	0x459bcee9, 0x16f7, 0x41ae, 0x81, 0x55, 0x7e, 0x49, 0xec, 0x98, 0x56, 0x7d	**EFI_PCI_IO_PROTOCOL.CopyMem – CopyMem()** copying Data between overlapping regions (destination address > source address) returns **EFI_SUCCESS**.	1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / *Width* / 2 * *Width*. 2. Call **Mem.Write()** to write the buffer into the beginning address. 3. Call **CopyMem()** to copy Data between overlapping regions (destination address > source address). The return status should be **EFI_SUCCESS**.
5.8.2.9.4	0x9ca6f1d4, 0xfb7c, 0x416c, 0xa6, 0x09, 0x06, 0xa4, 0xcb, 0x0f, 0x44, 0x59	**EFI_PCI_IO_PROTOCOL.CopyMem –** When copying Data between overlapping regions (destination > source), the data is copied.	1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / *Width* / 2 * *Width*. 2. Call **Mem.Write()** to write the buffer into the beginning address. 3. Call **CopyMem()** to copy Data between overlapping regions (destination address > source address). 4. Call **Mem.Read()** to read out the data 5. Compare the data read out with the data written in.
5.8.2.9.5	0xb8eb3987, 0x9915, 0x40d2, 0x93, 0xc6, 0xe1, 0x83, 0x7e, 0x49, 0x4e, 0x1a	**EFI_PCI_IO_PROTOCOL.CopyMem – CopyMem()** copying Data between overlapping regions (destination address < source address) returns **EFI_SUCCESS**.	1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / *Width* / 2 * *Width*. 2. Call **Mem.Write()** to write the buffer into the beginning address. 3. Call **CopyMem()** to copy Data between overlapping regions (destination address < source address). The return status should be **EFI_SUCCESS**.
5.8.2.9.6	0x3294319c, 0xc3f0, 0x46f2, 0x81, 0xfd, 0x14, 0xc0, 0xd0, 0x61, 0xc4, 0x42	**EFI_PCI_IO_PROTOCOL.CopyMem –** When copying Data between overlapping regions (destination < source) the data is copied.	1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / *Width* / 2 * *Width*. 2. Call **Mem.Write()** to write the buffer into the beginning address. 3. Call **CopyMem()** to copy Data between overlapping regions (destination address < source address). 4. Call **Mem.Read()** to read out the data. 5. Compare the data read out with the data written in.

Number	GUID	Assertion	Test Description
5.8.2.9.7	0xd0b52eb3, 0x3d19, 0x4b72, 0xb5, 0xba, 0xe3, 0xa3, 0x7c, 0xd0, 0xcb, 0x93	**EFI_PCI_IO_PROTOCOL.CopyMem – CopyMem()** copying Data between different BARs returns **EFI_SUCCESS**.	1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / *Width* / 2 * *Width*. 2. Call **Mem.Write()** to write the buffer into the beginning address. 3. Call **CopyMem()** to copy Data between different BARs. The return status should be **EFI_SUCCESS**.
5.8.2.9.8	0xe0863095, 0x4854, 0x4099, 0x89, 0xf0, 0x01, 0xbf, 0xda, 0x41, 0xa4, 0xe3	**EFI_PCI_IO_PROTOCOL.CopyMem –** When copying Data between different BARs the data is copied.	1. Allocate a buffer, the size of which is: **BufferSize** = Address Range Size / *Width* / 2 * *Width*. 2. Call **Mem.Write()** to write the buffer into the beginning address. 3. Call **CopyMem()** to copy Data between different BARs. 4. Call **Mem.Read()** to read out the data. 5. Compare the data read out with the data written in.
5.8.2.9.9	0x45056bf8, 0xe6e4, 0x4397, 0xb7, 0xe1, 0x89, 0x0b, 0x42, 0x4d, 0xe3, 0x54	**EFI_PCI_IO_PROTOCOL.CopyMem –** With *Width* as **EfiPciIoWidthMaximum** the return status is **EFI_INVALID_PARAMETER**.	1. Call **CopyMem()** with *Width* as **EfiPciWidthMaximum**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.9.10	0xf780b74f, 0x6b93, 0x4e64, 0x8a, 0xb5, 0x05, 0x77, 0xdd, 0x99, 0xc1, 0xfa	**EFI_PCI_IO_PROTOCOL.CopyMem –** With invalid *Width* type -1 the return status is **EFI_INVALID_PARAMETER**.	1. Call **CopyMem()** with *Width* as -1. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.9.11	0xebf7fa5c, 0xb4c9, 0x406c, 0x8d, 0x12, 0x90, 0x3a, 0x81, 0x85, 0x26, 0x17	**EFI_PCI_IO_PROTOCOL.CopyMem –** With *Width* as **EfiPciWidthFifoUintX** the return status is **EFI_INVALID_PARAMETER**.	1. Call **CopyMem()** with *Width* as **EfiPciWidthFifoUintX**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.9.12	0xc07ea144, 0x18b5, 0x40e5, 0xa0, 0xa0, 0xd4, 0xca, 0x6c, 0x82, 0xc2, 0x9d	**EFI_PCI_IO_PROTOCOL.CopyMem –** With *Width* as **EfiPciWidthFillUintX**, the return status is **EFI_INVALID_PARAMETER**.	1. Call **CopyMem()** with *Width* as **EfiPciWidthFillUintX**. The return status must be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.8.2.9.13	0x299293a3, 0xe8db, 0x43a4, 0x9b, 0x3f, 0x5e, 0x23, 0xb2, 0x9e, 0x37, 0x31	`EFI_PCI_IO_PROTOCOL.CopyMem` – With Source Address area out of BAR range, the return status is `EFI_UNSUPPORTED`.	1. Call `CopyMem()` with `Source` Address area out of BAR range. The return status must be `EFI_UNSUPPORTED`.
5.8.2.9.14	0x33c447ae, 0x5caf, 0x4904, 0xaf, 0x90, 0x66, 0x78, 0x17, 0x45, 0x0e, 0x12	`EFI_PCI_IO_PROTOCOL.CopyMem` – With Destination Address area out of BAR range, the return status is `EFI_UNSUPPORTED`.	1. Call `CopyMem()` with Destination Address area out of BAR range. The return status must be `EFI_UNSUPPORTED`.
5.8.2.9.15	0xe2dd0321, 0xac26, 0x4aac, 0xa6, 0x28, 0xc2, 0x59, 0xbc, 0x8a, 0xd5, 0x2c	`EFI_PCI_IO_PROTOCOL.CopyMem` – With invalid Source BAR Index, return status is `EFI_UNSUPPORTED`.	1. Call `CopyMem()` with invalid Source BAR Index. The return status must be `EFI_UNSUPPORTED`.
5.8.2.9.16	0x110a96a1, 0x7a2e, 0x4eab, 0xbc, 0x11, 0xf3, 0xda, 0x30, 0xd0, 0xa2, 0xff	`EFI_PCI_IO_PROTOCOL.CopyMem` – With invalid Destination, BAR Index return status is `EFI_UNSUPPORTED`.	1. Call `CopyMem()` with invalid Destination BAR Index. The return status must be `EFI_UNSUPPORTED`.
5.8.2.9.17	0x7d1c3de1, 0xa7b8, 0x4923, 0x94, 0x13, 0x76, 0x49, 0x05, 0x01, 0xf6, 0x9f	`EFI_PCI_IO_PROTOCOL.CopyMem` – With Source BAR Index as an IO type BAR, the return status is `EFI_UNSUPPORTED`.	1. Call `CopyMem()` with Source BAR Index as an IO type BAR. The return status must be `EFI_UNSUPPORTED`.
5.8.2.9.18	0xaacfb1ec, 0xd6fb, 0x4c3a, 0xa4, 0x8c, 0x4a, 0xc2, 0x77, 0xfb, 0xc8, 0xe3	`EFI_PCI_IO_PROTOCOL.CopyMem` – With Destination BAR Index as an IO type BAR, the return status is `EFI_UNSUPPORTED`.	1. Call `CopyMem()` with Destination BAR Index as an IO type BAR. The return status must be `EFI_UNSUPPORTED`.

Number	GUID	Assertion	Test Description
5.8.2.9.19	0x83b30e84, 0x528f, 0x420d, 0x87, 0x48, 0x7d, 0x96, 0x36, 0x8e, 0x33, 0x58	**EFI_PCI_IO_PROTOCOL.CopyMem** – With invalid Width, the return status is **EFI_INVALID_PARAMETER**.	1. Call **CopyMem()** with invalid Width. The return status must be **EFI_INVALID_PARAMETER**.

10.2.10 Map()

Number	GUID	Assertion	Test Description
5.8.2.10.1	0x720e6fdc, 0x91c8, 0x4fd5, 0xb5, 0xde, 0xb1, 0xcc, 0x3b, 0x0c, 0x0c, 0xba	**EFI_PCI_IO_PROTOCOL.Map – Map()** with Bus Master Read returns **EFI_SUCCESS**.	1. Allocate a buffer (4K + 1 Bytes). 2. Fill in the buffer with some data. 3. Call **Map()** with Operation as **EfiPciIoOperationBusMasterRead** to map this range to a new DMA capable location. The return status should be **EFI_SUCCESS**.
5.8.2.10.2	0xbf7f859c, 0x20e5, 0x4418, 0x8e, 0x21, 0x87, 0x60, 0x60, 0x58, 0x73, 0xa2	**EFI_PCI_IO_PROTOCOL.Map – Map()** with Bus Master Read, mapped bytes are > 0.	1. Allocate a buffer (4K + 1 Bytes). 2. Fill in the buffer with some data. 3. Call **Map()** with Operation as **EfiPciIoOperationBusMasterRead** to map this range to a new DMA capable location. 4. Check if the number of bytes mapped great than 0.
5.8.2.10.3	0xd56b3a96, 0x7c58, 0x4209, 0x85, 0xe9, 0x90, 0xb2, 0x07, 0x90, 0x6d, 0x55	**EFI_PCI_IO_PROTOCOL.Map – Map()** with Bus Master Read, the mapped area equals original area.	1. Allocate a buffer (4K + 1 Bytes). 2. Fill in the buffer with some data. 3. Call **Map()** with Operation as **EfiPciIoOperationBusMasterRead** to map this range to a new DMA capable location. 4. Check if data of mapped area ishe same as the data of original area.
5.8.2.10.4	0x5539608f, 0xed60, 0x4172, 0x94, 0x4e, 0xe9, 0x4a, 0x0f, 0x61, 0xf7, 0xe8	**EFI_PCI_IO_PROTOCOL.Map – Map()** with Bus Master Write returns **EFI_SUCCESS**.	1. Allocate a buffer (4K + 1 Bytes) . 2. Fill in the buffer with some data. 3. Call **Map()** with Operation as **EfiPciIoOperationBusMasterWrite** to map this range to a new DMA capable location The return status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.8.2.10.5	0xb4019165, 0x7b45, 0x4ec4, 0xa7, 0xeb, 0xc5, 0x67, 0x71, 0x07, 0xd9, 0x4c	`EFI_PCI_IO_PROTOCOL.Map - Map()` with Bus Master Write, mapped bytes are > 0.	1. Allocate a buffer (4K + 1 Bytes). 2. Fill in the buffer with some data. 3. Call `Map()` with Operation as `EfiPciIoOperationBusMasterWrite` to map this range to a new DMA capable location. 4. Check if the number of bytes mapped great than 0.
5.8.2.10.6	0x6b4e9c1e, 0xa1e7, 0x4cf5, 0x8d, 0x0f, 0xdd, 0x68, 0x80, 0xcd, 0x8f, 0x43	`EFI_PCI_IO_PROTOCOL.Map - Map()` with Bus Master Write, original data remains unchanged immediatelyafter mapping.	1. Allocate a buffer (4K + 1 Bytes) . 2. Fill in the buffer with some data. 3. Call `Map()` with Operation as `EfiPciIoOperationBusMasterWrite` to map this range to a new DMA capable location. 4. Check if the data of the original area is unchanged.
5.8.2.10.7	0x9a37eb62, 0x4bab, 0x4fce, 0x81, 0x9d, 0x0d, 0x80, 0x42, 0xea, 0x46, 0x7e	`EFI_PCI_IO_PROTOCOL.Map - Map()` with Bus Master Common, the Buffer returns `EFI_SUCCESS`.	1. Allocate a buffer (4K + 1 Bytes) 2. Fill in the buffer with some data 3. Call `Map()` with Operation as `EfiPciIoOperationBusMasterCommon` to map this range to a new DMA capable location The return status should be `EFI_SUCCESS`.
5.8.2.10.8	0x4d562d9c, 0xb028, 0x43ff, 0xb7, 0xfc, 0x92, 0xdb, 0x62, 0x40, 0xd5, 0x9a	`EFI_PCI_IO_PROTOCOL.Map - Map()` with Bus Master Common, the Buffer mapped bytes are > 0.	1. Allocate a buffer (4K + 1 Bytes) . 2. Fill in the buffer with some data. 3. Call `Map()` with Operation as `EfiPciIoOperationBusMasterCommon` to map this range to a new DMA capable location. 4. Check if the number of bytes mapped are greater than 0.
5.8.2.10.9	0x8bd3ecc4, 0x43ea, 0x4f9e, 0x84, 0x79, 0x8c, 0x36, 0xde, 0x51, 0x13, 0x2f	`EFI_PCI_IO_PROTOCOL.Map - Map()` with Bus Master Common, the Buffer mapped area equalsthe original area after mapping.	1. Allocate a buffer (4K + 1 Bytes). 2. Fill in the buffer with some data. 3. Call `Map()` with Operation as `EfiPciIoOperationBusMasterCommon` to map this range to a new DMA capable location. 4. Check if the data of mapped areais is the same as the data of the original area.

Number	GUID	Assertion	Test Description
5.8.2.10.10	0x673d01f2, 0xdabf, 0x49bb, 0xbe, 0xc5, 0xe7, 0xa0, 0x3a, 0xd7, 0x71, 0xbc	**`EFI_PCI_IO_PROTOCOL.Map – Map()`** with Bus Master Common, the Buffer data of the original area is sync'd with the mapped area.	1. Allocate a buffer (4K + 1 Bytes). 2. Fill in the buffer with some data. 3. Call **`Map()`** with Operation as **`EfiPciIoOperationBusMasterCommon`** to map this range to a new DMA capable location. 4. Call **`SetMem()`** to fill in mapped address with some fixed data. 5. Check if the data of the original area is synchronized with the mapped area.
5.8.2.10.11	0xbd5fcf21, 0xdb42, 0x4f4f, 0xb0, 0xfb, 0x56, 0x62, 0xd5, 0x1a, 0xba, 0x68	**`EFI_PCI_IO_PROTOCOL.Map – Map()`** with Bus Master Common, the Buffer data of the mapped area syncs with original area.	1. Allocate a buffer (4K + 1 Bytes) . 2. Fill in the buffer with some data. 3. Call **`Map()`** with Operation as **`EfiPciIoOperationBusMasterCommon`** to map this range to a new DMA capable location. 4. Call **`SetMem()`** to fill in original address with some fixed data. 5. Check if the data of mapped area is synchronized with the original area.
5.8.2.10.12	0xe2fa9ae5, 0xea93, 0x48b2, 0xba, 0x85, 0xa3, 0x74, 0xe2, 0xdb, 0xe2, 0xaf	**`EFI_PCI_IO_PROTOCOL.Map –`** Mapping with Operation as **`EfiPciIoOperationMaximum`** returns a status of **`EFI_INVALID_PARAMETER`**.	1. Call **`Map()`** with Operation as **`EfiPciIoOperationMaximum`**. The return status must be **`EFI_INVALID_PARAMETER`**.
5.8.2.10.13	0x3b337461, 0x98da, 0x4117, 0xab, 0xef, 0x57, 0x60, 0x34, 0xfd, 0xc6, 0x22	**`EFI_PCI_IO_PROTOCOL.Map –`** Mapping with *`Operation`* as **`EfiPciIoOperationMaximum`** + 1 returns a status of **`EFI_INVALID_PARAMETER`**.	1. Call **`Map()`** with *`Operation`* as **`EfiPciIoOperationMaximum`** + 1. The return status must be **`EFI_INVALID_PARAMETER`**.
5.8.2.10.14	0xdce36bfb, 0xde48, 0x4f84, 0x9d, 0xc1, 0xde, 0x92, 0xa4, 0x40, 0x50, 0xbb	**`EFI_PCI_IO_PROTOCOL.Map –`** Mapping with *`Operation`* as -1 returns a status of **`EFI_INVALID_PARAMETER`**.	1. Call **`Map()`** with *`Operation`* as -1. The return status must be **`EFI_INVALID_PARAMETER`**.

Number	GUID	Assertion	Test Description
5.8.2.10.15	0x8aa3c1cb, 0x5c8d, 0x4a74, 0x83, 0x81, 0x4b, 0x15, 0x3a, 0xf8, 0xff, 0x17	**EFI_PCI_IO_PROTOCOL.Map** – Mapping with *HostAddress* as **NULL** returns a status of **EFI_INVALID_PARAMETER**	1. Call **Map()** with *HostAddress* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.10.16	0x495cff3e, 0x5f7a, 0x4888, 0x85, 0x9f, 0xb7, 0x26, 0x0b, 0xb4, 0x18, 0xaf	**EFI_PCI_IO_PROTOCOL.Map** – Mapping with *NumberOfBytes* as **NULL** returns a status of **EFI_INVALID_PARAMETER**	1. Call **Map()** with *NumberOfBytes* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.10.17	0x7e34b406, 0x0821, 0x4b95, 0xa4, 0x18, 0xc2, 0x0e, 0xfc, 0xfc, 0x00, 0xef	**EFI_PCI_IO_PROTOCOL.Map** – Mapping with *DeviceAddress* as **NULL** returns a status of **EFI_INVALID_PARAMETER**	1. Call **Map()** with *DeviceAddress* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.10.18	0x6b450eae, 0x225c, 0x4ff1, 0x93, 0xd1, 0x55, 0xf9, 0xae, 0x35, 0x3e, 0xf8	**EFI_PCI_IO_PROTOCOL.Map** – Mapping with *Mapping* as **NULL** returns a status of **EFI_INVALID_PARAMETER**.	1. Call **Map()** with *Mapping* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.
5.8.2.10.19	0x07a924a7, 0xe637, 0x4f46, 0x9b, 0x3c, 0x04, 0x63, 0x86, 0xfb, 0xf6, 0xf0	**EFI_PCI_IO_PROTOCOL.Map** – Mapping with *HostAddress* + *NumberOfByte* > 4G returns a status of **EFI_UNSUPPORTED**.	1. Call **Map()** with *HostAddress* + *NumberOfByte* > 4G. The return status must be **EFI_UNSUPPORTED**.

10.2.11 Unmap()

Number	GUID	Assertion	Test Description
5.8.2.11.1	0xd9f80cd4, 0x8f0b, 0x4a27, 0x99, 0x16, 0x1a, 0x47, 0xfd, 0x8f, 0x07, 0x25	`EFI_PCI_IO_PROTOCOL.Unmap - Unmap()` area mapped wih `BusMasterRead` returns `EFI_SUCCESS`	1. Allocate a buffer (4K + 1 Bytes) 2. Fill in the buffer with some data 3. Call `Map()` with Operation as `EfiPciIoOperationBusMasterRead` to map this range to a new DMA capable location 4. Call `Unmap()` to release the mapped resources The return status should be `EFI_SUCCESS`
5.8.2.11.2	0x8f86dbbf, 0xcc86, 0x40d0, 0x89, 0xb3, 0x97, 0xd6, 0xf1, 0xe8, 0xd7, 0x80	`EFI_PCI_IO_PROTOCOL.Unmap - Unmap()`leaves data in the original area mapped wih `BusMasterRead` unchangedafter Unmap	1. Allocate a buffer (4K + 1 Bytes) 2. Fill in the buffer with some data 3. Call `Map()` with Operation as `EfiPciIoOperationBusMasterRead` to map this range to a new DMA capable location 4. Call `Unmap()` to release the mapped resources 5. Check if the data in original area remains unchanged
5.8.2.11.3	0xab8555aa, 0x8c45, 0x4bec, 0x90, 0x9a, 0xad, 0xc7, 0xfe, 0xe9, 0xaf, 0xf4	`EFI_PCI_IO_PROTOCOL.Unmap - Unmap()` area mapped wih `BusMasterWrite` returns `EFI_SUCCESS`	1. Allocate a buffer (4K + 1 Bytes) 2. Fill in the buffer with some data 3. Call `Map()` with Operation as `EfiPciIoOperationBusMasterWrite` to map this range to a new DMA capable location 4. Call `Unmap()` to release the mapped resources The return status should be `EFI_SUCCESS`
5.8.2.11.4	0xa6537c2a, 0x34bc, 0x4604, 0x81, 0x48, 0xb1, 0x41, 0x70, 0x46, 0x86, 0xe4	`EFI_PCI_IO_PROTOCOL.Unmap - Unmap()` leaves data in the original area mapped wih `BusMasterWrite` equal with the data written in mapped area	1. Allocate a buffer (4K + 1 Bytes) 2. Fill in the buffer with some data 3. Call `Map()` with Operation as `EfiPciIoOperationBusMasterWrite` to map this range to a new DMA capable location 4. Call `Unmap()` to release the mapped resources 5. Check if the data in the original area is equal with the data in mapped area

Number	GUID	Assertion	Test Description
5.8.2.11.5	0x79009fa0, 0x5b72, 0x4e82, 0x84, 0x84, 0x3a, 0x21, 0xe0, 0x57, 0x93, 0xb9	`EFI_PCI_IO_PROTOCOL.Unmap - Unmap()` area mapped wih Bus Master Common Read returns `EFI_SUCCESS`	1. Allocate a buffer (4K + 1 Bytes) 2. Fill in the buffer with some data 3. Call `Map()` with Operation as `EfiPciIoOperationBusMasterCommon` to map this range to a new DMA capable location 4. Call `Unmap()` to release the mapped resources The return status should be `EFI_SUCCESS`
5.8.2.11.6	0xda153716, 0xcd62, 0x4612, 0xae, 0x11, 0x71, 0x5e, 0x97, 0xeb, 0x6a, 0x9a	`EFI_PCI_IO_PROTOCOL.Unmap - Unmap()` leaves data in the original area mapped wih Bus Master Common Read unchanged after Unmap	1. Allocate a buffer (4K + 1 Bytes) 2. Fill in the buffer with some data 3. Call `Map()` with Operation as `EfiPciIoOperationBusMasterCommon` to map this range to a new DMA capable location 4. Call `Unmap()` to release the mapped resources 5. Check if the data in the original area remains unchanged

10.2.12 AllocateBuffer()

Number	GUID	Assertion	Test Description

5.8.2.12.1	0x841e89ab, 0x9c60, 0x48e5, 0xae, 0x7d, 0x51, 0x21, 0xf5, 0x08, 0xe1, 0x0c	`EFI_PCI_IO_PROTOCOL.AllocateBuffer - AllocateBuffer()` with correct Parameter status returns `EFI_SUCCESS`	1. Call `AllocateBuffer()` with the following parameters having multiple enumerated values:
			`MemoryType` – `EfiBootServicesData` and `EfiRuntimeServicesData`
			`Attributes` – 0, `EFI_PCI_ATTRIBUTE_MEMORY_WRITE_COMBINE` `EFI_PCI_ATTRIBUTE_MEMORY_CACHED`, and `EFI_PCI_ATTRIBUTE_MEMORY_WRITE_COMBINE` | `EFI_PCI_ATTRIBUTE_MEMORY_CACHED`
			The return status should be `EFI_SUCCESS`

Number	GUID	Assertion	Test Description
5.8.2.12.2	0x576894ad, 0x9229, 0x4078, 0xa9, 0x69, 0x70, 0x0e, 0x6e, 0x04, 0x4b, 0xb3	**EFI_PCI_IO_PROTOCOL.AllocateBuffer** – With invalid memory type the status is **EFI_INVALID_PARAMETER**	1. Call **AllocateBuffer()** with invalid memory type. The return status must be **EFI_INVALID_PARAMETER**
5.8.2.12.3	0xa0c5c95e, 0xf251, 0x4c00, 0x9f, 0xdf, 0x9c, 0x88, 0xa2, 0xaa, 0x45, 0x6b	**EFI_PCI_IO_PROTOCOL.AllocateBuffer** – With *HostAddress* as **NULL** the status is **EFI_INVALID_PARAMETER**	1. Call **AllocateBuffer()** with *HostAddress* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**
5.8.2.12.4	0xfacb1e1b, 0x0327, 0x4341, 0xa9, 0x42, 0x4d, 0xb9, 0x9f, 0x1d, 0xe5, 0x68	**EFI_PCI_IO_PROTOCOL.AllocateBuffer** – With invalid *Attributes* the status is **EFI_UNSUPPORTED**	1. Call **AllocateBuffer()** with invalid *Attributes*. The return status must be **EFI_UNSUPPORTED**

10.2.13 FreeBuffer()

Number	GUID	Assertion	Test Description
5.8.2.13.1	0x00312f50, 0x721c, 0x4085, 0x82, 0x63, 0x04, 0xd1, 0x1f, 0x37, 0x2c, 0x6c	**EFI_PCI_IO_PROTOCOL.FreeBuffer** – **FreeBuffer()** return status is **EFI_SUCCESS**	1. Call **AllocateBuffer()** with the following parameters having multiple enumerated values: **MemoryType** – **EfiBootServicesData** and **EfiRuntimeServicesData** **Attributes** – 0, **EFI_PCI_ATTRIBUTE_MEMORY_WRITE_COMBINE** **EFI_PCI_ATTRIBUTE_MEMORY_CACHED**, and **EFI_PCI_ATTRIBUTE_MEMORY_WRITE_COMBINE** | **EFI_PCI_ATTRIBUTE_MEMORY_CACHED** 2. Call **FreeBuffer()** The return status should be **EFI_SUCCESS**

10.2.14 Flush()

Number	GUID	Assertion	Test Description
5.8.2.14.1	0x2c9f36a3, 0x4cab, 0x4434, 0xa8, 0xc1, 0x7b, 0xf6, 0x3c, 0x46, 0x8f, 0x05	**EFI_PCI_IO_PROTOCOL.Flush - Flush()** return status is **EFI_SUCCESS**	1. Call **Flush()** The return status should be **EFI_SUCCESS**

10.2.15 GetLocation()

Number	GUID	Assertion	Test Description
5.8.2.15.1	0xfb478a8e, 0x58e2, 0x41b9, 0x89, 0x35, 0x71, 0x7b, 0x5a, 0x90, 0xa1, 0x84	**EFI_PCI_IO_PROTOCOL.GetLocation - GetLocation()** return status is **EFI_SUCCESS**	1. Call **GetLocation()** The return status should be **EFI_SUCCESS**
5.8.2.15.2	0x07b74ac9, 0x96f4, 0x4d00, 0x94, 0xbd, 0x09, 0x60, 0xd4, 0xe9, 0xa6, 0xe7	**EFI_PCI_IO_PROTOCOL.GetLocation - GetLocation()** returns a *BusNumber* < 256	1. Call **GetLocation()** 2. Check if the returned *BusNumber* is less than 256
5.8.2.15.3	0xaf7155de, 0x45f4, 0x4b97, 0xb4, 0xac, 0x07, 0x1a, 0x53, 0x43, 0x32, 0x48	**EFI_PCI_IO_PROTOCOL.GetLocation - GetLocation()** returns a *DeviceNumber* < 32	1. Call **GetLocation()** 2. Check if the returned *DeviceNumber* is less than 32
5.8.2.15.4	0x838f7bf6, 0xfa36, 0x4149, 0x92, 0x29, 0xce, 0x60, 0x8a, 0x66, 0x35, 0x61	**EFI_PCI_IO_PROTOCOL.GetLocation - GetLocation()** returns a *FunctionNumber* < 8	1. Call **GetLocation()** 2. Check if the returned *FunctionNumber* is less than 8

Number	GUID	Assertion	Test Description
5.8.2.15.5	0xa5510fe8, 0x2178, 0x47e6, 0x9e, 0xcc, 0xe9, 0x0b, 0x92, 0xcf, 0x1b, 0xbb	**EFI_PCI_IO_PROTOCOL.GetLocation -** With *SegmentNumber* as **NULL**, the status is **EFI_INVALID_PARAMETER**	1. Call **GetLocation()** with *SegmentNumber* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**
5.8.2.15.6	0x2a1ff8b2, 0xc540, 0x4f12, 0x9c, 0x06, 0x36, 0x8d, 0x45, 0x7c, 0x02, 0x7c	**EFI_PCI_IO_PROTOCOL.GetLocation -** With *BusNumber* as **NULL**, the status is **EFI_INVALID_PARAMETER**	1. Call **GetLocation()** with *BusNumber* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**
5.8.2.15.7	0x5e74e7e0, 0x36b0, 0x4c5d, 0x88, 0xb8, 0xb7, 0x52, 0xad, 0x2c, 0xbf, 0x61	**EFI_PCI_IO_PROTOCOL.GetLocation -** With *DeviceNumber* as **NULL**, the status is **EFI_INVALID_PARAMETER**	1. Call **GetLocation()** with *DeviceNumber* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**
5.8.2.15.8	0xb37cb86c, 0xdd05, 0x4082, 0xa6, 0xf1, 0x8c, 0xf9, 0xc3, 0x46, 0x77, 0x7a	**EFI_PCI_IO_PROTOCOL.GetLocation -** With *FunctionNumber* as **NULL**, the status is **EFI_INVALID_PARAMETER**	1. Call **GetLocation()** with *FunctionNumber* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**

10.2.16 Attributes()

Number	GUID	Assertion	Test Description
5.8.2.16.1	0x33ca89a5, 0xefa8, 0x4f52, 0x84, 0xf6, 0x2e, 0x95, 0x26, 0x23, 0xb0, 0xe1	**EFI_PCI_IO_PROTOCOL.Attributes -** Call **Attributes()** get current attribute status must be **EFI_SUCCESS**	1. Call **Attributes()** with **EfiPciIoAttributeOperationGet** to get the current attributes of the PCI controller The return status should be **EFI_SUCCESS**

Number	GUID	Assertion	Test Description
5.8.2.16.2	0xa11652df, 0x8818, 0x4a05, 0xbe, 0xd9, 0x27, 0xf9, 0xe5, 0xad, 0x78, 0x3c	EFI_PCI_IO_PROTOCOL.Attributes – Call Attributes() get supported attribute status must be EFI_SUCCESS	1. Call Attributes() with EfiPciIoAttributeOperationSupported to get the supported attributes of the PCI controller The return status should be EFI_SUCCESS
5.8.2.16.3	0x69ce5213, 0x7180, 0x4beb, 0x9f, 0x39, 0x1d, 0x1f, 0x17, 0x00, 0x59, 0x9a	EFI_PCI_IO_PROTOCOL.Attributes - Current attributes should in supported attributes	1. Call Attributes() with EfiPciIoAttributeOperationGet to get the current attributes of the PCI controller 2. Call Attributes() with EfiPciIoAttributeOperationSupported to get the supported attributes of the PCI controller 3. Check if the current attributes is a subset of Supported attributes
5.8.2.16.4	0xfac8ddb3, 0xbfae, 0x40ff, 0xb7, 0x31, 0x26, 0x8e, 0x58, 0x29, 0x25, 0xb0	EFI_PCI_IO_PROTOCOL.Attributes – Call Attributes() set *Attributes* as Supported attributes return status must be EFI_SUCCESS	1. Call Attributes() with EfiPciIoAttributeOperationSet with a supported attribute of the PCI controller. The return status should be EFI_SUCCESS
5.8.2.16.5	0xf8e48da6, 0x72e2, 0x4905, 0xa7, 0x19, 0xe3, 0xa5, 0x77, 0xca, 0xa2, 0xa8	EFI_PCI_IO_PROTOCOL.Attributes - Set *Attributes* as supported attributes the attributes should really be cleared	1. Call Attributes() with EfiPciIoAttributeOperationSet with a supported attribute of the PCI controller. 2. Call Attributes() with EfiPciIoAttributeOperationGet to get the attributes of the PCI controller 3. Check if the gotten attributes is the same as the set ones.
5.8.2.16.6	0x02cab1a9, 0x4be9, 0x4c47, 0xb2, 0x75, 0xca, 0xed, 0x59, 0x62, 0x1f, 0x41	EFI_PCI_IO_PROTOCOL.Attributes – Call Attributes() set *Attributes* as 0 return status must be EFI_SUCCESS	1. Call Attributes() with EfiPciIoAttributeOperationSet with an attribute value of 0 to clear all attributes. The return status should be EFI_SUCCESS

Number	GUID	Assertion	Test Description
5.8.2.16.7	0x88791167, 0xb9f3, 0x42ae, 0x84, 0xd1, 0xa1, 0xb6, 0xd3, 0xeb, 0xb8, 0x2f	**EFI_PCI_IO_PROTOCOL.Attributes** - Set *Attributes* as 0 the attributes should really be cleared	1. Call **Attributes()** with **EfiPciIoAttributeOperationSet** with an *Attributes* value of 0 to clear all attributes. 2. Call **Attributes()** with **EfiPciIoAttributeOperationGet** to get the attributes of the PCI controller 3. Check if the gotten attributes is the same as that of set
5.8.2.16.8	0x04479c23, 0xc700, 0x439f, 0xb7, 0x42, 0x91, 0x9a, 0x6b, 0x2e, 0x71, 0x5a	**EFI_PCI_IO_PROTOCOL.Attributes** - Call **Attributes()** enable *Attributes* as original attributes return status must be **EFI_SUCCESS**	1. Call **Attributes()** with ***EfiPciIoAttributeOperation*Enable** with supported attributes The return status should be **EFI_SUCCESS**
5.8.2.16.9	0x1d011f3e, 0xaa23, 0x4b0b, 0xb1, 0x65, 0x8f, 0x6f, 0x21, 0xf3, 0x85, 0x6d	**EFI_PCI_IO_PROTOCOL.Attributes** - enable *Attributes* as original attributes the attributes should really be **Enabled**	1. Call **Attributes()** with **EfiPciIoAttributeOperationEnable** with supported attributes 2. Call **Attributes()** with **EfiPciIoAttributeOperationGet** to get the attributes 3. Check if the attributes value is the same as original attributes
5.8.2.16.10	0x35e690e9, 0xd037, 0x41a1, 0x93, 0x44, 0x86, 0x78, 0x02, 0xe2, 0x37, 0xfc	**EFI_PCI_IO_PROTOCOL.Attributes** - Call **Attributes()** disable original attributes return status must be **EFI_SUCCESS**	1. Call **Attributes()** with **EfiPciIoAttributeOperationDisable** with supported attributes The return status should be **EFI_SUCCESS**
5.8.2.16.11	0xb7376265, 0xfb7f, 0x410c, 0x99, 0xb5, 0x5b, 0x17, 0x37, 0x41, 0xf7, 0x03	**EFI_PCI_IO_PROTOCOL.Attributes** - Disable original attributes the attributes should really be disabled	1. Call **Attributes()** with EfiPciIoAttributeOperationDisable with supported attributes 2. Call **Attributes()** with **EfiPciIoAttributeOperationGet** to get the attributes 3. Check if the attributes is 0
5.8.2.16.12	0x00c4352a, 0x0747, 0x4175, 0x8d, 0xa6, 0xd1, 0xad, 0xc7, 0x30, 0x31, 0xf4	**EFI_PCI_IO_PROTOCOL.Attributes** - Call **Attributes()** set original attributes return status must be **EFI_SUCCESS**	1. Call **Attributes()** with **EfiPciIoAttributeOperationSet** with original attributes The return status should be **EFI_SUCCESS**

Number	GUID	Assertion	Test Description
5.8.2.16.13	0x7ba1d37a, 0xa654, 0x4738, 0x96, 0x98, 0x11, 0x1b, 0x4b, 0x43, 0xad, 0x6c	**EFI_PCI_IO_PROTOCOL.Attributes** - Set original attributes the attributes should really be set	1. Call **Attributes()** with **EfiPciIoAttributeOperationSet** with original attributes 2. Call **Attributes()** with **EfiPciIoAttributeOperationGet** to get the attributes 3. Check if the attributes is the same as original attributes
5.8.2.16.14	0xca3478fa, 0x7a9a, 0x4452, 0x93, 0x23, 0x98, 0xda, 0xe1, 0xf9, 0x17, 0xde	**EFI_PCI_IO_PROTOCOL.Attributes** - With Operation as ***EfiPciIoAttributeOperationMaximum*** status must be **EFI_INVALID_PARAMETER**	1. Call **Attributes()** with Operation as **EfiPciIoAttributeOperationMaximum**. The return status must be **EFI_INVALID_PARAMETER**
5.8.2.16.15	0xf09e9c22, 0xd061, 0x4a52, 0xa6, 0xea, 0xa9, 0x4a, 0x90, 0x2e, 0x15, 0x0e	**EFI_PCI_IO_PROTOCOL.Attributes** - With Operation as ***EfiPciIoAttributeOperationMaximum*** + 1 status must be **EFI_INVALID_PARAMETER**	1. Call **Attributes()** with Operation as **EfiPciIoAttributeOperationMaximum** + 1. The return status must be **EFI_INVALID_PARAMETER**
5.8.2.16.16	0x1a5371a2, 0x9f8f, 0x4a0a, 0x90, 0x3c, 0x61, 0xca, 0xf0, 0x47, 0xc4, 0x30	**EFI_PCI_IO_PROTOCOL.Attributes** - With Operation as -1 the status must be **EFI_INVALID_PARAMETER**	1. Call **Attributes()** with Operation as -1. The return status must be **EFI_INVALID_PARAMETER**
5.8.2.16.17	0x63c39f67, 0xb02f, 0x4f78, 0x88, 0x49, 0x63, 0x3a, 0xa9, 0x0b, 0xfd, 0xd8	**EFI_PCI_IO_PROTOCOL.Attributes** - With Operation as **EfiPciIoAttributeOperationGet** and *Result* as **NULL** then the status must be **EFI_INVALID_PARAMETER**	1. Call **Attributes()** with Operation as **EfiPciIoAttributeOperationGet** and *Result* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**

Number	GUID	Assertion	Test Description
5.8.2.16.18	0xacfb1410, 0x3824, 0x42f0, 0x89, 0xfe, 0x93, 0x0c, 0xda, 0xb7, 0xe0, 0x3a	EFI_PCI_IO_PROTOCOL.Attributes - With Operation as EfiPciIoAttributeOperationSupported and *Result* as NULL, status is EFI_INVALID_PARAMETER	1. Call Attributes() with Operation as EfiPciIoAttributeOperationSupported and *Result* as NULL. The return status must be EFI_INVALID_PARAMETER
5.8.2.16.19	0xabcd2d94, 0x9389, 0x49a5, 0x91, 0xd7, 0x91, 0x83, 0x0b, 0x80, 0xfe, 0xc2	EFI_PCI_IO_PROTOCOL.Attributes - Setting unsupported *Attributes* returns a status of EFI_UNSUPPORTED	1. Find unsupported attributes by this device 2. Call Attributes() with Operation as EfiPciIoAttributeOperationSet and unsupported *Attributes*. The return status must be EFI_UNSUPPORTED
5.8.2.16.20	0xdbe5ef54, 0x5b5e, 0x45e8, 0x9f, 0x8b, 0x9d, 0xa5, 0x72, 0xdb, 0xcd, 0xb7	EFI_PCI_IO_PROTOCOL.Attributes - Enabling unsupported *Attributes* returns a status of EFI_UNSUPPORTED	1. Find unsupported attributes by this device 2. Call Attributes() with Operation as EfiPciIoAttributeOperationEnable and unsupported *Attributes*. The return status must be EFI_UNSUPPORTED
5.8.2.16.21	0x781416ce, 0xc545, 0x4542, 0xb5, 0xd8, 0xbc, 0xc0, 0xc4, 0xe0, 0x2a, 0x52	EFI_PCI_IO_PROTOCOL.Attributes - Disabling unsupported *Attributes* returns a status of EFI_UNSUPPORTED	1. Find unsupported attributes by this device 2. Call Attributes() with Operation as EfiPciIoAttributeOperationDisable and unsupported *Attributes*. The return status must be EFI_UNSUPPORTED

10.2.17 GetBarAttributes()

Number	GUID	Assertion	Test Description
5.8.2.17.1	0xbc76b1a7, 0x767b, 0x4f5c, 0x94, 0x03, 0x34, 0x40, 0xfb, 0xd9, 0x40, 0x95	EFI_PCI_IO_PROTOCOL.GetBarAttributes - Calling GetBarAttributes() returns a status of EFI_SUCCESS.	1. Call GetBarAttributes() with a valid BAR Index and a valid *Resources* pointer. The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.8.2.17.2	0x8414d9a1, 0x0339, 0x4d7c, 0xa2,0xa4, 0x45,0x3d, 0xd6,0x8d, 0x6b,0x5f	**EFI_PCI_IO_PROTOCOL.GetBarAttributes** - Calling **GetBarAttributes()** with only *Supports* is **NULL** returns status of **EFI_SUCCESS**.	1. Call **GetBarAttributes()** with a valid BAR Index and **NULL** *Supports*. The return status should be **EFI_SUCCESS**.
5.8.2.17.3	0x211c1b15, 0xc4ce, 0x452d, 0x96, 0x93, 0xec, 0xf4, 0xc2, 0x3d, 0x20, 0xfe	**EFI_PCI_IO_PROTOCOL.GetBarAttributes** - Calling **GetBarAttributes()** with only *Resource* is **NULL** returns a status of **EFI_SUCCESS**.	1. Call **GetBarAttributes()** with a valid BAR Index and **NULL** *Resources* pointer. The return status should be **EFI_SUCCESS**.
5.8.2.17.4	0xcb909d56, 0x1d18, 0x44b5, 0xb0, 0x30, 0xa2, 0x58, 0x30, 0x9e, 0xd6, 0x6c	**EFI_PCI_IO_PROTOCOL.GetBarAttributes** - The Resource Descriptor List is valid.	1. Call **GetBarAttributes()** with a valid BAR Index and a valid *Resources* pointer. 2. Check that the returned resource descriptor is valid.
5.8.2.17.5	0xc0d61a6d, 0x5d07, 0x4748, 0x9f, 0x14, 0x78, 0x00, 0xb6, 0xcf, 0x4b, 0x47	**EFI_PCI_IO_PROTOCOL.GetBarAttributes** - The attributes are in Device Supported Attributes.	1. Call **GetBarAttributes()** with a valid BAR Index and a valid *Resources* pointer. 2. Call **Attributes()** with **EfiPciIoAttributeOperationSupported** to get the supported attributes of the PCI controller. 3. Check that the current attributes are a subset of Supported attributes.
5.8.2.17.6	0x50f8ec56, 0xc28c, 0x417c, 0x8f, 0x43, 0x43, 0xfd, 0xfc, 0xbd, 0x4e, 0xdf	**EFI_PCI_IO_PROTOCOL.GetBarAttributes** - With invalid BAR Index the status is **EFI_UNSUPPORTED**.	1. Call **GetBarAttributes()** with invalid BAR Index. The return status must be **EFI_UNSUPPORTED**.
5.8.2.17.7	0xf52eed93, 0x6c9d, 0x4008, 0xad, 0x9d, 0xe9, 0xab, 0xc8, 0xa4, 0x88, 0x01	**EFI_PCI_IO_PROTOCOL.GetBarAttributes** - With both *Supports* and *Resources* as **NULL** status is **EFI_INVALID_PARAMETER**.	1. Call **GetBarAttributes()** with both *Supports* and *Resources* as **NULL**. The return status must be **EFI_INVALID_PARAMETER**.

10.2.18 SetBarAttributes()

Number	GUID	Assertion	Test Description
5.8.2.18.1	0x51ec0763, 0x0edb, 0x4ad3, 0xb1, 0x0c, 0x2d, 0x3f, 0x88, 0x34, 0x78, 0x44	`EFI_PCI_IO_PROTOCOL.SetBarAttributes` - Calling `SetBarAttributes()` returns a status of `EFI_SUCCESS`.	1. Call `GetBarAttributes()` with a valid BAR Index to get the BAR supported attributes resource. 2. Call `SetBarAttributes()` with BAR Supported attributes and resource information. The return status should be `EFI_SUCCESS`.
5.8.2.18.2	0x9cbd1e01, 0x86a4, 0x4b9f, 0xbb, 0x00, 0x3e, 0xff, 0xfb, 0x35, 0xf3, 0xbd	`EFI_PCI_IO_PROTOCOL.SetBarAttributes` - With invalid BAR Index, the status is `EFI_UNSUPPORTED`.	1. Call `SetBarAttributes()` with invalid BAR Index. The return status must be `EFI_UNSUPPORTED`.
5.8.2.18.3	0x445e37a9, 0xc8e7, 0x402b, 0xb7, 0xf8, 0x93, 0x96, 0xa0, 0xbd, 0x5e, 0xc5	`EFI_PCI_IO_PROTOCOL.SetBarAttributes` - With *`Offset`* as `NULL`, the status is `EFI_INVALID_PARAMETER`.	1. Call `SetBarAttributes()` with *`Offset`* as `NULL`. The return status must be `EFI_INVALID_PARAMETER`.
5.8.2.18.4	0x32edd10b, 0x4a81, 0x4a98, 0x8b, 0x7a, 0xef, 0x1b, 0x9a, 0xe8, 0x25, 0x69	`EFI_PCI_IO_PROTOCOL.SetBarAttributes` - With *`Length`* as `NULL` the status is `EFI_INVALID_PARAMETER`.	1. Call `SetBarAttributes()` with *`Length`* as `NULL`. The return status must be `EFI_INVALID_PARAMETER`.
5.8.2.18.5	0xfbb0d8fc, 0xffcf, 0x4562, 0xba, 0x86, 0x1f, 0x9b, 0x41, 0x45, 0x1f, 0x9c	`EFI_PCI_IO_PROTOCOL.SetBarAttributes` - With *`Offset`* + *`Length`* out of the BAR resource range, the status is `EFI_UNSUPPORTED`.	1. Call `SetBarAttributes()` with *`Offset`* + *`Length`* out of the BAR resourcde range. The return status must be `EFI_UNSUPPORTED`.
5.8.2.18.6	0x48602f8b, 0xbb69, 0x4421, 0xb0, 0x21, 0x5a, 0x10, 0x78, 0x5b, 0xba, 0xf9	`EFI_PCI_IO_PROTOCOL.SetBarAttributes` - With unsupported *`Attributes`* the status is `EFI_UNSUPPORTED`.	1. Find unsupported attributes by this device 2. Call `SetBarAttributes()` with unsupported *`Attributes`*. The return status must be `EFI_UNSUPPORTED`.

11 Protocols USB Support Test

11.1 EFI_USB2_HC_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_USB2_HC_PROTOCOL Section.

Most of functionalities rely on the real USB devices. They are not covered in below checkpoints.

11.1.1 GetCapability()

Number	GUID	Assertion	Test Description
5.21.1.1.1	0xbe0fffbd, 0xc5cb, 0x4ab7, 0xa0, 0x8a, 0x79, 0xd1, 0x02, 0xb3, 0x5f, 0xf8	**EFI_USB2_HC_PROTOCOL. GetCapability** - **GetCapability()** returns **EFI_INVALID_PARAMETER** with a *MaxSpeed* value of **NULL**.	1. Call **GetCapability()** with a *MaxSpeed* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.1.2	0x6dd53bd5, 0x463b, 0x46a7, 0xb0, 0x98, 0x06, 0xa6, 0xf6, 0xa5, 0x62, 0xdd	**EFI_USB2_HC_PROTOCOL. GetCapability** - **GetCapability ()** returns **EFI_INVALID_PARAMETER** with a *PortNumber* value of **NULL**.	1. Call **GetCapability ()** with a *PortNumber* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.1.3	0x0ffa5751, 0x96dd, 0x4a70, 0xa1, 0x01, 0x63, 0x66, 0x7b, 0x15, 0xcc, 0xf5	**EFI_USB2_HC_PROTOCOL. GetCapability** - **GetCapability ()** returns **EFI_INVALID_PARAMETER** with an *Is64BitCapable* value of **NULL**.	1. Call **GetCapability ()** with an *Is64BitCapable* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.

11.1.2 Reset()

Number	GUID	Assertion	Test Description
5.21.1.2.1	0xf8dd84cb, 0x72a2, 0x4cab, 0xac, 0x2e, 0x11, 0x6f, 0x3c, 0x0d, 0x5d, 0xb5	**EFI_USB2_HC_PROTOCOL.Reset - Reset()** returns **EFI_SUCCESS** with *Attributes* values of **EFI_USB_HC_RESET_GLOBAL**.	1. Call **Reset()** with *Attributes* values of **EFI_USB_HC_RESET_GLOBAL**. The return status should be **EFI_SUCCESS**. 2. Call **GetState()** to get the state of the USB host controller. The controller should be in halt state.
5.21.1.2.2	0x3bdb0674, 0x621b, 0x4319, 0xb2, 0x4f, 0xc6, 0x1a, 0xd4, 0x09, 0x73, 0xd0	**EFI_USB2_HC_PROTOCOL.Reset - Reset()** returns **EFI_SUCCESS** with *Attributes* values of **EFI_USB_HC_RESET_HOST_CONTROLLER**.	1. Call **Reset()** with *Attributes* values of **EFI_USB_HC_RESET_HOST_CONTROLLER**. The return status should be **EFI_SUCCESS**. 2. Call **GetState()** to get the state of the USB host controller. The controller should be in halt state.
5.21.1.2.3	0xd243c0fd, 0x7654, 0x4400, 0xb3, 0x4a, 0xe3, 0x09, 0x8f, 0x9e, 0x5e, 0xd4	**EFI_USB2_HC_PROTOCOL.Reset - Reset()** returns **EFI_SUCCESS** with *Attributes* values of **EFI_USB_HC_RESET_GLOBAL | EFI_USB_HC_RESET_HOST_CONTROLLER**.	1. Call **Reset()** with *Attributes* values of **EFI_USB_HC_RESET_GLOBAL | EFI_USB_HC_RESET_HOST_CONTROLLER**. The return status should be **EFI_SUCCESS**. 2. Call **GetState()** to get the state of the USB host controller. The controller should be in halt state.
5.21.1.2.4	0xa4f18be1, 0x15f2, 0x424f, 0xa6, 0xdb, 0x58, 0x6e, 0x0d, 0x54, 0x80, 0x25	**EFI_USB2_HC_PROTOCOL.Reset - Reset()** returns **EFI_SUCCESS** with *Attributes* values of **EFI_USB_HC_RESET_GLOBAL_DEBUG**.	1. Call **Reset()** with *Attributes* values of **EFI_USB_HC_RESET_GLOBAL_DEBUG**. The return status should be **EFI_SUCCESS**. 2. Call **GetState()** to get the state of the USB host controller. The controller should be in halt state.
5.21.1.2.5	0xe2df74c7, 0x7aea, 0x488c, 0xb9, 0xa2, 0x71, 0x94, 0xb2, 0x5f, 0xf3, 0x8b	**EFI_USB2_HC_PROTOCOL.Reset - Reset()** returns **EFI_SUCCESS** with *Attributes* values of **EFI_USB_HC_RESET_HOST_CONTROLLER_DEBUG**.	1. Call **Reset()** with an *Attributes* value of **EFI_USB_HC_RESET_HOST_CONTROLLER_DEBUG**. The return status should be **EFI_SUCCESS**. 2. Call **GetState()** to get the state of the USB host controller. The controller should be in halt state.

Number	GUID	Assertion	Test Description
5.21.1.2.6	0xda7ef15c, 0x01a4, 0x4004, 0x8c, 0x7a, 0x33, 0xc7, 0x89, 0x47, 0xd9, 0xfc	EFI_USB2_HC_PROTOCOL.Reset - Reset() returns EFI_SUCCESS with an *Attributes* value of EFI_USB_HC_RESET_GLOBAL_DEBUG | EFI_USB_HC_RESET_HOST_CONTROLLER_DEBUG.	1. Call Reset() with an *Attributes* value of EFI_USB_HC_RESET_GLOBAL_DEBUG | EFI_USB_HC_RESET_HOST_CONTROLLER_DEBUG. The return status should be EFI_SUCCESS. 2. Call GetState() to get the state of this USB host controller. This controller should be in halt state.
5.21.1.2.7	0xd2e6a8f0, 0x6c97, 0x4134, 0x81, 0x2e, 0x25, 0xf1, 0x75, 0x18, 0x6a, 0xe4	EFI_USB2_HC_PROTOCOL.Reset - Reset() returns EFI_INVALID_PARAMETER with an invalid *Attributes*.	1. Call Reset() with an invalid *Attributes* value of 0. The return status should be EFI_INVALID_PARAMETER.

11.1.3 GetState()

5.21.1.3.1	0x19be62be, 0xf20c, 0x4fa2, 0x89, 0xcc, 0x3a, 0x89, 0x39, 0x48, 0x4d, 0x86	EFI_USB2_HC_PROTOCOL.GetState - GetState() returns EFI_SUCCESS while the host controller is in halt state.	1. Call SetState() with a *State* value of EfiUsbHcStateHalt. The return status should be EFI_SUCCESS. 2. Call GetState() to get the state of this USB host controller. This controller should be in halt state.
5.21.1.3.2	0xc2b1cb6a, 0x66b4, 0x4c6d, 0xb9, 0x0a, 0xc9, 0x5d, 0x27, 0xd6, 0xa5, 0xd1	EFI_USB2_HC_PROTOCOL.GetState - GetState() returns EFI_SUCCESS while the host controller is in an operational state.	1. Call SetState() with a *State* value of EfiUsbHcStateOperational. The return status should be EFI_SUCCESS. 2. Call GetState() to get the state of this USB host controller. This controller should be in an Operational state.
5.21.1.3.3	0x95e913a0, 0x5ca9, 0x4edb, 0x92, 0x4f, 0xaa, 0x2f, 0x18, 0x9b, 0x57, 0x6a	EFI_USB2_HC_PROTOCOL.GetState - GetState() returns EFI_SUCCESS while the host controller is in suspend state.	1. Call SetState() with a *State* value of EfiUsbHcStateSuspend. The return status should be EFI_SUCCESS. 2. Call GetState() to get the state of this USB host controller. This controller should be in Suspend state.
5.21.1.3.4	0xbc1b8f2e, 0xf1aa, 0x446f, 0x81, 0x78, 0x6e, 0x4e, 0xd5, 0x53, 0x02, 0x08	EFI_USB2_HC_PROTOCOL.GetState - GetState() returns EFI_INVALID_PARAMETER with a State value of NULL.	1. Call GetState() with a *State* value of NULL. The return status should be EFI_INVALID_PARAMETER.

11.1.4 SetState()

Number	GUID	Assertion	Test Description
5.21.1.4.1	0x5d2282fe, 0xc37c, 0x4901, 0xbb, 0xf7, 0xf1, 0xb6, 0xf0, 0xae, 0x82, 0x91	**EFI_USB2_HC_PROTOCOL.SetState** - **SetState()** returns **EFI_SUCCESS** when changing the state from halt to halt.	1. Call **SetState()** with a *State* value of **EfiUsbHcStateHalt**. The return status should be **EFI_SUCCESS**. 2. Call **SetState()** with a *State* value of **EfiUsbHcStateHalt** again. The return status should be **EFI_SUCCESS**. 3. Call **GetState()** to get the state of this USB host controller. This controller should be in halt state.
5.21.1.4.2	0x6f6e6713, 0x07dc, 0x4413, 0x85, 0x05, 0xee, 0x69, 0x9e, 0x32, 0x69, 0x27	**EFI_USB2_HC_PROTOCOL.SetState** - **SetState()** returns **EFI_SUCCESS** when changing the state from halt state to operational state.	1. Call **SetState()** with a *State* value of **EfiUsbHcStateHalt**. The return status should be **EFI_SUCCESS**. 2. Call **SetState()** with a *State* value of **EfiUsbHcStateOperational** again. The return status should be **EFI_SUCCESS**. 3. Call **GetState()** to get the state of this USB host controller. This controller should be in an Operational state.
5.21.1.4.3	0x49ca37bc, 0x208d, 0x4feb, 0xa6, 0xd9, 0x68, 0xa3, 0x69, 0xca, 0xb3, 0xf1	**EFI_USB2_HC_PROTOCOL.SetState** - **SetState()** returns **EFI_SUCCESS** when changing the state from halt state to suspend state.	1. Call **SetState()** with a *State* value of **EfiUsbHcStateHalt**. The return status should be **EFI_SUCCESS**. 2. Call **SetState()** with a *State* value of **EfiUsbHcStateSuspend** again. The return status should be **EFI_SUCCESS**. 3. Call **GetState()** to get the state of this USB host controller. This controller should be in Suspend state.
5.21.1.4.4	0xa4663706, 0xd0c0, 0x45d7, 0x9a, 0x9d, 0x5e, 0x0e, 0xf8, 0xba, 0x2c, 0x26	**EFI_USB2_HC_PROTOCOL.SetState** - **SetState()** returns **EFI_SUCCESS** when changing the state from operational state to operational state.	1. Call **SetState()** with a *State* value of **EfiUsbHcStateOperational**. The return status should be **EFI_SUCCESS**. 2. Call **SetState()** with a *State* value of **EfiUsbHcStateOperational** again. The return status should be **EFI_SUCCESS**. 3. Call **GetState()** to get the state of this USB host controller. This controller should be in an Operational state.

Number	GUID	Assertion	Test Description
5.21.1.4.5	0xa9b73b45, 0xb3ca, 0x4579, 0x87, 0x38, 0xb3, 0xcc, 0xc4, 0x50, 0x09, 0x97	`EFI_USB2_HC_PROTOCOL.SetState` - `SetState()` returns `EFI_SUCCESS` when changing the state from operational state to halt state.	1. Call `SetState()` with a *`State`* value of `EfiUsbHcStateOperational`. The return status should be `EFI_SUCCESS`. 2. Call `SetState()` with a *`State`* value of `EfiUsbHcStateHalt` again. The return status should be `EFI_SUCCESS`. 3. Call `GetState()` to get the state of this USB host controller. This controller should be in halt state.
5.21.1.4.6	0x54936ebc, 0x9732, 0x4d9f, 0x83, 0x5c, 0x95, 0x77, 0xf5, 0xdb, 0x0e, 0xb1	`EFI_USB2_HC_PROTOCOL.SetState` - `SetState()` returns `EFI_SUCCESS` when changing the state from operational state to suspend state.	1. Call `SetState()` with a *`State`* value of `EfiUsbHcStateOperational`. The return status should be `EFI_SUCCESS`. 2. Call `SetState()` with a *`State`* value of `EfiUsbHcStateSuspend` again. The return status should be `EFI_SUCCESS`. 3. Call `GetState()` to get the state of this USB host controller. This controller should be in Suspend state.
5.21.1.4.7	0x9da57b17, 0x7841, 0x423a, 0xb1, 0xf8, 0x6d, 0x61, 0xf0, 0xd3, 0x17, 0xf0	`EFI_USB2_HC_PROTOCOL.SetState` - `SetState()` returns `EFI_SUCCESS` when changing the state from suspend state to suspend state.	1. Call `SetState()` with a *`State`* value of `EfiUsbHcStateSuspend`. The return status should be `EFI_SUCCESS`. 2. Call `SetState()` with a *`State`* value of `EfiUsbHcStateSuspend` again. The return status should be `EFI_SUCCESS`. 3. Call `GetState()` to get the state of this USB host controller. This controller should be in Suspend state.
5.21.1.4.8	0x5b4bf27e, 0xad64, 0x41a4, 0xa9, 0x8b, 0xd2, 0xb0, 0x7d, 0x32, 0xbb, 0xa3	`EFI_USB2_HC_PROTOCOL.SetState` - `SetState()` returns `EFI_SUCCESS` when changing the state from suspend state to halt state.	1. Call `SetState()` with a *`State`* value of `EfiUsbHcStateSuspend`. The return status should be `EFI_SUCCESS`. 2. Call `SetState()` with a *`State`* value of `EfiUsbHcStateHalt` again. The return status should be `EFI_SUCCESS`. 3. Call `GetState()` to get the state of this USB host controller. This controller should be in halt state.
5.21.1.4.9	0xc12e9ca0, 0x0e9c, 0x4204, 0xaa, 0xc3, 0x6d, 0x12, 0x33, 0x1b, 0x28, 0x9b	`EFI_USB2_HC_PROTOCOL.SetState` - `SetState()` returns `EFI_SUCCESS` when changing the state from suspend state to operational state.	1. Call `SetState()` with a *`State`* value of `EfiUsbHcStateSuspend`. The return status should be `EFI_SUCCESS`. 2. Call `SetState()` with a *`State`* value of `EfiUsbHcStateOperational` again. The return status should be `EFI_SUCCESS`. 3. Call `GetState()` to get the state of this USB host controller. This controller should be in an Operational state.

Number	GUID	Assertion	Test Description
5.21.1.4.10	0x5168c4ef, 0x91f4, 0x48c5, 0x88, 0x1f, 0xf8, 0x01, 0x80, 0xd2, 0x98, 0x07	EFI_USB2_HC_PROTOCOL.SetState - SetState() returns EFI_INVALID_PARAMETER with an invalid State.	1. Call SetState() with an invalid State value of -1. The return status should be EFI_INVALID_PARAMETER. 2. Call SetState() with an invalid State value of EfiUsbHcStateMaximum. The return status should be EFI_INVALID_PARAMETER.

11.1.5 ControlTransfer()

Number	GUID	Assertion	Test Description
5.21.1.5.1	0x36308487, 0x3a2c, 0x48fa, 0x91, 0xed, 0xec, 0xc3, 0x59, 0xd0, 0x78, 0x46	EFI_USB2_HC_PROTOCOL. ControlTransfer - ControlTransfer() returns EFI_INVALID_PARAMETER with an invalid TransferDirection.	1. Call ControlTransfer() with an invalid TransferDirection value (-1). The return status should be EFI_INVALID_PARAMETER. 2. Call ControlTransfer() with an invalid TransferDirection value (0x7FFFFFFF). The return status should be EFI_INVALID_PARAMETER.
5.21.1.5.2	0x26532efd, 0x62ab, 0x4d60, 0x9c, 0xd8, 0x14, 0xb7, 0x9d, 0x48, 0x8e, 0xa1	EFI_USB2_HC_PROTOCOL. ControlTransfer - ControlTransfer() returns EFI_INVALID_PARAMETER with a invalid Data and DataLength values.	1. Call ControlTransfer() with an invalid Data (value of) and TransferDirection is either EfiUsbDataIn or EfiUsbDataOut. The return status should be EFI_INVALID_PARAMETER. 2. Call ControlTransfer() with an invalid DataLength (value of 0) and TransferDirection is either EfiUsbDataIn or EfiUsbDataOut. The return status should be EFI_INVALID_PARAMETER. 3. Call ControlTransfer() with an invalid Data (not value of NULL) and TransferDirection value of EfiUsbNoData.The return status should be EFI_INVALID_PARAMETER. 4. Call ControlTransfer() with an invalid DataLength(value of 1)and TransferDirection value of EfiUsbNoData.The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.21.1.5.3	0x28f002fd, 0x3797, 0x46cb, 0xaf, 0x66, 0xd5, 0xb4, 0x27, 0x23, 0x1b, 0x7a	**EFI_USB2_HC_PROTOCOL.ControlTransfer** - **ControlTransfer()** returns **EFI_INVALID_PARAMETER** with an *Request* value of **NULL**.	1. Call **ControlTransfer()** with an invalid *Request* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.5.4	0xddf99154, 0x12ea, 0x4c99, 0x9a, 0x49, 0x6a, 0x1c, 0x51, 0xc2, 0x7a, 0x77	**EFI_USB2_HC_PROTOCOL.ControlTransfer** - **ControlTransfer()** returns **EFI_INVALID_PARAMETER** with an invalid *MaximumPacketLength*.	1. Call **ControlTransfer()** with an invalid *MaximumPacketLength* (value is not 8) when *DeviceSpeed* is **EFI_USB_SPEED_LOW**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.5.5	0xc258056b, 0x13ae, 0x4839, 0xbb, 0xda, 0xa0, 0x1f, 0x5c, 0x14, 0x0a, 0x51	**EFI_USB2_HC_PROTOCOL.ControlTransfer** - **ControlTransfer()** returns **EFI_INVALID_PARAMETER** with an invalid *MaximumPacketLength*.	1. Call **ControlTransfer()** with an invalid *MaximumPacketLength* (value of 128 not 8/16/32/64) when *DeviceSpeed* is **EFI_USB_SPEED_FULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.5.6	0x5f6973f9, 0x9d75, 0x4e26, 0x8a, 0x30, 0xb5, 0xc2, 0x0e, 0x47, 0xf0, 0xb3	**EFI_USB2_HC_PROTOCOL.ControlTransfer** - **ControlTransfer()** returns **EFI_INVALID_PARAMETER** with an invalid *MaximumPacketLength*.	1. Call **ControlTransfer()** with an invalid *MaximumPacketLength* (value of 128 not 8/16/32/64) when *DeviceSpeed* is **EFI_USB_SPEED_HIGH**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.5.7	0x66a39c82, 0xfb44, 0x4057, 0xbb, 0xd7, 0x4b, 0x24, 0x30, 0xff, 0x19, 0xa9	**EFI_USB2_HC_PROTOCOL.ControlTransfer** - **ControlTransfer()** returns **EFI_INVALID_PARAMETER** with an invalid *MaximumPacketLength*.	1. Call **ControlTransfer()** with an invalid *MaximumPacketLength* (value of 256 not 512) when *DeviceSpeed* is **EFI_USB_SPEED_SUPER**. The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.21.1.5.8	0xf63896ea, 0x5143, 0x4b7a, 0x93, 0x51, 0x63, 0xb5, 0xb5, 0x95, 0x81, 0x5c	**EFI_USB2_HC_PROTOCOL. ControlTransfer** - **ControlTransfer ()** returns **EFI_INVALID_PARAMETER** with a *TransferResult* value of **NULL**.	1. Call **ControlTransfer()** with an invalid *TransferResult* (value of **NULL**). The return status should be **EFI_INVALID_PARAMETER**.

11.1.6 BulkTransfer()

Number	GUID	Assertion	Test Description
5.21.1.6.1	0x0498c13e, 0xc21b, 0x4c4e, 0x95, 0xd2, 0x11, 0x9a, 0x10, 0x07, 0x51, 0x02	**EFI_USB2_HC_PROTOCOL. BulkTransfer** - **BulkTransfer ()** returns **EFI_INVALID_PARAMETER** with a *Data* value of **NULL**.	1. Call **BulkTransfer()** with an invalid *Data* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.6.2	0x2a1df585, 0xf82a, 0x42ab, 0x97, 0x4f, 0xfe, 0xfb, 0xf7, 0x89, 0xe6, 0xf5	**EFI_USB2_HC_PROTOCOL. BulkTransfer** - **BulkTransfer ()** returns **EFI_INVALID_PARAMETER** with a *DataLength* value of 0.	1. Call **BulkTransfer()** with an invalid *DataLength* value of 0. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.6.3	0x26ad2292 , 0x449b, 0x4545, 0x80, 0xaa, 0x13, 0x39, 0x13, 0x15, 0x04, 0xf6	**EFI_USB2_HC_PROTOCOL. BulkTransfer** - **BulkTransfer ()** returns **EFI_INVALID_PARAMETER** with a *DeviceSpeed* value of **EFI_USB_SPEED_LOW**.	1. Call **BulkTransfer()** with an invalid *DeviceSpeed* value of **EFI_USB_SPEED_LOW**. The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.21.1.6.4	0x1d89742e, 0xd026, 0x47d7, 0xa4, 0xcb, 0xe0, 0xb6, 0xd9, 0xc3, 0xd9, 0x54	**EFI_USB2_HC_PROTOCOL. BulkTransfer** - **BulkTransfer ()** returns **EFI_INVALID_PARAMETER** with an invalid *MaximumPacketLength*.	1. Call **BulkTransfer()** with an invalid *MaximumPacketLength* (value of 65) when *DeviceSpeed* is **EFI_USB_SPEED_FULL**. The return status should be **EFI_INVALID_PARAMETER**. 2. Call **BulkTransfer()** with an invalid *MaximumPacketLength* (value of 513) when *DeviceSpeed* is **EFI_USB_SPEED_HIGH**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **BulkTransfer()** with an invalid *MaximumPacketLength* (value of 1025) when DeviceSpeed is **EFI_USB_SPEED_SUPER**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.6.5	0xbc90875e, 0x0a8b, 0x4e3c, 0xbb, 0xf2, 0x5a, 0x43, 0x40, 0x3a, 0x6b, 0x05	**EFI_USB2_HC_PROTOCOL. BulkTransfer** - **BulkTransfer ()** returns **EFI_INVALID_PARAMETER** with a *DataToggle* value other than 0 and 1.	1. Call **BulkTransfer()** with an invalid *DataToggle* (value of 2*)*. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.6.6	0x0dfea5a1, 0xf82a, 0x41a5, 0xbf, 0x67, 0xea, 0x89, 0xed, 0x74, 0x61, 0x21	**EFI_USB2_HC_PROTOCOL. BulkTransfer** - **BulkTransfer ()** returns **EFI_INVALID_PARAMETER** with a *TransferResult* value of **NULL**.	1. Call **BulkTransfer()** with an invalid *TransferResult* (value of **NULL**). The return status should be **EFI_INVALID_PARAMETER**.

11.1.7 AsyncInterruptTransfer()

Number	GUID	Assertion	Test Description
5.21.1.7.1	0xec3427c4, 0xe4df, 0x4646, 0x8b, 0x63, 0xdc, 0x0b, 0x7d, 0xc0, 0x0d, 0xdd	**EFI_USB2_HC_PROTOCOL.AsyncInterruptTransfer** - **AsyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with a *EndPointAddress* value other than **EfiUsbDataIn**.	1. Call **AsyncInterruptTransfer ()** with an *EndPointAddress* value other than **EfiUsbDataIn**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.7.2	0xc0cddbce, 0x4853, 0x4d71, 0xad, 0xe1, 0x59, 0x94, 0x90, 0x7c, 0x31, 0xcc	**EFI_USB2_HC_PROTOCOL.AsyncInterruptTransfer** - **AsyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with a new, invalid transfer.	1. Call **AsyncInterruptTransfer ()** with the *IsNewTransfer* value of **TRUE** and *DataLength* value of `0`. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.7.3	0xaf26077c, 0x75e5, 0x4fbc, 0xad, 0x5e, 0x99, 0x3b, 0xce, 0x66, 0xb5, 0xc5	**EFI_USB2_HC_PROTOCOL.AsyncInterruptTransfer** - **AsyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with a new, invalid transfer.	1. Call **AsyncInterruptTransfer ()** with the *IsNewTransfer* value of **TRUE** and a *DataToggle* value other than `0` and `1`. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.7.4	0xccd35e94, 0x51db, 0x4118, 0xa8, 0xd4, 0x40, 0xbd, 0x2e, 0xee, 0x77, 0xd9	**EFI_USB2_HC_PROTOCOL.AsyncInterruptTransfer** - **AsyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with new, invalid transfer.	1. Call **AsyncInterruptTransfer ()** with the *IsNewTransfer* value of **TRUE** and *PollingInterval* value of `0`. The return status should be **EFI_INVALID_PARAMETER**. 2. Call **AsyncInterruptTransfer ()** with the *IsNewTransfer* value of **TRUE** and *PollingInterval* value of `256`. The return status should be **EFI_INVALID_PARAMETER**.

11.1.8 SyncInterruptTransfer()

Number	GUID	Assertion	Test Description
5.21.1.8.1	0x509cb496, 0x1d63, 0x4faf, 0x8d, 0xdf, 0x00, 0xbc, 0x58, 0x05, 0x0d, 0xe6	EFI_USB2_HC_PROTOCOL.SyncInterruptTransfer - SyncInterruptTransfer () returns EFI_INVALID_PARAMETER with the *EndPointAddress* set other than EfiUsbDataIn.	1. Call SyncInterruptTransfer () with the *EndPointAddress* set other than EfiUsbDataIn. The return status should be EFI_INVALID_PARAMETER.
5.21.1.8.2	0x3a0ad565, 0xb82c, 0x450f, 0xbc, 0xe6, 0x88, 0xb3, 0xd1, 0x6a, 0xde, 0x35	EFI_USB2_HC_PROTOCOL.SyncInterruptTransfer - SyncInterruptTransfer () returns EFI_INVALID_PARAMETER with a *Data* value of NULL.	1. Call SyncInterruptTransfer () with a *Data* value of NULL. The return status should be EFI_INVALID_PARAMETER.
5.21.1.8.3	0xc139127a, 0x3797, 0x482f, 0xb3, 0x5c, 0xaa, 0xf7, 0x99, 0xbd, 0xf6, 0xc6	EFI_USB2_HC_PROTOCOL.SyncInterruptTransfer - SyncInterruptTransfer () returns EFI_INVALID_PARAMETER with a *DataLength* value of 0.	1. Call SyncInterruptTransfer () with a *DataLength* value of 0. The return status should be EFI_INVALID_PARAMETER.
5.21.1.8.4	0x14cb206c, 0x422b, 0x47ee, 0x8c, 0x4b, 0xf3, 0x16, 0xfe, 0x33, 0xda, 0xfb	EFI_USB2_HC_PROTOCOL.SyncInterruptTransfer - SyncInterruptTransfer () returns EFI_INVALID_PARAMETER with an invalid *MaximumPacketLength*.	1. Call SyncInterruptTransfer () with a *MaximumPacketLength* value of 9 and *DeviceSpeed* value of EFI_USB_SPEED_LOW. The return status should be EFI_INVALID_PARAMETER. 2. Call SyncInterruptTransfer () with a *MaximumPacketLength* value of 65 and *DeviceSpeed* value of EFI_USB_SPEED_FULL. The return status should be EFI_INVALID_PARAMETER. 3. Call SyncInterruptTransfer () with a *MaximumPacketLength* value of 3073 and *DeviceSpeed* value of EFI_USB_SPEED_HIGH. The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.21.1.8.5	0xf4353439, 0x47e4, 0x4df3, 0x85, 0xe9, 0x9e, 0xfe, 0x72, 0x3a, 0x1e, 0x4b	**EFI_USB2_HC_PROTOCOL. SyncInterruptTransfer** - **SyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with *DataToggle* pointing to a value other than 0 and 1.	1. Call **SyncInterruptTransfer ()** with *DataToggle* points to a value other than 0 and 1. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.8.6	0x81dfdb23, 0x681e, 0x4df7, 0xa7, 0x73, 0x6d, 0x41, 0x58, 0xdb, 0x88, 0x3e	**EFI_USB2_HC_PROTOCOL. SyncInterruptTransfer** - **SyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with a *TransferResult* value of **NULL**.	1. Call **SyncInterruptTransfer ()** with a *TransferResult* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.

11.1.9 IsochronousTransfer()

Number	GUID	Assertion	Test Description
5.21.1.9.1	0x74e2dcbf, 0xae9f, 0x4499, 0x82, 0x74, 0xcb, 0xbe, 0x86, 0x59, 0x5d, 0xb7	**EFI_USB2_HC_PROTOCOL. IsochronousTransfer** - **IsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with a *Data* value of **NULL**.	1. Call **IsochronousTransfer ()** with a *Data* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.9.2	0xd93babd4, 0xd7de, 0x4e87, 0x9b, 0x5c, 0x68, 0xd2, 0xa6, 0x77, 0x33, 0xc4	**EFI_USB2_HC_PROTOCOL. IsochronousTransfer** - **IsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with a *DataLength* value of **0**.	1. Call **IsochronousTransfer ()** with a *DataLength* value of **0**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.9.3	0x9b220909, 0x662c, 0x4b5e, 0x9e, 0x42, 0xdc, 0x66, 0x4c, 0xdb, 0xb1, 0x5f	**EFI_USB2_HC_PROTOCOL. IsochronousTransfer** - **IsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with a *MaxiumPacketLenth* set larger than **1023**.	1. Call **IsochronousTransfer ()** with a *MaxiumPacketLenth* value of **1024**. The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.21.1.9.4	0x68898a17, 0x5ae9, 0x456a, 0xb1, 0xe0, 0xa3, 0xc0, 0x42, 0xeb, 0x50, 0x8d	**EFI_USB2_HC_PROTOCOL. IsochronousTransfer** - **IsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with a *TransferResult* value of **NULL**.	1. Call **IsochronousTransfer ()** with a *TransferResult* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.9.5	0xfa4f5868, 0xf004, 0x4cbe, 0x88, 0x97, 0xfd, 0x6, 0xb2, 0x72, 0x76, 0x71	**EFI_USB2_HC_PROCOTOL.IsochronousTransfer** **IsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** when *DeviceSpeed* is not one of the supported values.	**IsochronousTransfer()** returns **EFI_INVALID_PARAMETER** when *DeviceSpeed* is not one of the supported values.

11.1.10 AsyncIsochronousTransfer()

Number	GUID	Assertion	Test Description
5.21.1.10.1	0x55a7ea0c, 0x9ffc, 0x47dc, 0xb7, 0x5e, 0x5c, 0xfa, 0x8c, 0xed, 0xe1, 0x53	**EFI_USB2_HC_PROTOCOL. AsyncIsochronousTransfer** - **AsyncIsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with a *Data* value of **NULL**.	1. Call **AsyncIsochronousTransfer ()** with a *Data* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.10.2	0xfa310dd6, 0x4b8a, 0x4799, 0xa5, 0xdc, 0x80, 0xe7, 0xbb, 0xe0, 0x4e, 0xac	**EFI_USB2_HC_PROTOCOL. AsyncIsochronousTransfer** - **AsyncIsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with a *DataLength* value of 0.	1. Call **AsyncIsochronousTransfer ()** with a *DataLength* value of 0. The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.21.1.10.3	0x4083742a, 0x6c43, 0x49b4, 0x8d, 0xe1, 0x7a, 0xf8, 0x0c, 0x8b, 0x02, 0x33	**EFI_USB2_HC_PROTOCOL.AsyncIsochronousTransfer** - **AsyncIsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with a *MaxiumPacketLenth* value of larger than 1023.	1. Call **AsyncIsochronousTransfer ()** with a *MaxiumPacketLenth* value of 1024. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.10.4	0x474590c4, 0x8410, 0x4871, 0x93, 0xb4, 0x2b, 0xe, 0x9f, 0xb5, 0xe8, 0x30	**USB2_HC_PROCOTOL. AsyncIsochronousTransfer – AsyncIsochronousTransfer ()** returns **EFI_INVALID_PARAMETER**when *DeviceSpeed* is not one of the supported values.	**AsyncIsochronousTransfer()** returns **EFI_INVALID_PARAMETER** when *DeviceSpeed* is not one of the supported values.

11.1.11 GetRootHubPortStatus()

Number	GUID	Assertion	Test Description
5.21.1.11.1	0x089705c5, 0xf134, 0x42b4, 0xbd, 0xeb, 0x7a, 0x74, 0xc7, 0x93, 0xa0, 0xf5	**EFI_USB2_HC_PROTOCOL.GetRootHubPortStatus** - **GetRootHubPortStatus ()** returns **EFI_INVALID_PARAMETER** with an invalid *PortNumber*.	1. Call **GetCapability()** to get the number of ports. The return status should be **EFI_SUCCESS**. 2. Call **GetRootHubPortStatus()** with *PortNumber* greater than the number of ports. The return status should be **EFI_INVALID_PARAMETER**.

11.1.12 SetRootHubPortFeature()

Number	GUID	Assertion	Test Description
5.21.1.12.1	0xf74da277, 0x4ac2, 0x422c, 0x90, 0xda, 0xb4, 0x9f, 0xc7, 0x4f, 0x2a, 0x65	**EFI_USB2_HC_PROTOCOL.SetRootHubPortFeature** - **SetRootHubPortFeature ()** returns **EFI_INVALID_PARAMETER** with an invalid *PortNumber*.	1. Call **GetRootHubPortNumber ()** to get the number of ports. The return status should be **EFI_SUCCESS**. 2. Call **SetRootHubPortFeature()** with a *PortNumber* greater than the number of ports. The return status should be **EFI_INVALID_PARAMETER**.

5.21.1.12.2	0xd7071255, 0x61db, 0x446a, 0xad, 0x65, 0x01, 0xb4, 0x54, 0x72, 0x1f, 0x80	**EFI_USB2_HC_PROTOCOL.SetRootHubPortFeature** - **SetRootHubPortFeature ()** returns **EFI_INVALID_PARAMETER** with an invalid *PortFeature*.	1. Call **GetRootHubPortNumber ()** to get the number of ports. The return status should be **EFI_SUCCESS**. 2. Call **SetRootHubPortFeature()** with a *PortFeature* not value of **EfiUsbPortEnable**, **EfiUsbPortSuspend**, **EfiUsbPortReset** nor **EfiUsbPortPower**. The return status should be **EFI_INVALID_PARAMETER**.

11.1.13 ClearRootHubPortFeature()

Number	GUID	Assertion	Test Description
5.21.1.13.1	0x88cda060, 0xbe70, 0x4c49, 0x95, 0xac, 0xae, 0xa0, 0x37, 0xfa, 0x7f, 0x51	**EFI_USB2_HC_PROTOCOL.ClearRootHubPortFeature** - **ClearRootHubPortFeature ()** returns **EFI_INVALID_PARAMETER** with an invalid *PortNumber*.	1. Call **GetRootHubPortNumber ()** to get the number of ports. The return status should be **EFI_SUCCESS**. 2. Call **ClearRootHubPortFeature()** with a *PortNumber* greater than the number of ports. The return status should be **EFI_INVALID_PARAMETER**.
5.21.1.13.2	0x59de7e7c, 0x078d, 0x4217, 0xa5, 0xfd, 0xf0, 0x1e, 0x15, 0xeb, 0xa3, 0x67	**EFI_USB2_HC_PROTOCOL.ClearRootHubPortFeature** - **ClearRootHubPortFeature ()** returns **EFI_INVALID_PARAMETER** with an invalid *PortFeature*.	1. Call **GetRootHubPortNumber ()** to get the number of ports. The return status should be **EFI_SUCCESS**. 2. Call **ClearRootHubPortFeature()** with a *PortFeature* not value of **EfiUsbPortEnable**, **EfiUsbPortSuspend**, **EfiUsbPortPower**, **EfiUsbPortConnectChange**, **EfiUsbPortResetChange**, **EfiUsbPortEnableChange**, **EfiUsbPortSuspendChange**, nor **EfiUsbPortOverCurrentChange**. The return status should be **EFI_INVALID_PARAMETER**.

11.2 EFI_USB_IO_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_USB_IO_PROTOCOL Section.

Most of functionalities rely on real USB devices. They are not covered in below checkpoints.

11.2.1 UsbControlTransfer()

Number	GUID	Assertion	Test Description
5.21.2.1.1	0xe687694c, 0xc7ec, 0x444b, 0xac, 0xc5, 0xa3, 0x56, 0xf2, 0xb6, 0x3f, 0x15	EFI_USB_IO_PROTOCOL.UsbControlTransfer - UsbControlTransfer () returns EFI_INVALID_PARAMETER with an invalid *TransferDirection*.	1. Call UsbControlTransfer() with an invalid *TransferDirection* (value of -1). The return status should be EFI_INVALID_PARAMETER. 2. Call UsbControlTransfer() with an invalid *TransferDirection* (value of 0x7FFFFFFF). The return status should be EFI_INVALID_PARAMETER.
5.21.2.1.2	0x4aa535ad, 0x7985, 0x49f3, 0x81, 0x53, 0xa3, 0xd7, 0x04, 0x1e, 0x3f, 0xd0	EFI_USB_IO_PROTOCOL.UsbControlTransfer - UsbControlTransfer () returns EFI_INVALID_PARAMETER with a *Request* value of NULL.	1. Call UsbControlTransfer() with a *Request* value of NULL. The return status should be EFI_INVALID_PARAMETER.
5.21.2.1.3	0xc6bfebde, 0xd2d6, 0x44fa, 0xa6, 0xd9, 0x9b, 0x3c, 0x88, 0x9a, 0x52, 0x81	EFI_USB_IO_PROTOCOL.UsbControlTransfer - UsbControlTransfer () returns EFI_INVALID_PARAMETER with a *Status* value of NULL.	1. Call UsbControlTransfer() with a *Status* value of NULL. The return status should be EFI_INVALID_PARAMETER.
5.21.2.1.4	0x937f99d5, 0x18ef, 0x424c, 0xb4, 0x4c, 0x54, 0xaf, 0xf6, 0x20, 0xe0, 0xdc	EFI_USB_IO_PROTOCOL.UsbControlTransfer - UsbControlTransfer () returns EFI_SUCCESS or EFI_DEVICE_ERROR when the parameter *Timeout* is 0.	1. Call UsbControlTransfer () when the parameter *Timeout* is 0. The return code must be EFI_SUCCESS or EFI_DEVICE_ERROR. .

11.2.2 UsbBulkTransfer()

Number	GUID	Assertion	Test Description
5.21.2.2.1	0xf7c2276a, 0xfcd0, 0x4aeb, 0x99, 0x79, 0xf8, 0x79, 0x24, 0xd4, 0xc4, 0x83	**EFI_USB_IO_PROTOCOL.UsbBulkTransfer** - **UsbBulkTransfer()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbBulkTransfer()** with an invalid *DeviceEndpoint* value of 0. The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.2.2	0xa0365348, 0xba4c, 0x43fe, 0xba, 0xde, 0x8e, 0x35, 0x26, 0x39, 0x7e, 0xbd	**EFI_USB_IO_PROTOCOL.UsbBulkTransfer** - **UsbBulkTransfer()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbBulkTransfer()** with an invalid *DeviceEndpoint* (value of **0x10**). The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.2.3	0xafcf7b82, 0x16ad, 0x4721, 0x92, 0x46, 0x0d, 0x7b, 0xbb, 0xbd, 0xc9, 0x3a	**EFI_USB_IO_PROTOCOL.UsbBulkTransfer** - **UsbBulkTransfer()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbBulkTransfer()** with an invalid *DeviceEndpoint* (value of **0x80**). The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.2.4	0x88c28425, 0xfbc6, 0x4441, 0x91, 0x23, 0x88, 0x83, 0x76, 0x9c, 0xed, 0x1e	**EFI_USB_IO_PROTOCOL.UsbBulkTransfer** - **UsbBulkTransfer()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbBulkTransfer()** with an invalid *DeviceEndpoint* (value of **0x90**). The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.2.5	0x383c6bd1, 0xb1f3, 0x4987, 0x8c, 0x6f, 0xb5, 0xd5, 0x23, 0xb4, 0x93, 0xc1	**EFI_USB_IO_PROTOCOL.UsbBulkTransfer** - **UsbBulkTransfer()** returns **EFI_INVALID_PARAMETER** with a *DeviceEndpoint* value of not a BULK endpoint.	1. Call **UsbBulkTransfer()** with an invalid *DeviceEndpoint* which is **not a BULK endpoint.** The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.21.2.2.6	0x141aa66b, 0x7628, 0x4275, 0xae, 0xe3, 0x8c, 0xe1, 0x17, 0x65, 0x0d, 0xcc	EFI_USB_IO_PROTOCOL.UsbBulkTransfer - UsbBulkTransfer() returns EFI_INVALID_PARAMETER with a *Data* value of NULL.	1. Call UsbBulkTransfer() with a *Data* value of NULL. The return status should be EFI_INVALID_PARAMETER.
5.21.2.2.7	0x486552a5, 0x9863, 0x4eed, 0x8b, 0x37, 0x92, 0xb3, 0x8b, 0xc3, 0xe3, 0xeb	EFI_USB_IO_PROTOCOL.UsbBulkTransfer - UsbBulkTransfer() returns EFI_INVALID_PARAMETER with a *DataLength* value of NULL.	1. Call UsbBulkTransfer() with a *DataLength* value of NULL. The return status should be EFI_INVALID_PARAMETER.
5.21.2.2.8	0x582d809f, 0x88ce, 0x4a35, 0x89, 0xc6, 0xb5, 0x79, 0xf3, 0x70, 0x54, 0x66	EFI_USB_IO_PROTOCOL.UsbBulkTransfer - UsbBulkTransfer() returns EFI_INVALID_PARAMETER with a *Status* value of NULL.	1. Call UsbBulkTransfer() with a *Status* value of NULL. The return status should be EFI_INVALID_PARAMETER.
5.21.2.2.9	0x3d1b8608, 0x8c1e, 0x4b09, 0x81, 0x0f, 0xd9, 0x5c, 0x2a, 0xd7, 0x66, 0xae	EFI_USB_IO_PROTOCOL.UsbBulkTransfer - UsbBulkTransfer() returns EFI_SUCCESS or EFI_DEVICE_ERROR when the parameter *Timeout* is 0.	1. Call UsbBulkTransfer () when the parameter *Timeout* is 0. The return code must be EFI_SUCCESS or EFI_DEVICE_ERROR.

11.2.3 UsbAsyncInterruptTransfer()

Number	GUID	Assertion	Test Description
5.21.2.3.1	0x551fbef7, 0xd9e9, 0x4302, 0xa4, 0xcd, 0x2d, 0xb6, 0x83, 0x47, 0xc9, 0x4a	EFI_USB_IO_PROTOCOL.UsbAsyncInterruptTransfer - UsbAsyncInterruptTransfer () returns EFI_INVALID_PARAMETER with an invalid *DeviceEndpoint*.	1. Call UsbAsyncInterruptTransfer () with an invalid *DeviceEndpoint* value of 0. The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.21.2.3.2	0xbb293ec7, 0x3a01, 0x493d, 0xa2, 0x2b, 0x71, 0x97, 0x48, 0x0b, 0x4f, 0x64	**EFI_USB_IO_PROTOCOL. UsbAsyncInterruptTransfer** - **UsbAsyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbAsyncInterruptTransfer ()** with an invalid *DeviceEndpoint* value of 0x10. The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.3.3	0xf2436425, 0xee55, 0x41ee, 0x81, 0x3d, 0xa4, 0x64, 0x47, 0x17, 0x18, 0xfa	**EFI_USB_IO_PROTOCOL. UsbAsyncInterruptTransfer** - **UsbAsyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbAsyncInterruptTransfer ()** with an invalid *DeviceEndpoint*(value of 0x80). The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.3.4	0x7ab9696d, 0x6687, 0x4f7f, 0xac, 0x16, 0x6a, 0x60, 0x23, 0x57, 0x41, 0xa7	**EFI_USB_IO_PROTOCOL. UsbAsyncInterruptTransfer** - **UsbAsyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbAsyncInterruptTransfer ()** with an invalid *DeviceEndpoint* (value of 0x90). The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.3.5	0x17646b64, 0x413f, 0x41cc, 0xbd, 0x8c, 0x91, 0x66, 0xe4, 0xef, 0x3e, 0x4c	**EFI_USB_IO_PROTOCOL. UsbAsyncInterruptTransfer** - **UsbAsyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with a *DeviceEndpoint* value of not an Interrupt endpoint.	1. Call **UsbAsyncInterruptTransfer ()** with a *DeviceEndpoint* value of not an Interrupt endpoint. The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.3.6	0x4d89db86, 0x4acc, 0x4ed8, 0xb8, 0xd1, 0xc3, 0xaa, 0x75, 0x08, 0xb3, 0xee	**EFI_USB_IO_PROTOCOL. UsbAsyncInterruptTransfer** - **UsbAsyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with an invalid *PollingInterval*.	1. Call **UsbAsyncInterruptTransfer ()** with an invalid *PollingInterval* value of 0. The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.21.2.3.7	0x808d9c7c, 0x2397, 0x406d, 0x97, 0x69, 0xcd, 0xeb, 0x4f, 0xde, 0x11, 0x16	`EFI_USB_IO_PROTOCOL.UsbAsyncInterruptTransfer` - `UsbAsyncInterruptTransfer ()` returns `EFI_INVALID_PARAMETER` with an invalid *`PollingInterval`*.	1. Call `UsbAsyncInterruptTransfer ()` with an invalid *`PollingInterval`* (value of `256`. The return status should be `EFI_INVALID_PARAMETER`.

11.2.4 UsbSyncInterruptTransfer()

Number	GUID	Assertion	Test Description
5.21.2.4.1	0x59735398, 0x5d31, 0x42e2, 0x8e, 0x65, 0x68, 0xbd, 0x6c, 0x1e, 0xbb, 0xb6	`EFI_USB_IO_PROTOCOL.UsbSyncInterruptTransfer` - `UsbSyncInterruptTransfer ()` returns `EFI_INVALID_PARAMETER` with an invalid *`DeviceEndpoint`*.	1. Call `UsbSyncInterruptTransfer ()` with an invalid *`DeviceEndpoint`* (value of `0`). The return status should be `EFI_INVALID_PARAMETER`.
5.21.2.4.2	0xdd2221a8, 0x7dc1, 0x4d2a, 0x85, 0x99, 0x6b, 0x86, 0x9d, 0x74, 0xf0, 0xa7	`EFI_USB_IO_PROTOCOL.UsbSyncInterruptTransfer` - `UsbSyncInterruptTransfer ()` returns `EFI_INVALID_PARAMETER` with an invalid *`DeviceEndpoint`*.	1. Call `UsbSyncInterruptTransfer ()` with an invalid *`DeviceEndpoint`* value of `0x10`. The return status should be `EFI_INVALID_PARAMETER`.
5.21.2.4.3	0x15c6a9c5, 0x9912, 0x4474, 0xac, 0xe5, 0xa3, 0x1d, 0x49, 0xde, 0x63, 0x28	`EFI_USB_IO_PROTOCOL.UsbSyncInterruptTransfer` - `UsbSyncInterruptTransfer ()` returns `EFI_INVALID_PARAMETER` with an invalid *`DeviceEndpoint`*.	1. Call `UsbSyncInterruptTransfer ()` with an invalid *`DeviceEndpoint`* value of `0x80`. The return status should be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.21.2.4.4	0x833ca596, 0xf83d, 0x455f, 0x95, 0x95, 0xe5, 0x77, 0xa6, 0xaf, 0x62, 0xdc	**EFI_USB_IO_PROTOCOL.UsbSyncInterruptTransfer** - **UsbSyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbSyncInterruptTransfer ()** with an invalid *DeviceEndpoint* value of **0x90**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.4.5	0x60a2a3d0, 0xb657, 0x413d, 0x9b, 0x1c, 0xa7, 0x2b, 0x46, 0xaa, 0xa6, 0x77	**EFI_USB_IO_PROTOCOL.UsbSyncInterruptTransfer** - **UsbSyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with a *DeviceEndpoint* value of not an Interrupt endpoint.	1. Call **UsbSyncInterruptTransfer ()** with a *DeviceEndpoint* value of not an Interrupt endpoint. The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.4.6	0xd4730bf3, 0x8b92, 0x4bcf, 0x99, 0xef, 0xe1, 0xdb, 0x65, 0xe9, 0x86, 0xec	**EFI_USB_IO_PROTOCOL.UsbSyncInterruptTransfer** - **UsbSyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with a *Data* value of **NULL**.	1. Call **UsbSyncInterruptTransfer ()** with a *Data* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.4.7	0x0dbc8bd6, 0x4405, 0x49c0, 0xa5, 0xd1, 0xbc, 0x01, 0xca, 0x61, 0x67, 0xb2	**EFI_USB_IO_PROTOCOL.UsbSyncInterruptTransfer** - **UsbSyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with a *DataLength* value of **NULL**.	1. Call **UsbSyncInterruptTransfer ()** with a *DataLength* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.4.8	0xa5e94a41, 0xc3ef, 0x4172, 0x94, 0xc2, 0xc7, 0xba, 0xa8, 0x72, 0xc3, 0x74	**EFI_USB_IO_PROTOCOL.UsbSyncInterruptTransfer** - **UsbSyncInterruptTransfer ()** returns **EFI_INVALID_PARAMETER** with a *Status* value of **NULL**.	1. Call **UsbSyncInterruptTransfer ()** with a *Status* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.

11.2.5 UsbIsochronousTransfer()

Number	GUID	Assertion	Test Description
5.21.2.5.1	0x006bb343, 0x842a, 0x417a, 0xa8, 0x23, 0x29, 0x75, 0x68, 0x9b, 0x9e, 0x2a	**EFI_USB_IO_PROTOCOL. UsbIsochronousTransfer** - **UsbIsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbIsochronousTransfer ()** with an invalid *DeviceEndpoint* value of 0. The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.5.2	0xd4f5400e, 0x3ed0, 0x4659, 0xa4, 0x80, 0xff, 0xf5, 0xeb, 0x8b, 0xae, 0x9b	**EFI_USB_IO_PROTOCOL. UsbIsochronousTransfer** - **UsbIsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbIsochronousTransfer ()** with an invalid *DeviceEndpoint* value of 0x10. The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.5.3	0xcfbc4d53, 0x07b7, 0x4366, 0x85, 0x98, 0x85, 0xf1, 0x6a, 0x15, 0x82, 0xb3	**EFI_USB_IO_PROTOCOL. UsbIsochronousTransfer** - **UsbIsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbIsochronousTransfer ()** with an invalid *DeviceEndpoint* value of 0x80. The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.5.4	0xc9cc277e, 0x02a3, 0x4392, 0x82, 0x24, 0x87, 0xe5, 0x26, 0x21, 0xfd, 0xd6	**EFI_USB_IO_PROTOCOL. UsbIsochronousTransfer** - **UsbIsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbIsochronousTransfer ()** with an invalid *DeviceEndpoint* value of 0x90. The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.21.2.5.5	0x686e7854, 0xe518, 0x41c1, 0xb1, 0x71, 0x60, 0x4e, 0x6f, 0x7e, 0xe2, 0x91	EFI_USB_IO_PROTOCOL.UsbIsochronousTransfer - UsbIsochronousTransfer () returns EFI_INVALID_PARAMETER with a *DeviceEndpoint* which is not an Isochronous endpoint.	1. Call UsbIsochronousTransfer () with a *DeviceEndpoint* value of not an Isochronous endpoint. The return status should be EFI_INVALID_PARAMETER.

11.2.6 UsbAsyncIsochronousTransfer()

Number	GUID	Assertion	Test Description
5.21.2.6.1	0x5a8a2a48, 0xd6cc, 0x4993, 0x82, 0x1e, 0xf7, 0x2f, 0x48, 0x40, 0xa7, 0x26	EFI_USB_IO_PROTOCOL.UsbAsyncIsochronousTransfer - UsbAsyncIsochronousTransfer () returns EFI_INVALID_PARAMETER with an invalid *DeviceEndpoint*.	1. Call UsbAsyncIsochronousTransfer () with an invalid *DeviceEndpoint* value of 0. The return status should be EFI_INVALID_PARAMETER.
5.21.2.6.2	0x7df33f6b, 0x7525, 0x4999, 0x83, 0x9c, 0xb2, 0xc7, 0x73, 0xd1, 0xa2, 0xa5	EFI_USB_IO_PROTOCOL.UsbAsyncIsochronousTransfer - UsbAsyncIsochronousTransfer () returns EFI_INVALID_PARAMETER with an invalid *DeviceEndpoint*.	1. Call UsbAsyncIsochronousTransfer () with an invalid *DeviceEndpoint* value of 0x10. The return status should be EFI_INVALID_PARAMETER.
5.21.2.6.3	0x586d899f, 0x34f8, 0x474d, 0x99, 0x5e, 0x9e, 0x3e, 0x98, 0x9f, 0xf0, 0xee	EFI_USB_IO_PROTOCOL.UsbAsyncIsochronousTransfer - UsbAsyncIsochronousTransfer () returns EFI_INVALID_PARAMETER with an invalid *DeviceEndpoint*.	1. Call UsbAsyncIsochronousTransfer () with an invalid *DeviceEndpoint* value of 0x80. The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.21.2.6.4	0xfbe98aec, 0xeab8, 0x45a3, 0x85, 0xd3, 0x00, 0x32, 0x0d, 0x1c, 0xaa, 0xe3	**EFI_USB_IO_PROTOCOL.UsbAsyncIsochronousTransfer** - **UsbAsyncIsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with an invalid *DeviceEndpoint*.	1. Call **UsbAsyncIsochronousTransfer ()** with an invalid *DeviceEndpoint* value of **0x90**. The return status should be **EFI_INVALID_PARAMETER**.
5.21.2.6.5	0x7588b124, 0xdaa7, 0x4715, 0xa1, 0x99, 0xa4, 0xdc, 0x32, 0x19, 0x1c, 0xc9	**EFI_USB_IO_PROTOCOL.UsbAsyncIsochronousTransfer** - **UsbAsyncIsochronousTransfer ()** returns **EFI_INVALID_PARAMETER** with a *DeviceEndpoint* value of not an Isochronous endpoint.	1. Call **UsbAsyncIsochronousTransfer ()** with a *DeviceEndpoint* value of not an Isochronous endpoint. The return status should be **EFI_INVALID_PARAMETER**.

11.2.7 UsbGetDeviceDescriptor()

Number	GUID	Assertion	Test Description
5.21.2.7.1	0xe789ba3f, 0x2405, 0x4d45, 0xbf, 0xdb, 0x7e, 0xa7, 0xe8, 0x33, 0xc6, 0x8b	**EFI_USB_IO_PROTOCOL.UsbGetDeviceDescriptor** - **UsbGetDeviceDescriptor ()** returns **EFI_INVALID_PARAMETER** with a *DeviceDescriptor* value of **NULL**.	1. Call **UsbGetDeviceDescriptor ()** with a *DeviceDescriptor* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.

11.2.8 UsbGetConfigDescriptor()

Number	GUID	Assertion	Test Description
5.21.2.8.1	0x387570c3, 0x6923, 0x4cbb, 0x82, 0xb2, 0x59, 0xc7, 0x41, 0xab, 0x92, 0x4b	**EFI_USB_IO_PROTOCOL. UsbGetConfigDescriptor** - **UsbGetConfigDescriptor ()** returns **EFI_INVALID_PARAMETER** with a *ConfigurationDescriptor* value of **NULL**.	1. Call **UsbGetConfigDescriptor ()** with a *ConfigurationDescriptor* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.

11.2.9 UsbGetInterfaceDescriptor()

Number	GUID	Assertion	Test Description
5.21.2.9.1	0x47c33713, 0x8fbc, 0x43a4, 0xa2, 0xcd, 0xc1, 0x6b, 0xc7, 0xa5, 0xd4, 0x37	**EFI_USB_IO_PROTOCOL. UsbGetInterfaceDescriptor** - **UsbGetInterfaceDescriptor ()** returns **EFI_INVALID_PARAMETER** with a *InterfaceDescriptor* value of **NULL**.	1. Call **UsbGetInterfaceDescriptor ()** with a *InterfaceDescriptor* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.

11.2.10 UsbGetEndpointDescriptor()

Number	GUID	Assertion	Test Description
5.21.2.10.1	0x8167f778, 0xa58c, 0x4837, 0xaf, 0xfb, 0x5e, 0x10, 0x69, 0x66, 0xa8, 0x74	**EFI_USB_IO_PROTOCOL. UsbGetEndpointDescriptor** - **UsbGetEndpointDescriptor ()** returns **EFI_INVALID_PARAMETER** with an *EndpointDescriptor* value of **NULL**.	1. Call **UsbGetEndpointDescriptor ()** with an *EndpointDescriptor* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.21.2.10.2	0xb0da5669, 0x163d, 0x4d93, 0xae, 0xf0, 0x7b, 0x28, 0x53, 0x5f, 0x47, 0x3e	EFI_USB_IO_PROTOCOL.UsbGetEndpointDescriptor - UsbGetEndpointDescriptor () returns EFI_INVALID_PARAMETER with an *EndpointIndex* value of larger than 15.	1. Call UsbGetEndpointDescriptor () with an *EndpointIndex* value of larger than 15. The return status should be EFI_INVALID_PARAMETER.
5.21.2.10.3	0x692ec6a6, 0x057d, 0x43c3, 0x94, 0x74, 0x5c, 0x29, 0xb2, 0x5e, 0x5c, 0xe5	EFI_USB_IO_PROTOCOL.UsbGetEndpointDescriptor - UsbGetEndpointDescriptor () returns EFI_INVALID_PARAMETER with an *EndpointIndex* value of equal to the number of endpoints.	1. Call UsbGetInterfaceDescriptor () to get the number of endpoints. The return status should be EFI_SUCCESS. 2. Call UsbGetEndpointDescriptor () with an *EndpointIndex* value of equal to the number of endpoints. The return status should be EFI_INVALID_PARAMETER.

11.2.11 UsbPortReset()

Number	GUID	Assertion	Test Description
5.21.2.11.1	0x27431330, 0x54c8, 0x40fe, 0x93, 0x74, 0x9d, 0x39, 0x4d, 0x10, 0x75, 0x3b	EFI_USB_IO_PROTOCOL.UsbPortReset - UsbPortReset () returns EFI_INVALID_PARAMETER with a USB hub.	1. Call UsbPortReset () with a USB hub. The return status should be EFI_INVALID_PARAMETER.

12 Protocols SCSI Bus Support Test

12.1 EFI_SCSI_IO_PROTOCOL Function Test

Reference Document:

UEFI Specification, EFI_SCSI_IO_PROTOCOL Section..

12.1.1 GetDeviceType() Function

Number	GUID	Assertion	Test Description
5.9.2.1.1	0xa9b53582, 0xcbd5, 0x4934, 0x85, 0x95, 0x2e, 0x4d, 0xc6, 0x8a, 0xb1, 0x34	**EFI_SCSI_IO_PROTOCOL.GetDeviceType** – **GetDeviceType()** should return **EFI_SUCCESS** with SCSI device correctly installed	Call **GetDeviceType()**. The return status should be **EFI_SUCCESS**.

12.1.2 GetDeviceLocation() Function

Number	GUID	Assertion	Test Description
5.9.2.2.1	0x2d1db8e2, 0xb4d3, 0x4bbf, 0x80, 0xa6, 0x4c, 0x15, 0xef, 0x54, 0x87, 0x31	**EFI_SCSI_IO_PROTOCOL.GetDeviceLocation** – **GetDeviceLocation()** should return **EFI_SUCCESS** after setting *Target* and *Lun*.	Call **GetDeviceLocation()** with valid *Target* and *Lun*. The return status should be **EFI_SUCCESS**.

12.1.3 ResetBus() Function

Number	GUID	Assertion	Test Description
5.9.2.3.1	0xb11aec12, 0x0ffb, 0x46da, 0x82, 0x37, 0xaa, 0xa0, 0xed, 0x46, 0x29, 0x05	**EFI_SCSI_IO_PROTOCOL.ResetBus** - **ResetBus()** should return **EFI_SUCCESS** or **EFI_UNSUPPORTED** with SCSI device correctly installed.	Call **ResetBus()** after SCSI device correctly installed. The return status should be **EFI_SUCCESS** or **EFI_UNSUPPORTED**.

12.1.4 ResetDevice() Function

Number	GUID	Assertion	Test Description
5.9.2.4.1	0x05720e96, 0xf8ab, 0x46f5, 0xbc, 0xf9, 0xc9, 0x24, 0x51, 0x1c, 0xd5, 0x44	`EFI_SCSI_IO_PROTOCOL.ResetDevice` - `ResetDevice()` should return `EFI_SUCCESS` or `EFI_UNSUPPORTED` with SCSI device correctly installed.	Call `ResetDevice()` after SCSI device correctly installed. The return status should be `EFI_SUCCESS` or `EFI_UNSUPPORTED`.

12.1.5 ExecuteScsiCommand () Function

Number	GUID	Assertion	Test Description
5.9.2.5.1	0xaf88a458, 0xdeab, 0x4744, 0xae, 0xf5, 0xe4, 0x1c, 0xb1, 0x0e, 0xbb, 0xb3	`EFI_SCSI_IO_PROTOCOL.ExecuteScsiCommand` - Invokes `ExecuteScsiCommand()` with `NULL` *`Event`* will verify interface correctness by returning `EFI_SUCCESS`.	Call `ExecuteScsiCommand ()` with `NULL` *`Event`*. The return status should be `EFI_SUCCESS`.
5.9.2.5.2	0x96789d65, 0x11e6, 0x4a2d, 0xbb, 0x5b, 0xe3, 0x3d, 0x22, 0x6b, 0x28, 0xf1	`EFI_SCSI_IO_PROTOCOL.ExecuteScsiCommand` - Invokes `ExecuteScsiCommand()` with *`Event`* verifies interface correctness.	Call `ExecuteScsiCommand ()` with *`Event`*. The return status should be `EFI_SUCCESS` and the event should be invoked.

12.2 EFI_SCSI_IO_PROTOCOL Conformance Test

Reference Document:

UEFI Specification, EFI_SCSI_IO_PROTOCOL Section.

12.2.1 GetDeviceType() Conformance

Number	GUID	Assertion	Test Description
5.9.3.1.1	0x37a8da14, 0x170a, 0x4620, 0xaa, 0xea, 0x26, 0x6f, 0x35, 0x8f, 0x0c, 0x75	**EFI_SCSI_IO_PROTOCOL.GetDeviceType** – **GetDeviceType()** should return **EFI_INVALID_PARAMETER** with *DeviceType* set **NULL**.	Call **GetDeviceType()** with a *DeviceType* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.

12.2.2 GetDeviceLocation() Conformance

Number	GUID	Assertion	Test Description
5.9.3.2.1	0x6937c784, 0xb044, 0x4828, 0xb8, 0x77, 0xff, 0xc7, 0x35, 0x8f, 0xf2, 0xaa	**EFI_SCSI_IO_PROTOCOL.GetDeviceLocation** – **GetDeviceLocation()** should return **EFI_INVALID_PARAMETER** with *Target* set **NULL**.	Call **GetDeviceLocation ()** with a *Target* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.9.3.2.2	0x6a48edf9, 0x8a3b, 0x4e9c, 0xb7, 0x6f, 0x37, 0x45, 0x83, 0xc7, 0xdc, 0x2b	**EFI_SCSI_IO_PROTOCOL.GetDeviceLocation** – **GetDeviceLocation()** should return **EFI_INVALID_PARAMETER** with *Lun* set **NULL**.	Call **GetDeviceLocation ()** with a *Lun* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.

12.2.3 ExecuteScsiCommand () Conformance

Number	GUID	Assertion	Test Description
5.9.3.3.1	0x17503bd1, 0x4d36, 0x4183, 0x9f, 0xf1, 0x9d, 0x0f, 0xc2, 0x21, 0x33, 0x26	**EFI_SCSI_IO_PROTOCOL. ExecuteScsiCommand** - Calling **ExecuteScsiCommand ()** with an too long *InTransferLength* value and **NULL** *Event* returns **EFI_BAD_BUFFER_SIZE**.	Call **ExecuteScsiCommand ()** with an *InTransferLength* value larger than the length which SCSI controller can handle. The return status should be **EFI_BAD_BUFFER_SIZE** and *InTransferLength* will be updated to the length that SCSI controller be able to handle.
5.9.3.3.2	0x8c27b8c2, 0x2c40, 0x4f6a, 0xbb, 0x54, 0x26, 0x5d, 0x12, 0x9a, 0x97, 0xce	**EFI_SCSI_IO_PROTOCOL. ExecuteScsiCommand** - Calling **ExecuteScsiCommand ()** with invalid *Packet* and **NULL** *Event* returns **EFI_INVALID_PARAMETER**.	Call **ExecuteScsiCommand ()** with invalid *Packet* . The return status should be **EFI_INVALID_PARAMETER**.
5.9.3.3.3	0xbeb81209, 0x808d, 0x46d1, 0xa2, 0x36, 0x23, 0x7f, 0x17, 0x22, 0x30, 0x37	**EFI_SCSI_IO_PROTOCOL. ExecuteScsiCommand** - Calling **ExecuteScsiCommand ()** with an too long *InTransferLength* value and no **NULL** *Event* returns **EFI_BAD_BUFFER_SIZE**.	Call **ExecuteScsiCommand ()** with an *InTransferLength* value larger than the length which SCSI controller can handle. The return status should be **EFI_BAD_BUFFER_SIZE** and *InTransferLength* will be updated to the length that SCSI controller be able to handle.
5.9.3.3.4	0x994fd5e2, 0x2d39, 0x4fa9, 0xa7, 0x4f, 0x8d, 0x09, 0xe0, 0xb6, 0x84, 0x1c	**EFI_SCSI_IO_PROTOCOL. ExecuteScsiCommand** - Calling **ExecuteScsiCommand ()** with invalid *Packet* and no **NULL** *Event* returns **EFI_INVALID_PARAMETER**.	Call **ExecuteScsiCommand ()** with invalid *Packet* . The return status should be **EFI_INVALID_PARAMETER**.

12.3 EFI_EXT_SCSI_PASS_PROTOCOL Function Test

Reference Document:

UEFI Specification, EFI_EXT_SCSI_PASS_THRU_PROTOCOL Section.

12.3.1 GetNextTargetLun() Function

Number	GUID	Assertion	Test Description
5.9.4.1.1	0x4f658292, 0xa409, 0x4d67, 0xba, 0x13, 0x04, 0xc2, 0x51, 0x85, 0xf2, 0x80	`EFI_EXT_SCSI_PASS_THRU_PROTOCOL.GetNextTargetLun` – `GetNextTargetLun()` retrieves the list of legal Target IDs and LUNs for SCSI devices on a SCSI channel.	Call `GetNextTargetLun ()` with a *`Target`* value of 0xFF's to get the first SCSI device present on a SCSI channel. Use the values of *`Target`* and *`Lun`* values that are returned to get the next SCSI device until the end. Every call of `GetNextTargetLun()` should return `EFI_SUCCESS` except the last one. The last call should return `EFI_NOT_FOUND`.

12.3.2 BuildDevicePath() Function

Number	GUID	Assertion	Test Description
5.9.4.2.1	0x130d44b6, 0xce53, 0x42b6, 0x9b, 0xa6, 0x3d, 0x11, 0x5d, 0x49, 0x2b, 0x33	`EFI_EXT_SCSI_PASS_THRU_PROTOCOL.BuildDevicePath` - Invoking `BuildDevicePath()` will verify interface correctness by returning `EFI_SUCCESS`.	Call `GetNextTargetLun()` to get the first device's *`Target`* and *`Lun`*. Call `BuildDevicePath()` with a valid parameter. Free the *`DevicePath`*. The return status should be `EFI_SUCCESS`.

12.3.3 GetTargetLun() Function

Number	GUID	Assertion	Test Description
5.9.4.3.1	0x6ea827e4, 0x522c, 0x44b6, 0x99, 0xe4, 0x25, 0x93, 0x19, 0xba, 0xcc, 0x57	`EFI_EXT_SCSI_PASS_THRU_PROTOCOL.GetTargetLun` - Invoking `GetTargetLun()` will verify interface correctness by returning `EFI_SUCCESS`.	Call `GetNextTargetLun()` and `BuildDevicePath()` to get the valid *`DevicePath`*. Use this *`DevicePath`* to call `GetTargetLun()`. The return value should be `EFI_SUCCESS`.

12.3.4 ResetChannel() Function

Number	GUID	Assertion	Test Description
5.9.4.4.1	0x4e0080d2, 0x4065, 0x4b92, 0xa4, 0x61, 0x52, 0x49, 0xf3, 0x8f, 0xaf, 0x55	EFI_EXT_SCSI_PASS_THRU_PROTOCOL.ResetChannel - Invoking ResetChannel() will verify interface correctness via return code of EFI_SUCCESS or EFI_UNSUPPORTED.	Call ResetChannel(). The return value should be EFI_SUCCESS or EFI_UNSUPPORTED.

12.3.5 ResetTargetLun() Function

Number	GUID	Assertion	Test Description
5.9.4.5.1	0x9400bc81, 0x9e48, 0x469b, 0xa0, 0x97, 0xd0, 0x08, 0x45, 0xb6, 0x69, 0xe8	EFI_EXT_SCSI_PASS_THRU_PROTOCOL.ResetTargetLun - Invoking ResetTargetLun() will verify interface correctness via return code of EFI_SUCCESS or EFI_UNSUPPORTED.	Call GetNextTargetLun() to get valid *Target* and *Lun*. Use the *Target* and *Lun* values that are returned to call ResetTargetLun(). The return value should be EFI_SUCCESS or EFI_UNSUPPORTED.

12.3.6 GetNextTarget () Function

Number	GUID	Assertion	Test Description
5.9.4.6.1	0xc89631f3, 0xbd59, 0x4959, 0xba, 0x10, 0x3f, 0xa9, 0x94, 0x62, 0x02, 0xdf	EFI_EXT_SCSI_PASS_THRU_PROTOCOL. GetNextTarget – GetNextTarget() retrieves the list of legal Target IDs for SCSI devices on a SCSI channel.	Call GetNextTarget () with a *Target* value of 0xFF's to get the first SCSI device present on a SCSI channel. Use the *Target* value that is returned to get the next SCSI device until the end. Every call of GetNextTarget () should return EFI_SUCCESS except the last one. The last call should return EFI_NOT_FOUND.

12.3.7 PassThru () Function

Number	GUID	Assertion	Test Description
5.9.4.7.1	0xdb7841b9, 0x2a4a, 0x45b1, 0xa9, 0x9f, 0x67, 0x7a, 0xb4, 0xcd, 0x79, 0xa2	EFI_EXT_SCSI_PASS_THRU_PROTOCOL.PassThru - Invoking PassThru() with NULL *Event* will verify interface correctness by returning EFI_SUCCESS.	Call GetNextDevice() to get valid *Target* and *Lun* values. Call PassThru() with the returned values of *Target*, *Lun* , and a NULL *Event*. The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.9.4.7.2	0x4787ed6f, 0xa984, 0x4b15, 0xb2, 0xf3, 0xa0, 0xd1, 0xb8, 0xce, 0x61, 0x89	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.PassThru** - Invoking **PassThru()** with *Event* will verify interface correctness by returning **EFI_SUCCESS**.	Call **GetNextDevice()** to get valid *Target* and *Lun* values. Call **PassThru()** with the returned value of *Target*, *Lun* and a **Event**. The return status should be **EFI_SUCCESS** and the event should be invoked.

12.4 EFI_EXT_SCSI_PASS_PROTOCOL Conformance Test

Reference Document:

UEFI Specification, EFI_EXT_SCSI_PASS_THRU_PROTOCOL Section.

12.4.1 GetNextTargetLun() Conformance

Number	GUID	Assertion	Test Description
5.9.5.1.1	0xaad50e59, 0x9423, 0x427d, 0xa7, 0x5d, 0x69, 0x1c, 0x90, 0xb7, 0xf9, 0x75	**EFI_SCSI_PASS_THRU_PROTOCOL.GetNextTargetLun** - Call **GetNextTargetLun()** with an invalid *Target*.	Call **GetNextTargetLun()** with *Target*'s all bits are 1 to get the first device. Call **GetNextTargetLun()** with an invalid *Target*. It should return **EFI_INVALID_PARAMETER**.
5.9.5.1.2	0xb3e87aa1, 0x6e9c, 0x478f, 0x9b, 0xd5, 0x39, 0x50, 0x08, 0x01, 0x28, 0x96	**EFI_SCSI_PASS_THRU_PROTOCOL.GetNextTargetLun** - Call **GetNextTargetLun()** with an invalid *Lun*.	Call **GetNextTargetLun()** with *Target*'s all bits are 1 to get the first device. Call **GetNextTargetLun()** with an invalid *Lun*. It should return **EFI_INVALID_PARAMETER**.

12.4.2 BuildDevicePath() Conformance

Number	GUID	Assertion	Test Description
5.9.5.2.1	0x942a0e01, 0x7b80, 0x46e4, 0xa7, 0x57, 0x86, 0xc4, 0xec, 0x53, 0xf4, 0xe4	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.BuildDevicePath** - Calling **BuildDevicePath()** with an invalid *Target* returns **EFI_NOT_FOUND**.	Call **BuildDevicePath()** with an invalid *Target*. The return status should be **EFI_NOT_FOUND**.
5.9.5.2.2	0x222f00c1, 0xf6bf, 0x41ed, 0xae, 0xfd, 0xaa, 0xc4, 0x8f, 0x3f, 0xa9, 0xdb	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.BuildDevicePath** - Calling **BuildDevicePath()** with invalid *Lun* returns **EFI_NOT_FOUND**.	Call **BuildDevicePath()** with invalid *Lun*. The return status should be **EFI_NOT_FOUND**.
5.9.5.2.3	0xc72e6a78, 0x5292, 0x4493, 0x90, 0x40, 0xb0, 0x44, 0x5a, 0x9c, 0x17, 0x14	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.BuildDevicePath** - Calling **BuildDevicePath()** with **NULL** *DevicePath* returns **EFI_INVALID_PARAMETER**.	Call **BuildDevicePath()** with **NULL** *DevicePath*. The return status should be **EFI_INVALID_PARAMETER**.

12.4.3 GetTargetLun() Conformance

Number	GUID	Assertion	Test Description
5.9.5.3.1	0xff2f0849, 0x690b, 0x48ea, 0x8e, 0x35, 0x64, 0x36, 0x3f, 0xaa, 0x8c, 0x5c	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.GetTargetLun** – Invoking **GetTargetLun()** with **NULL** *DevicePath* returns **EFI_INVALID_PARAMETER**.	Call **GetTargetLun()** with **NULL** *DevicePath*. The return status should be **EFI_INVALID_PARAMETER**.
5.9.5.3.2	0x6602bd0a, 0x1c05, 0x49e5, 0xa8, 0xd4, 0xc6, 0x03, 0x8c, 0x43, 0x9a, 0xf9	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.GetTargetLun** – Invoking **GetTargetLun()** with **NULL** *Target* returns **EFI_INVALID_PARAMETER**.	Call **GetTargetLun()** with **NULL** *Target*. The return status should be **EFI_INVALID_PARAMETER**.

5.9.5.3.3	0x1b64d49a, 0x1f1b, 0x4610, 0xa2, 0x66, 0xde, 0x32, 0xa1, 0x07, 0x2b, 0x32	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.GetTargetLun** – Invoking **GetTargetLun()** with **NULL** *Lun* returns **EFI_INVALID_PARAMETER**.	Call **GetTargetLun()** with **NULL** *Lun*. The return status should be **EFI_INVALID_PARAMETER**.
5.9.5.3.4	0xf7830eaf, 0xba30, 0x4224, 0xab, 0xc4, 0x42, 0x42, 0x8b, 0x7a, 0x04, 0x5d	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.GetTargetLun** - Calling **GetTargetLun()** with unsupported *DevicePath* returns **EFI_UNSUPPORTED**.	Call **GetTargetLun()** with unsupported *DevicePath*. The return status should be **EFI_UNSUPPORTED**.

12.4.4 ResetTargetLun() Conformance

Number	GUID	Assertion	Test Description
5.9.5.4.1	0x106ae2fc, 0x3f34, 0x4afe, 0x82, 0x44, 0x40, 0x27, 0x57, 0x60, 0x98, 0x31	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.ResetTargetLun** - Calling **ResetTargetLun()** with an invalid *Target* returns **EFI_INVALID_PARAMETER**.	Call **GetResetTargetLun()** with an invalid *Target*. The return status should be **EFI_INVALID_PARAMETER**.
5.9.5.4.2	0xc9378047, 0x9b4b, 0x4abf, 0xaa, 0x6b, 0xe3, 0xcd, 0xb6, 0xc4, 0x19, 0x39	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.ResetTargetLun** - Calling **ResetTargetLun()** with an invalid *Lun* returns **EFI_INVALID_PARAMETER**.	Call **GetResetTargetLun()** with an invalid *Lun*. The return status should be **EFI_INVALID_PARAMETER**.

12.4.5 GetNextTarget () Conformance

Number	GUID	Assertion	Test Description
5.9.5.5.1	0xb564ad60, 0x32ce, 0x4f5f, 0x86, 0x7a, 0xef, 0x9f, 0xef, 0x5e, 0x94, 0xa2	**EFI_SCSI_PASS_THRU_PROTOCOL.GetNextTarget** - Call **GetNextTarget()** with an invalid *Target*	Call **GetNextTarget()** with an invalid *Target*. The return status should be **EFI_INVALID_PARAMETER**.

12.4.6 PassThru() Conformance

Number	GUID	Assertion	Test Description
5.9.5.6.1	0x6d6fcacd, 0x3463, 0x41c8, 0xa5, 0x01, 0xa2, 0x99, 0x40, 0x44, 0x59, 0xb8	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.PassThru** – Calling **PassThru()** with an too long *InTransferLength* and **NULL** *Event* returns **EFI_BAD_BUFFER_SIZE**.	Call **PassThru()** with an *InTransferLength* larger than the SCSI controller can handle. The return status should be **EFI_BAD_BUFFER_SIZE** and the *InTransferLength* will be updated to the length that SCSI controller can handle.
5.9.5.6.2	0x645295b5, 0xc36b, 0x4b23, 0xaf, 0xc7, 0xd4, 0xcc, 0xc0, 0x1d, 0xb6, 0x4f	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.PassThru** – Calling **PassThru()** with an invalid *Target* and **NULL** *Event* returns **EFI_INVALID_PARAMETER**.	Call **PassThru()** with an invalid *Target*. The return status should be **EFI_INVALID_PARAMETER**.
5.9.5.6.3	0x9f9489a2, 0x23f3, 0x4962, 0x9d, 0x8f, 0xd2, 0xc0, 0xa7, 0xcb, 0x2f, 0xb1	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.PassThru – Calling PassThru()** with an invalid *Lun* and **NULL** *Event* returns **EFI_INVALID_PARAMETER**.	Call **PassThru()** with an invalid *Lun*. The return status should be **EFI_INVALID_PARAMETER**.
5.9.5.6.4	0xc584b074, 0xa8cd, 0x438c, 0xb5, 0x18, 0xb1, 0xec, 0x59, 0xfa, 0xc8, 0xee	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.PassThru** – Calling **PassThru()** with invalid *Packet* content and **NULL** *Event* returns **EFI_INVALID_PARAMETER**.	Call **PassThru()** with invalid *Packet* content. The return status should be **EFI_INVALID_PARAMETER**.
5.9.5.6.5	0x3cd806fd, 0x3742, 0x44e9, 0xa6, 0x19, 0xdf, 0x2d, 0x37, 0x47, 0xe7, 0x8f	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.PassThru** – Calling **PassThru()** with an too long *InTransferLength* and no **NULL** *Event* returns **EFI_BAD_BUFFER_SIZE**.	Call **PassThru()** with an *InTransferLength* larger than the SCSI controller can handle. The return status should be **EFI_BAD_BUFFER_SIZE** and the *InTransferLength* will be updated to the length that SCSI controller can handle.

Number	GUID	Assertion	Test Description
5.9.5.6.6	0x9648ab45, 0x898b, 0x4b44, 0xab, 0x9e, 0x24, 0x6b, 0xc6, 0x49, 0xc9, 0xfd	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.PassThru** – Calling **PassThru()** with an invalid *Target* and no **NULL** *Event* returns **EFI_INVALID_PARAMETER**	Call **PassThru()** with an invalid *Target*. The return status should be **EFI_INVALID_PARAMETER**.
5.9.5.6.7	0x8662da7d, 0x6f98, 0x4051, 0xb1, 0x87, 0x85, 0xb0, 0xf4, 0xb5, 0x3a, 0xf1	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.PassThru – Calling PassThru()** with an invalid *Lun* and no **NULL** *Event* returns **EFI_INVALID_PARAMETER**.	Call **PassThru()** with an invalid *Lun*. The return status should be **EFI_INVALID_PARAMETER**.
5.9.5.6.8	0xf9ec9bf2, 0x743f, 0x4eed, 0x82, 0xbc, 0x35, 0xf2, 0xcc, 0x56, 0x45, 0xda	**EFI_EXT_SCSI_PASS_THRU_PROTOCOL.PassThru** – Calling **PassThru()** with invalid *Packet* content and no **NULL** *Event* returns **EFI_INVALID_PARAMETER**.	Call **PassThru()** with invalid *Packet* content. The return status should be **EFI_INVALID_PARAMETER**.

13 Protocols iSCSI Boot Test

EFI_ISCSI_INITIATOR_NAME_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_ISCSI_INITIATOR_NAME_PROTOCOL Section.

13.1 EFI_ISCSI_INITIATOR_NAME_PROTOCOL Function Test

13.1.1 Get() Function

Number	GUID	Assertion	Test Description
5.17.1.1.1	0xed92f3eb, 0xdda4, 0x4c65, 0xb3, 0x9f, 0x6c, 0x90, 0xfb, 0x2e, 0x77, 0xf9	**EFI_ISCSI_INITIATOR_NAME_PROTOCOL.Get** – Calling **Get()** returns **EFI_SUCCESS.**	Call **Get()** with a valid *BufferSize* value. The return status should be **EFI_SUCCESS**.

13.1.2 Set() Function

Number	GUID	Assertion	Test Description
5.17.1.2.1	0x56cd69be, 0xcfea, 0x4a43, 0xae, 0x1a, 0x41, 0xe4, 0xde, 0x78, 0x83, 0xc8	**EFI_ISCSI_INITIATOR_NAME_PROTOCOL.Set** – Calling **Set()** returns **EFI_SUCCESS.**	Call **Set()** with valid *BufferSize* and *Buffer* values. The return status should be **EFI_SUCCESS**.

13.2 EFI_ISCSI_INITIATOR_NAME_PROTOCOL Conformance Test

13.2.1 Get() Conformance

Number	GUID	Assertion	Test Description
5.17.2.1.1	0x4c893a1c, 0x9c28, 0x4038, 0x9a, 0x34, 0xce, 0xe3, 0x15, 0x70, 0xc4, 0xa6	**EFI_ISCSI_INITIATOR_NAME_PROTOCOL.Get** – Calling **Get()** should return **EFI_SUCCESS** with valid parameters.	Call **Get()** with valid parameters. The return status should be **EFI_SUCCESS**.
5.17.2.1.2	0x5f4d6864, 0xe8ed, 0x452e, 0xb2, 0xbc, 0x9a, 0x0e, 0x06, 0x61, 0x7e, 0x3a	**EFI_ISCSI_INITIATOR_NAME_PROTOCOL.Get** – Calling **Get()** should return **EFI_INVALID_PARAMETER** with a *BufferSize* or *Buffer* value of **NULL**.	1. Call **Get()** with a *BufferSize* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER.** 2. Call **Get()** with a *Buffer* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER.**
5.17.2.1.3	0x2502087d , 0xd853, 0x494e, 0xbd, 0xc5, 0x8b, 0x1a, 0xc1, 0x26, 0xd4, 0x61	**EFI_ISCSI_INITIATOR_NAME_PROTOCOL.Get** – Calling **Get()** should return **EFI_INVALID_PARAMETER** with a *BufferSize* value that is too small.	Call **Get()** with a *BufferSize* that is too small. The return status should be **EFI_INVALID_PARAMETER.**

13.2.2 Set() Conformance

Number	GUID	Assertion	Test Description
5.17.2.2.1	0x5bd1c13e, 0x1b9c, 0x432f, 0xb9, 0x33, 0xd9, 0xcf, 0x6f, 0xac, 0xd4, 0x2d	**EFI_ISCSI_INITIATOR_NAME_PROTOCOL.Set** – Calling **Set()** should return **EFI_SUCCESS** with valid parameters.	Call **Set()** with valid parameters. The return status should be **EFI_SUCCESS**.
5.17.2.2.2	0xacb61cfd, 0xe82b, 0x4250, 0xb0, 0x60, 0xdb, 0x18, 0x55, 0x9e, 0x58, 0xb1	**EFI_ISCSI_INITIATOR_NAME_PROTOCOL.Set** – Calling **Set()** should return **EFI_INVALID_PARAMETER** with a *BufferSize* or *Buffer* value of **NULL**.	1. Call **Set()** with a *BufferSize* value of **NULL**. The return should be **EFI_INVALID_PARAMETER.** 2. Call **Set()** with a *Buffer* value of **NULL**. The return should be **EFI_INVALID_PARAMETER.**

Number	GUID	Assertion	Test Description
5.17.2.2.3	0xdc419b8e, 0xb074, 0x4388, 0xbb, 0x85, 0xc8, 0xed, 0xa0, 0x19, 0x95, 0xd3	**EFI_ISCSI_INITIATOR_NAME_PROTOCOL.Get** – Calling **Get()** should return **EFI_INVALID_PARAMETER** with a *BufferSize* value that exceeds the maximum.	Call **Get()** with a *BufferSize* value that exceeds the maximum. The return should be **EFI_INVALID_PARAMETER.**

14 Network Protocols SNP, PXE and BISTest

14.1 EFI_SIMPLE_NETWORK_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_SIMPLE_NETWORK_PROTOCOL Section..

14.1.1 Start()

Number	GUID	Assertion	Test Description
5.11.1.1.1	0x200d5d39, 0x8131, 0x434f, 0x95, 0x89, 0xc6, 0xbe, 0x88, 0x69, 0x5d, 0xf4	**EFI_SIMPLE_NETWORK_PROTOCOL.Start** - returns **EFI_ALREADY_STARTED** when calling **Start()** while the network interface is already started	Call **Start()** when the network interface is already started. The return status should be **EFI_ALREADY_STARTED** and the state should be "Started".
5.11.1.1.2	0xf58651fe, 0x0538, 0x4407, 0x88, 0xe0, 0x88, 0xb8, 0xda, 0x18, 0x38, 0x3a	**EFI_SIMPLE_NETWORK_PROTOCOL.Start** - returns **EFI_SUCCESS** when calling **Start()** to verify the interface state.	Call **Start()** The return status should be **EFI_SUCCESS** and the interface state should be *EfiSimpleNetworkStarted*.

14.1.2 Stop()

Number	GUID	Assertion	Test Description
5.11.1.2.1	0xda5a5aea, 0x0a26, 0x4b65, 0x90, 0x84, 0x92, 0x15, 0xc5, 0x43, 0x21, 0xa0	**EFI_SIMPLE_NETWORK_PROTOCOL.Stop** - Invokes **Stop()** when the network interface is not started returns **EFI_NOT_STARTED**.	Call **Stop()** when the network interface is not started. The return status should be **EFI_NOT_STARTED** and the state should be "Stopped".
5.11.1.2.2	0xd0ecac27, 0xfa2e, 0x4b7d, 0x89, 0x2c, 0xc0, 0xff, 0x70, 0x54, 0x13, 0x44	**EFI_SIMPLE_NETWORK_PROTOCOL.Stop** - Invokes **Stop()** verifies the interface state and returns **EFI_SUCCESS**.	Call **Stop()**. The return status should be **EFI_SUCCESS** and the interface state should be *EfiSimpleNetworkStopped*.

14.1.3 Initialize()

Number	GUID	Assertion	Test Description
5.11.1.3.1	0xbaa11393 , 0x2bfc, 0x43ef, 0xbd, 0xb7, 0x0a, 0xc5, 0x0e, 0x8a, 0x3a, 0x21	**EFI_SIMPLE_NETWORK_PROTOCOL.Initialize** - Invokes **Initialize()** when the network interface is not started returns **EFI_NOT_STARTED**.	Call **Initialize()** when the network interface is not started. The return status should be **EFI_NOT_STARTED** and the state should be “Stopped”.
5.11.1.3.2	0x9d4eec8d, 0xdf2f, 0x4f5e, 0x9f, 0x95, 0x7e, 0x51, 0x62, 0xc2, 0x51, 0x0d	**EFI_SIMPLE_NETWORK_PROTOCOL.Initialize** - Invokes **Initialize()** to verify the interface state and returns **EFI_SUCCESS**.	Call **Initialize()**. The return status should be **EFI_SUCCESS** and the interface state should be *EfiSimpleNetworkInitialized*.
5.11.1.3.3	0x7b547661 , 0xa0aa, 0x4041, 0x99, 0xf6, 0xe2, 0x07, 0x31, 0xf7, 0x98, 0x3c	**EFI_SIMPLE_NETWORK_PROTOCOL.Initialize** - Invokes **Initialize()** with extra Tx/Rx specified to verify the interface state and returns **EFI_SUCCESS**.	Call **Initialize()** with extra Tx/Rx specified. The return status should be **EFI_SUCCESS** and the interface state should be *EfiSimpleNetworkInitialized*.

14.1.4 Reset()

Number	GUID	Assertion	Test Description
5.11.1.4.1	0xf2fed213, 0xb6ad, 0x4edc, 0x96, 0xd7, 0x4a, 0xdc, 0x2e, 0xbd, 0xbb, 0x1e	**EFI_SIMPLE_NETWORK_PROTOCOL.Reset** - Invokes **Reset()** when the network interface is not started returns **EFI_NOT_STARTED**.	Call **Reset()** when the network interface is not started. The return status should be **EFI_NOT_STARTED** and the state should be “Stopped”.
5.11.1.4.2	0x30314e89 , 0xdb26, 0x4b01, 0x90, 0xf3, 0x04, 0xd3, 0x1b, 0x19, 0xa6, 0x01	**EFI_SIMPLE_NETWORK_PROTOCOL.Reset** - Invokes **Reset()** with an *ExtendedVerification* value of **FALSE** verifies interface correctness and returns **EFI_SUCCESS**.	Call **Reset()** with an *ExtendedVerification* value of **FALSE**. The return status should be **EFI_SUCCESS** and the interface mode should be correct.

Number	GUID	Assertion	Test Description
5.11.1.4.3	0xa3135b96, 0xf9c6, 0x45b6, 0xae, 0x87, 0x15, 0xca, 0xae, 0x31, 0x7e, 0xfb	**EFI_SIMPLE_NETWORK_PROTOCOL.Reset** - Invokes **Reset()** with an *ExtendedVerification* value of **TRUE** verifies interface correctness and returns **EFI_SUCCESS**.	Call **Reset()** with an *ExtendedVerification* value of **TRUE**. The return status should be **EFI_SUCCESS** and the interface mode should be correct.

14.1.5 Shutdown()

Number	GUID	Assertion	Test Description
5.11.1.5.1	0x09bb5019, 0x1787, 0x4403, 0xb1, 0x2e, 0x91, 0x93, 0x5c, 0xbd, 0x08, 0xe3	**EFI_SIMPLE_NETWORK_PROTOCOL.Shutdown** - Invokes **Shutdown()** when the network interface is not started returns **EFI_NOT_STARTED**.	Call **Shutdown()** when the network interface is not started. The return status should be **EFI_NOT_STARTED** and the state should be “Stopped”.
5.11.1.5.2	0x49365eeb, 0xd66c, 0x4109, 0xb0, 0xcf, 0x36, 0xc8, 0x96, 0xc0, 0x07, 0xec	**EFI_SIMPLE_NETWORK_PROTOCOL.Shutdown** - Invokes **Shutdown()** verifies the interface state and returns **EFI_SUCCESS**.	Call **Shutdown()**. The return status should be **EFI_SUCCESS** and the interface state should be *EfiSimpleNetworkStarted*.

14.1.6 ReceiveFilters()

Number	GUID	Assertion	Test Description
5.11.1.6.1	0x3f8d8e2a, 0xdbb1, 0x41b8, 0xb9, 0xd9, 0x5f, 0x79, 0x44, 0xf1, 0xd1, 0xf4	**EFI_SIMPLE_NETWORK_PROTOCOL.ReceiveFilters** - Invokes **ReceiveFilters()** when the network interface is not started returns **EFI_NOT_STARTED**.	Call **ReceiveFilters()** when the network interface is not started. The return status should be **EFI_NOT_STARTED** and the state should be “Stopped”.

Number	GUID	Assertion	Test Description
5.11.1.6.2	0x8b4ed1bb, 0xa4a4, 0x45e8, 0xbf, 0x32, 0x0d, 0x0d, 0x6d, 0x0b, 0xd0, 0x2e	**EFI_SIMPLE_NETWORK_PROTOCOL.ReceiveFilters** - Invokes **ReceiveFilters()** when the network interface is not initialized returns **EFI_DEVICE_ERROR**.	Call **ReceiveFilters()** when the network interface is not initialized. The return status should be **EFI_DEVICE_ERROR**.
5.11.1.6.3	0xb6f84e0b, 0x286b, 0x44a6, 0xa0, 0xf8, 0x6d, 0x11, 0x89, 0x7d, 0x56, 0x55	**EFI_SIMPLE_NETWORK_PROTOCOL.ReceiveFilters** - Invokes **ReceiveFilters()** with an invalid *Enable* returns **EFI_INVALID_PARAMETER**.	Call **ReceiveFilters()** with an invalid *Enable*. The return status should be **EFI_INVALID_PARAMETER**.
5.11.1.6.4	0xead4b950, 0xf0d6, 0x4195, 0x94, 0xaa, 0x81, 0x92, 0x56, 0x44, 0xb3, 0x2c	**EFI_SIMPLE_NETWORK_PROTOCOL.ReceiveFilters** - Invokes **ReceiveFilters()** with an invalid *McastFilterCnt* returns **EFI_INVALID_PARAMETER**.	Call **ReceiveFilters()** with an invalid *MCastFilterCnt*. The return status should be **EFI_INVALID_PARAMETER**.
5.11.1.6.5	0x4497e853, 0xc54d, 0x409b, 0x85, 0x01, 0xd5, 0xfb, 0xd2, 0x7a, 0x95, 0xdc	**EFI_SIMPLE_NETWORK_PROTOCOL.ReceiveFilters** - Invokes **ReceiveFilters()** with *MCastFilterCnt* not matching *MCastFilter* returns **EFI_INVALID_PARAMETER**.	Call **ReceiveFilters()** with *MCastFilterCnt* not matching *MCastFilter*. The return status should be **EFI_INVALID_PARAMETER**.
5.11.1.6.6	0xd82baa78, 0x2bf8, 0x49db, 0xb5, 0x7f, 0x92, 0x2e, 0xe5, 0x79, 0xc3, 0x7a	**EFI_SIMPLE_NETWORK_PROTOCOL.ReceiveFilters** - Invokes **ReceiveFilters()** modifies the multicast receive filter mask (Disable Specified bit), verifies interface correctness, and returns **EFI_SUCCESS**.	Call **ReceiveFilters()** to modify the multicast receive filter mask (Disable Specified bit) and verify interface correctness. The return status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.11.1.6.7	0x9605c24a, 0x2090, 0x490d, 0x89, 0x4f, 0xfc, 0xb8, 0xc1, 0xb9, 0xd4, 0xf8	EFI_SIMPLE_NETWORK_PROTOCOL.ReceiveFilters - Invokes ReceiveFilters() modifies the multicast receive filter mask (Enable Specified bit), verifies interface correctness, and returns EFI_SUCCESS.	Call ReceiveFilters() to modify the multicast receive filter mask (Enable Specified bit) and verify interface correctness. The return status should be EFI_SUCCESS.
5.11.1.6.8	0xd9893cd3, 0x7269, 0x4931, 0x9e, 0xe8, 0x81, 0x62, 0x7a, 0x67, 0x45, 0xe9	EFI_SIMPLE_NETWORK_PROTOCOL.ReceiveFilters - Invokes ReceiveFilters() modifies the multicast receive filter masks (Enable and Disable Specified bit together), verifies interface correctness, and returns EFI_SUCCESS.	Call ReceiveFilters() to modify the multicast receive filter masks (Enable and Disable Specified bit together) and verify interface correctness. The return status should be EFI_SUCCESS.
5.11.1.6.9	0x056e2680, 0xbcc9, 0x460a, 0x94, 0xb4, 0x9a, 0xe2, 0x99, 0xa7, 0x2c, 0x2c	EFI_SIMPLE_NETWORK_PROTOCOL.ReceiveFilters - Invokes ReceiveFilters() modifies the multicast receive filters list, verifies interface correctness, and returns EFI_SUCCESS.	Call ReceiveFilters() to modify the multicast receive filters list and verify interface correctness. The return status should be EFI_SUCCESS.
5.11.1.6.10	0x2143092e, 0x03dd, 0x4806, 0x9f, 0xd6, 0x08, 0xd4, 0x2b, 0x9a, 0xbf, 0xc6	EFI_SIMPLE_NETWORK_PROTOCOL.ReceiveFilters - Invokes ReceiveFilters() resets the multicast receive filters list, verifies interface correctness within test case, and returns EFI_SUCCESS.	Call ReceiveFilters() to reset the multicast receive filters list and verify interface correctness within test case. The return status should be EFI_SUCCESS.

14.1.7 StationAddress()

Number	GUID	Assertion	Test Description
5.11.1.7.1	0x4235215c, 0xfad0, 0x4865, 0xa9, 0x7b, 0xde, 0xe4, 0xb7, 0xee, 0xef, 0x98	**EFI_SIMPLE_NETWORK_PROTOCOL.***StationAddress* - Invokes *StationAddress***()** when the network interface is not started returns **EFI_NOT_STARTED**.	Call *StationAddress***()** when the network interface is not started. The return status should be **EFI_NOT_STARTED** and the state should be “Stopped”.
5.11.1.7.2	0x9dfe127c, 0x14b0, 0x476d, 0x9d, 0x68, 0x69, 0x08, 0x15, 0x7e, 0x36, 0xa7	**EFI_SIMPLE_NETWORK_PROTOCOL.***StationAddress* - Invokes *StationAddress***()** when the network interface is not initialized returns **EFI_DEVICE_ERROR**.	Call *StationAddress***()** when the network interface is not initialized. The return status should be **EFI_DEVICE_ERROR**.
5.11.1.7.3	0x6c6fb7ad, 0xf89c, 0x45d6, 0xb3, 0xa6, 0x15, 0x34, 0xfd, 0x72, 0xfb, 0x9d	**EFI_SIMPLE_NETWORK_PROTOCOL.***StationAddress* - Invokes *StationAddress***()** with an invalid parameter returns **EFI_INVALID_PARAMETER**.	Call *StationAddress***()** to change the MAC address when the address is not allowed to be changed. The return status should be **EFI_INVALID_PARAMETER**.
5.11.1.7.4	0x29177bfa, 0x3775, 0x4d5a, 0x97, 0x37, 0x19, 0xd8, 0x34, 0xa7, 0xbb, 0x8e	**EFI_SIMPLE_NETWORK_PROTOCOL.***StationAddress* - Invokes *StationAddress***()** resets MAC Address, verifies interface correctness, and returns **EFI_SUCCESS**.	Call *StationAddress***()** to reset MAC Address and verify interface correctness. The return status should be **EFI_SUCCESS**.
5.11.1.7.5	0xbbbde63c, 0xa6f5, 0x4438, 0x8a, 0x82, 0xb4, 0xdf, 0xe8, 0xe8, 0x48, 0xfd	**EFI_SIMPLE_NETWORK_PROTOCOL.***StationAddress* - Invokes *StationAddress***()** modifies MAC Address, verifies interface correctness, and returns **EFI_SUCCESS**.	Call *StationAddress***()** to modify MAC Address and verify interface correctness. The return status should be **EFI_SUCCESS**.

14.1.8 Statistics()

Number	GUID	Assertion	Test Description
5.11.1.8.1	0x62a700f1, 0x075f, 0x4cc0, 0x85, 0x12, 0xee, 0x48, 0x0d, 0xbc, 0x69, 0x2c	EFI_SIMPLE_NETWORK_PROTOCOL.Statistics - Invokes Statistics() when the network interface is not started returns EFI_NOT_STARTED.	Call Statistics() when the network interface is not started. The return status should be EFI_NOT_STARTED and the state should be “Stopped”.
5.11.1.8.2	0x71173afd, 0x5dc9, 0x42ea, 0xa8, 0xad, 0x6e, 0xc0, 0x97, 0x7a, 0xdc, 0xa6	EFI_SIMPLE_NETWORK_PROTOCOL.Statistics - Invokes Statistics() when the network interface is not initialized returns EFI_DEVICE_ERROR.	Call Statistics() when the network interface is not initialized. The return status should be EFI_DEVICE_ERROR.
5.11.1.8.3	0x743b75d1, 0xaf66, 0x495c, 0xaf, 0x5a, 0x1d, 0xdf, 0x7f, 0xe4, 0xa6, 0x82	EFI_SIMPLE_NETWORK_PROTOCOL.Statistics - Invokes Statistics() with small buffer returns EFI_BUFFER_TOO_SMALL or EFI_UNSUPPORTED.	Call Statistics() with small buffer. The return status should be EFI_BUFFER_TOO_SMALL or EFI_UNSUPPORTED.
5.11.1.8.4	0xace9fa20, 0xff34, 0x4fba, 0x8b, 0x95, 0x39, 0xae, 0xca, 0xd9, 0x78, 0x7c	EFI_SIMPLE_NETWORK_PROTOCOL.Statistics - Invokes Statistics() without resetting the statistics and verifying interface correctness returns EFI_SUCCESSor EFI_UNSUPPORTED.	Call Statistics() without resetting the statistics and verifying interface correctness. The return status should be EFI_SUCCESSor EFI_UNSUPPORTED.
5.11.1.8.5	0x3de76704, 0x4bf5, 0x42cd, 0x8c, 0x89, 0x54, 0x7e, 0x4f, 0xad, 0x4f, 0x24	EFI_SIMPLE_NETWORK_PROTOCOL.Statistics - Invokes Statistics(), resetting the statistics, and verifying interface correctness returns EFI_SUCCESSor EFI_UNSUPPORTED.	Call Statistics() and reset the statistics and verify interface correctness. The return status should be EFI_SUCCESSor EFI_UNSUPPORTED.

14.1.9 MCastIPtoMAC()

Number	GUID	Assertion	Test Description
5.11.1.9.1	0x6880bd92, 0x7004, 0x41b8, 0x9e, 0x43, 0x7b, 0x27, 0x1f, 0xd9, 0xac, 0x2b	**EFI_SIMPLE_NETWORK_PROTOCOL.MCastIPtoMAC** - Invokes **MCastIPtoMAC()** when the network interface is not started returns **EFI_NOT_STARTED**.	Call **MCastIPtoMAC()** when the network interface is not started. The return status should be **EFI_NOT_STARTED** and the state should be "Stopped".
5.11.1.9.2	0x544b08c0, 0x1d26, 0x4462, 0x92, 0x07, 0xdd, 0x7e, 0xb7, 0x54, 0xdc, 0x9e	**EFI_SIMPLE_NETWORK_PROTOCOL.MCastIPtoMAC** - Invokes **MCastIPtoMAC()** verifies interface correctness and returns **EFI_SUCCESS**.	Call **MCastIPtoMAC()** and verify interface correctness. The return status should be **EFI_SUCCESS**.

14.1.10 NvData()

Number	GUID	Assertion	Test Description
5.11.1.10.1	0x1a0250a2, 0xd085, 0x42ac, 0xb7, 0x42, 0x52, 0x35, 0x26, 0xa1, 0xa9, 0x4f	**EFI_SIMPLE_NETWORK_PROTOCOL.NvData** - Invokes **NvData()** when the network interface is not started returns **EFI_NOT_STARTED**.	Call **NvData()** when the network interface is not started. The return status should be **EFI_NOT_STARTED** and the state should be "Stopped".
5.11.1.10.2	0xd2aaff2b, 0x6632, 0x4d23, 0x98, 0xca, 0x78, 0xd9, 0x0d, 0xea, 0xfb, 0x2f	**EFI_SIMPLE_NETWORK_PROTOCOL.NvData** - Invokes **NvData()** with *Offset* not a multiple of *NvRamAccessSize* returns **EFI_INVALID_PARAMETER**.	Call **NvData()** with *Offset* not a multiple of *NvRamAccessSize*. The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.11.1.10.3	0xfd0a8da6, 0xe94b, 0x45f0, 0x93, 0x92, 0xe4, 0x8f, 0x9d, 0x09, 0x92, 0xc7	**EFI_SIMPLE_NETWORK_PROTOCOL.NvData** - Invokes **NvData()** with *BufferSize* not a multiple of *NvRamAccessSize* returns **EFI_INVALID_PARAMETER**.	Call **NvData()** with *BufferSize* not a multiple of *NvRamAccessSize*. The return status should be **EFI_INVALID_PARAMETER**.
5.11.1.10.4	0x75fc17ba, 0x5329, 0x4931, 0x96, 0x93, 0xc7, 0x83, 0xf6, 0xac, 0x59, 0xc4	**EFI_SIMPLE_NETWORK_PROTOCOL.NvData** - Invokes **NvData()** with *BufferSize* + *Offset* exceeding *NvRamSize* returns **EFI_INVALID_PARAMETER**.	Call **NvData()** with *BufferSize* + *Offset* exceeding *NvRamSize*. The return status should be **EFI_INVALID_PARAMETER**.
5.11.1.10.5	0xba0b2393, 0x0078, 0x434b, 0x99, 0x13, 0xde, 0xa6, 0x6b, 0xdd, 0x83, 0xb3	**EFI_SIMPLE_NETWORK_PROTOCOL.NvData** - Invokes **NvData()** to read (0, n**NvRamAccessSize*) returns **EFI_SUCCESS**.	Call **NvData()** to read (0, n**NvRamAccessSize*) and verify interface correctness. The return status should be **EFI_SUCCESS**.
5.11.1.10.6	0xf9e2f307, 0x3f73, 0x4c00, 0xbc, 0x31, 0xd5, 0x88, 0xf2, 0x6f, 0x5e, 0xd6	**EFI_SIMPLE_NETWORK_PROTOCOL.NvData** - Invokes **NvData()** to read (*NvRamAccessSize*, (n-1)**NvRamAccessSize*) returns **EFI_SUCCESS**.	Call **NvData()** to read (*NvRamAccessSize*, (n-1)**NvRamAccessSize*) and verify interface correctness. The return status should be **EFI_SUCCESS**.
5.11.1.10.7	0x8f18c1d9, 0xbcb2, 0x4e15, 0xaa, 0x16, 0x58, 0xe8, 0x3c, 0x31, 0xd5, 0xe4	**EFI_SIMPLE_NETWORK_PROTOCOL.NvData** - Invokes **NvData()** to read ((n-1)**NvRamAccessSize*, *NvRamAccessSize*) returns **EFI_SUCCESS**.	Call **NvData()** to read ((n-1)**NvRamAccessSize*, *NvRamAccessSize*) and verify interface correctness. The return status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.11.1.10.8	0x443b58d6, 0x683c, 0x4018, 0x89, 0xc9, 0x2e, 0x70, 0xe8, 0x53, 0x6b, 0x7d	**EFI_SIMPLE_NETWORK_PROTOCOL.NvData** - Invokes **NvData()** writes and verifies interface correctness, returning **EFI_SUCCESS**.	Call **NvData()** to write and verify interface correctness. The return status should be **EFI_SUCCESS**.

14.1.11 GetStatus()

Number	GUID	Assertion	Test Description
5.11.1.11.1	0x21837ad9, 0x942b, 0x4b2b, 0x89, 0x6e, 0xc7, 0xb1, 0xe8, 0xa3, 0x6a, 0xaa	**EFI_SIMPLE_NETWORK_PROTOCOL.GetStatus** - Invokes **GetStatus()** when the network interface is not started returns **EFI_NOT_STARTED**.	Call **GetStatus()** when network interface is not started. The return status should be **EFI_NOT_STARTED** and the state should be “Stopped”.
5.11.1.11.2	0xce6f3aba, 0x9d91, 0x4ab4, 0xaa, 0x96, 0x01, 0x14, 0x3e, 0xea, 0xf8, 0x29	**EFI_SIMPLE_NETWORK_PROTOCOL.GetStatus** - Invokes **GetStatus()** when the network interface is not initialized returns **EFI_DEVICE_ERROR**.	Call **GetStatus()** when the network interface is not initialized. The return status should be **EFI_DEVICE_ERROR**.
5.11.1.11.3	0xa1ee7ee5, 0x2b46, 0x4da0, 0xb8, 0x19, 0x0d, 0x10, 0xe1, 0xd0, 0x6f, 0xc0	**EFI_SIMPLE_NETWORK_PROTOCOL.GetStatus** - Invokes **GetStatus()** with an invalid parameter returns **EFI_INVALID_PARAMETER**.	Call **GetStatus()** when both *InterruptStuts* and *TxBuf* are **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.11.1.11.4	0x8e8f1517, 0x330e, 0x45fd, 0x8d, 0x84, 0x33, 0xff, 0xf1, 0x60, 0x00, 0xf2	**EFI_SIMPLE_NETWORK_PROTOCOL.GetStatus** - Invokes **GetStatus()** verifies interface correctness and returns **EFI_SUCCESS**.	Call **GetStatus()** and verify interface correctness. The return status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.11.1.11.5	0xa32b5f48, 0x8215, 0x4024, 0x80, 0x31, 0x33, 0x70, 0x5, 0x20, 0x37, 0x54	**EFI_SIMPLE_NETWORK_PROTOCOL.GetStatus** - Invokes **GetStatus()** to verify the transmitted buffer should be shown up in the recycled transmit buffer.	1. The transmitted buffer should be shown up in the recycled transmit buffer.

14.1.12 Transmit()

Number	GUID	Assertion	Test Description
5.11.1.12.1	0xfe70e127, 0x6ea1, 0x4ff8, 0xa0, 0x41, 0x1f, 0x96, 0xad, 0x0c, 0xe8, 0x9d	**EFI_SIMPLE_NETWORK_PROTOCOL.Transmit** - Invokes **Transmit()** when the network interface is not started returns **EFI_NOT_STARTED**.	Call **Transmit()** when the network interface is not started. The return status should be **EFI_NOT_STARTED** and the state should be “Stopped”.
5.11.1.12.2	0xfdcadacb, 0x71cd, 0x416c, 0x9a, 0xa6, 0x8c, 0xf5, 0x3a, 0x85, 0x92, 0x05	**EFI_SIMPLE_NETWORK_PROTOCOL.Transmit** - Invokes **Transmit()** when the network interface is not initialized returns **EFI_DEVICE_ERROR**.	Call **Transmit()** when the network interface is not initialized. The return status should be **EFI_DEVICE_ERROR**.
5.11.1.12.3	0xea3773ea, 0x0e0f, 0x45a3, 0x82, 0xa0, 0x64, 0xd4, 0x85, 0xa1, 0x0b, 0x52	**EFI_SIMPLE_NETWORK_PROTOCOL.Transmit** - Invokes **Transmit()** with a *HeaderSize* value of non-0 and not equal to *MediaHeaderSize* returns **EFI_INVALID_PARAMETER**.	Call **Transmit()** with a *HeaderSize* value of non-0 and not equal to *MediaHeaderSize*. The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.11.1.12.4	0xde544de1, 0x178e, 0x4b5f, 0x97, 0xd7, 0x19, 0x11, 0x9b, 0x1b, 0x7b, 0x18	**EFI_SIMPLE_NETWORK_PROTOCOL.Transmit** - Invokes **Transmit()** with a *BufferSize* value of less than *MediaHeaderSize*.	Call **Transmit()** with a *BufferSize* value of less than *MediaHeaderSize*. The return status should be **EFI_INVALID_PARAMETER**.
5.11.1.12.5	0x4b33c0b2, 0x4ab8, 0x44a0, 0x8c, 0x0b, 0xd9, 0x8b, 0x70, 0x9d, 0xd1, 0x64	**EFI_SIMPLE_NETWORK_PROTOCOL.Transmit** - Invokes **Transmit()** with a *Buffer* value of **NULL** returns **EFI_INVALID_PARAMETER**.	Call **Transmit()** with a *Buffer* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.11.1.12.6	0xa449842c, 0xf5f8, 0x47e9, 0x98, 0x7b, 0x4b, 0x61, 0x41, 0xae, 0xbd, 0x45	**EFI_SIMPLE_NETWORK_PROTOCOL.Transmit** - Invokes **Transmit()** with a *HeaderSize* value of non-0 and *DestAddr* value of **NULL** returns **EFI_INVALID_PARAMETER**.	Call **Transmit()** with a *HeaderSize* value of non-0 and a *DestAddr* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.11.1.12.7	0x2e3dd087, 0xdd0c, 0x426e, 0x85, 0xba, 0x65, 0xe5, 0x83, 0x10, 0xb1, 0xde	**EFI_SIMPLE_NETWORK_PROTOCOL.Transmit** - Invokes **Transmit()** with a HeaderSize value of non-0 and a Protocol value of **NULL** returns **EFI_INVALID_PARAMETER**..	Call **Transmit()** with a *HeaderSize* value of non-0 and a Protocol value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.11.1.12.8	0x10e4090b, 0x284b, 0x4886, 0xba, 0x9b, 0x9f, 0x50, 0xc7, 0xff, 0xc5, 0x74	**EFI_SIMPLE_NETWORK_PROTOCOL.Transmit** - Invokes **Transmit()** with a *HeaderSize* value of non-0 and a *Protocol* value of not in accordance with *IfType* returning **EFI_INVALID_PARAMETER**.	Call **Transmit()** with a *HeaderSize* value of non-0 and a *Protocol* value of not in accordance with *IfType*. The return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.11.1.12.9	0xdaafbb2a, 0x434b, 0x452f, 0xa6, 0x44, 0xa7, 0x39, 0x2c, 0xf3, 0x59, 0x37	`EFI_SIMPLE_NETWORK_PROTOCOL.Transmit` - Calling `Transmit()` sends Over Sized Packets and returns `EFI_INVALID_PARAMETER`.	Call `Transmit()` to send Over Sized Packets. The return status should be `EFI_INVALID_PARAMETER`.
5.11.1.12.10	0x8f8ec6d7, 0x41b5, 0x4e06, 0x87, 0x12, 0xdb, 0x77, 0xba, 0xc6, 0x1a, 0x1f	`EFI_SIMPLE_NETWORK_PROTOCOL.Transmit` - Calling `Transmit()` sends Under Sized Packets and returns `EFI_INVALID_PARAMETER`.	Call `Transmit()` to send Under Sized Packets. The return status should be `EFI_INVALID_PARAMETER`.

14.1.13 Receive()

Number	GUID	Assertion	Test Description
5.11.1.13.1	0x6c2503ce, 0x7952, 0x4740, 0x88, 0xd2, 0xe1, 0xb3, 0xa2, 0xd9, 0x5d, 0x2e	`EFI_SIMPLE_NETWORK_PROTOCOL.Receive` - Invokes `Receive()` when the network interface is not started returns `EFI_NOT_STARTED`.	Call `Receive()` when the network interface is not started. The return status should be `EFI_NOT_STARTED` and the state should be "Stopped".
5.11.1.13.2	0xb0def89e, 0xbb48, 0x4829, 0xb5, 0x8e, 0x12, 0x7a, 0xf3, 0x7a, 0x38, 0x9d	`EFI_SIMPLE_NETWORK_PROTOCOL.Receive` - Invokes `Receive()` when the network interface is not initialized returns `EFI_DEVICE_ERROR`.	Call `Receive()` when the network interface is not initialized. The return status should be `EFI_DEVICE_ERROR`.
5.11.1.13.3	0xa6783502, 0xf69b, 0x4091, 0xac, 0x09, 0xf0, 0x10, 0x42, 0xa5, 0x93, 0x5e	`EFI_SIMPLE_NETWORK_PROTOCOL.Receive` - Invokes `Receive()` with a *`Buffer`* value of `NULL` returns `EFI_INVALID_PARAMETER`.	Call `Receive()` with a *`Buffer`* value of `NULL`. The return status should be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.11.1.13.4	0xb61dd219, 0x0b04, 0x49b7, 0x9a, 0xf9, 0x8c, 0x5f, 0x27, 0x0c, 0x44, 0x9b	`EFI_SIMPLE_NETWORK_PROTOCOL.Receive` - Invokes `Receive()` when *`BufferSize`* is smaller than the received Packets returns `EFI_INVALID_PARAMETER`.	Call `Receive()` when *`BufferSize`* is smaller than the received Packets. The return status should be `EFI_INVALID_PARAMETER`.
5.11.1.13.5	0x6a319f34, 0x0e40, 0x41aa, 0xae, 0x50, 0x16, 0x9c, 0x4d, 0xe7, 0xb8, 0xc7	`EFI_SIMPLE_NETWORK_PROTOCOL.Receive` - Invokes `Receive()` when no packet is received returns `EFI_NOT_READY`.	Call `Receive()` when no packet is received. The return status should be `EFI_NOT_READY`.

14.2 EFI_PXE_BASE_CODE_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_PXE_BASE_CODE_PROTOCOL Section.

14.2.1 Start()

Number	GUID	Assertion	Test Description
5.11.2.1.1	0x0a483bd1, 0x80cf, 0x463b, 0x8b, 0xb1, 0x2a, 0x33, 0x32, 0x90, 0xcc, 0x08	`EFI_PXE_BASE_CODE_PROTOCOL.Start` - Calling `Start()` when PXE Protocol is already started returns `EFI_ALREADY_STARTED`.	Call `Start()` when the `EFI_PXE_BASE_CODE_PROTOCOL` is already started. The return code should be `EFI_ALREADY_STARTED`.
5.11.2.1.2	0xc1505aee, 0xd73a, 0x416c, 0x9a, 0x3f, 0x9c, 0x00, 0x5d, 0x01, 0xd6, 0xeb	`EFI_PXE_BASE_CODE_PROTOCOL.Start` - Calling `Start()` using IPV6 when PXE Protocol does not support IPV6 returns `EFI_NOT_SUPPORTED`.	Call `Start()` when `EFI_PXE_BASE_CODE_PROTOCOL` does not support IPV6, but require its use. The return code should be `EFI_NOT_SUPPORTED`.

Number	GUID	Assertion	Test Description
5.11.2.1.3	0x13a4a599, 0xb35b, 0x4465, 0xa2, 0xdb, 0xc1, 0xe8, 0xa4, 0xca, 0x9a, 0x93	EFI_PXE_BASE_CODE_PROTOCOL.Start – Calling Start() without using IPv6 returns EFI_SUCCESS.	Call Start() without using IPv6. The return status code should be EFI_SUCCESS. Call Start() with using IPv6 if Ipv6Supported is FALSE. The return status code should be EFI_UNSUPPORTED.
5.11.2.1.4	0x33067ad5, 0xb3a5, 0x44f4, 0x9f, 0xf5, 0xf8, 0x63, 0xda, 0x1f, 0xbd, 0xb3	EFI_PXE_BASE_CODE_PROTOCOL.Start – Calling Start() returns correct mode without using IPv6.	Call Start() without using IPv6. The return mode should be correct, including Started is TRUE, UsingIpv6 is FALSE, AutoArp is TRUE, and Route Table is Empty.

14.2.2 Stop()

Number	GUID	Assertion	Test Description
5.11.2.2.1	0x8d75ffa1, 0xdfab, 0x4aff, 0x9f, 0xf7, 0xbb, 0x49, 0x49, 0x08, 0xdc, 0xa3	EFI_PXE_BASE_CODE_PROTOCOL.Stop – Calling Stop() while the PXE protocol is already stopped returns EFI_NOT_STARTED.	Call Stop() when the EFI_PXE_BASE_CODE_PROTOCOL is already stopped. The return code should be EFI_NOT_STARTED.
5.11.2.2.2	0xf88713ff, 0xf149, 0x4e9f, 0x8c, 0xf5, 0x6d, 0x63, 0x55, 0x8f, 0xf2, 0xbd	EFI_PXE_BASE_CODE_PROTOCOL.Stop – Calling Stop() to disable PXE protocol when it is enabled returns EFI_SUCCESS.	Enable PXE protocol, and call Stop() to disable PXE protocol. The return code should be EFI_SUCCESS.

14.2.3 Dhcp()

No automatic test is designed to verify this function.

14.2.4 Discover()

No automatic test is designed to verify this function.

14.2.5 Mtftp()

No automatic test is designed to verify this function.

14.2.6 UdpWrite()

No automatic test is designed to verify this function.

14.2.7 UdpRead()

No automatic test is designed to verify this function.

14.2.8 SetIpFilter()

Number	GUID	Assertion	Test Description
5.11.2.8.1	0x670cf69f, 0x530a, 0x4bec, 0xaa, 0xb8, 0x41, 0xd3, 0x58, 0x9e, 0x91, 0x99	**EFI_PXE_BASE_CODE_PROTOCOL.SetIpFilter** –Calling **SetIpFilter()** returns **EFI_SUCCESS**.	Enable PXE protocol, and call **SetIpFilter()**. The returned code should be **EFI_SUCCESS**.
5.11.2.8.2	0xe9ed28b0, 0x0b88, 0x4e4e, 0xa2, 0xdb, 0xe5, 0xc4, 0xea, 0xd2, 0x00, 0x87	**EFI_PXE_BASE_CODE_PROTOCOL.SetIpFilter** – Calling **SetIpFilter()** updates *IpFilter* Mode setting.	Enable PXE protocol, and call **SetIpFilter()**. The *IpFilter* filed at **EFI_PXE_BASE_CODE_MODE** is updated to the new setting.
5.11.2.8.3	0x13317b8d, 0x5d0d, 0x400f, 0x87, 0x4f, 0xaf, 0xe5, 0x08, 0xf1, 0x35, 0x86	**EFI_PXE_BASE_CODE_PROTOCOL.SetIpFilter** – Calling **SetIpFilter()** with PXE protocol not started returns **EFI_NOT_STARTED**.	Disable PXE protocol, and call **SetIpFilter()**. The return code should be **EFI_NOT_STARTED**.

14.2.9 Arp()

Number	GUID	Assertion	Test Description
5.11.2.13.1	0xdc8b9346, 0xc5c8, 0x4ef5, 0xaf, 0x22, 0xcd, 0xef, 0x81, 0x6d, 0xf6, 0x13	**EFI_PXE_BASE_CODE_PROTOCOL.Arp – Arp()** returns **EFI_INVALID_PARAMETER** when **IpAddr** is NULL	1.Call **Arp()** with **IpAddr** = NULL. The return code must be **EFI_INVALID_PARAMETER**
5.11.2.13.2	0xe893562b, 0xcb51, 0x409c, 0xa0, 0x93, 0x7c, 0xad, 0xe1, 0x43, 0xd6, 0xc0	**EFI_PXE_BASE_CODE_PROTOCOL.Arp – Arp()** returns **EFI_UNSUPPORTED** when **UsingIpv6** is TRUE	1.Call **Arp()** when **UsingIpv6** is TRUE. The return code must be **EFI_UNSUPPORTED**

14.2.10 SetParameters()

Number	GUID	Assertion	Test Description
5.11.2.10.1	0x3395102a, 0x1b16, 0x4267, 0xb8, 0x5e, 0x88, 0x4b, 0xd6, 0x56, 0xb8, 0x69	**EFI_PXE_BASE_CODE_PROTOCOL.SetParameters** – Calling **SetParameters()** with PXE protocol not started returns **EFI_NOT_STARTED**.	Disable PXE protocol, and call **SetParameters()**. The return code should be **EFI_NOT_STARTED**.

14.2.11 SetStationIp()

Number	GUID	Assertion	Test Description
5.11.2.11.1	0xe20afad4, 0x04e5, 0x4b09, 0xa2, 0x3a, 0xc0, 0xc1, 0xd5, 0x7f, 0x8b, 0x1b	**EFI_PXE_BASE_CODE_PROTOCOL.SetStationIp** – Calling **SetStationIp()** and modifying IP address and subnet mask returns **EFI_SUCCESS**.	Enable PXE protocol, and call **SetStationIp()** to modify IP address and subnet mask. The returned status code is **EFI_SUCCESS**.
5.11.2.11.2	0x47feb998, 0x7d0d, 0x4381, 0xae, 0x31, 0x71, 0xbe, 0xdf, 0xb0, 0x73, 0x23	**EFI_PXE_BASE_CODE_PROTOCOL.SetStationIp** – Calling **SetStationIp()** and only modifying IP address returns **EFI_SUCCESS**.	Enable PXE protocol, and call **SetStationIp()** only to modify IP address. The returned status code is **EFI_SUCCESS**.
5.11.2.11.3	0x78014f26, 0x0196, 0x4d38, 0xb6, 0xbd, 0x0c, 0x7c, 0x41, 0xf8, 0x5e, 0xa1	**EFI_PXE_BASE_CODE_PROTOCOL.SetStationIp** – Calling **SetStationIp()** and only modifying subnet mask returns **EFI_SUCCESS**.	Enable PXE protocol, and call **SetStationIp()** only to modify subnet mask of the network device. The returned status code is **EFI_SUCCESS**.
5.11.2.11.4	0x518491e5, 0xd4ab, 0x42c6, 0x8c, 0x73, 0x90, 0xc1, 0xeb, 0xc2, 0xf1, 0x78	**EFI_PXE_BASE_CODE_PROTOCOL.SetStationIp** – Calling **SetStationIp()** with PXE not started returns **EFI_NOT_STARTED**.	Disable PXE protocol, and call **SetStationIp()**. The return code should be **EFI_NOT_STARTED**.

14.2.12 SetPackets()

Number	GUID	Assertion	Test Description
5.11.2.12.1	0x66c10d09, 0x2578, 0x48b7, 0x80, 0x5b, 0x75, 0xd7, 0x17, 0xcf, 0x71, 0x49	`EFI_PXE_BASE_CODE_PROTOCOL.SetPackets` – Calling `SetPackets()` with PXE protocol not started returns `EFI_NOT_STARTED`.	Disable PXE protocol, and call `SetPackets()`. The return code should be `EFI_NOT_STARTED`.

14.3 EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL Section.

The **EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL** Test is covered in the test for the EFI PXE Base Code Protocol.

14.4 EFI_BIS_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_BIS_PROTOCOL Section.

No automatic test is designed to verify this protocol.

15 Protocols Compression Test

15.1 EFI_DECOMPRESS_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_DECOMPRESS_PROTOCOL Section.

15.1.1 GetInfo()

Number	GUID	Assertion	Test Description
5.13.1.1.1	0xb4929cbe, 0x0d83, 0x481f, 0x89, 0xc7, 0xb8, 0xbd, 0x49, 0x05, 0x7c, 0xae	**EFI_DECOMPRESS_PROTOCOL.GetInfo** - Calling **GetInfo()** returns **EFI_SUCCESS**.	1. Get the Compressed file name and uncompressed file size from the profile. 2. Read the Compressed file into memory 3. Call **GetInfo()** to retrieve the decompression info. The returned status should be **EFI_SUCCESS**.
5.13.1.1.2	0x1c5d4afb, 0x66b2, 0x4ff3, 0xb9, 0x20, 0x6a, 0x21, 0x32, 0x62, 0x9f, 0xae	**EFI_DECOMPRESS_PROTOCOL.GetInfo** - Calling **GetInfo()** returns a *DestinationSize* that is equal to the Uncompressed File Size.	1. Get the Compressed file name and uncompressed file size from the profile. 2. Read the Compressed file into memory. 3. Call **GetInfo()** to retrieve the decompression info. The returned *DestinationSize* should equal the Uncompressed File Size gotten from the profile.
5.13.1.1.3	0x01a92787, 0x0d15, 0x4213, 0x92, 0x06, 0x8a, 0x3a, 0xb4, 0xa3, 0xba, 0x54	**EFI_DECOMPRESS_PROTOCOL.GetInfo** - Calling **GetInfo()** the second time returns **EFI_SUCCESS**.	1. Get the Compressed file name and uncompressed file size from the profile. 2. Read the Compressed file into memory. 3. Call **GetInfo()** to retrieve the decompression info. 4. Call **GetInfo()** again. The returned status should be **EFI_SUCCESS**.
5.13.1.1.4	0xb80b38e3, 0x3f4c, 0x43e0, 0xb8, 0x6d, 0x5b, 0x01, 0x38, 0xbd, 0x0f, 0x3e	**EFI_DECOMPRESS_PROTOCOL.GetInfo** - Calling **GetInfo()** the second time returns a *DestinationSize* that is equal to the *DestinationSize* returned after the first call.	1. Get the Compressed file name and uncompressed file size from the profile. 2. Read the Compressed file into memory. 3. Call **GetInfo()** to retrieve the decompression info. 4. Call **GetInfo()** again. The returned *DestinationSize* should be the same value as the first time.

Number	GUID	Assertion	Test Description
5.13.1.1.5	0x43ee9ff0, 0x4867, 0x4fe6, 0xac, 0x09, 0x72, 0x0a, 0x33, 0x8b, 0x80, 0xd8	`EFI_DECOMPRESS_PROTOCOL.GetInfo` - Calling `GetInfo()` the second time returns a *ScratchSize* that is equal to the *ScratchSize* returned after the first call.	1. Get the Compressed file name and uncompressed file size from the profile. 2. Read the Compressed file into memory. 3. Call `GetInfo()` to retrieve the decompression info. 4. Call `GetInfo()` again. The returned *ScratchSize* should be the same value as the first time.
5.13.1.1.6	0x66c06d59, 0x77ab, 0x4bc6, 0x98, 0x20, 0xbf, 0x01, 0x60, 0xd6, 0x1e, 0x6a	`EFI_DECOMPRESS_PROTOCOL.GetInfo` - Calling `GetInfo()` with *SourceSize* < 8 returns `EFI_INVALID_PARAMETER`.	Call `GetInfo()` with *SourceSize* < 8. The returned status should be `EFI_INVALID_PARAMETER`.

15.1.2 Decompress()

Number	GUID	Assertion	Test Description
5.13.1.2.1	0x37d2514e, 0x27f0, 0x4182, 0xb7, 0x13, 0x14, 0xf4, 0xbf, 0x53, 0xbb, 0xae	`EFI_DECOMPRESS_PROTOCOL.Decompress` – Calling `Decompress()` on a 0 length file returns `EFI_SUCCESS`.	1. Get the Compressed file name and uncompressed file name from the profile. 2. Read the Compressed file and uncompressed file into memory. 3. Call `GetInfo()` to retrieve the decompression info. 4. Call `Decompress()` with the compressed file buffer. The returned status should be `EFI_SUCCESS`.
5.13.1.2.2	0xf2665735, 0x8992, 0x47bc, 0xb2, 0x99, 0x8a, 0x00, 0x32, 0xab, 0x59, 0x93	`EFI_DECOMPRESS_PROTOCOL.Decompress` - Calling `Decompress()` on a 0 length file does not modify the buffer.	1. Get the Compressed file name and uncompressed file name from the profile. 2. Read the Compressed file and uncompressed file into memory. 3. Call `GetInfo()` to retrieve the decompression info. 4. Call `Decompress()` with the compressed file buffer. If the uncompressed file size is 0, the destination buffer should not be modified.

Number	GUID	Assertion	Test Description
5.13.1.2.3	0x8eceea13, 0x34ce, 0x43af, 0xbf, 0x9c, 0xb8, 0x3d, 0xe6, 0x32, 0x29, 0x69	**EFI_DECOMPRESS_PROTOCOL.Decompress** - Calling **Decompress()** on a non-0 file returns **EFI_SUCCESS**.	1. Get the Compressed file name and uncompressed file name from the profile. 2. Read the Compressed file and uncompressed file into memory. 3. Call **GetInfo()** to retrieve the decompression info. 4. Call **Decompress()** with the compressed file buffer. The returned status should be **EFI_SUCCESS**.
5.13.1.2.4	0xd8aa9038, 0xc3d1, 0x4f9c, 0x9d, 0xbb, 0x3c, 0xc8, 0x6d, 0xee, 0xd1, 0xe6	**EFI_DECOMPRESS_PROTOCOL.Decompress** – After calling **Decompress()** on a non-0 file, the Decompressed data is equal to the Uncompressed data.	1. Get the Compressed file name and uncompressed file name from the profile. 2. Read the Compressed file and uncompressed file into memory. 3. Call **GetInfo()** to retrieve the decompression info. 4. Call **Decompress()** with the compressed file buffer. If the uncompressed file size is non-0, the Decompressed data should be equal to the Uncompressed file data.
5.13.1.2.5	0x9e6e6f21, 0x15f3, 0x4b0c, 0x9a, 0x9a, 0x17, 0xfc, 0xab, 0x5c, 0x54, 0x23	**EFI_DECOMPRESS_PROTOCOL.Decompress** - After calling **Decompress()** with an invalid compressed file, the returned status is **EFI_INVALID_PARAMETER**.	1. Get the invalid compressed format file name from the profile. 2. Call **GetInfo()** to retrieve the decompression info. 3. Call **Decompress()** with an invalid compress format buffer. The returned status should be **EFI_INVALID_PARAMETER**.
5.13.1.2.6	0xe145f85e, 0xcc48, 0x42d4, 0xab, 0x48, 0xb5, 0x16, 0x2f, 0xc3, 0xef, 0xae	**EFI_DECOMPRESS_PROTOCOL.Decompress** - Calling **Decompress()** with an incorrect *SourceSize* (*SourceSize* - 1) returns **EFI_INVALID_PARAMETER**.	1. Read the Compressed file into memory and save the buffer pointer. 2. Call **GetInfo()** to retrieve the decompression info. 3. Call **Decompress()** with incorrect *SourceSize* (*SourceSize* - 1) The returned status should be **EFI_INVALID_PARAMETER**.
5.13.1.2.7	0xfdc75fd3, 0x3a02, 0x48e5, 0x8d, 0x7f, 0x0b, 0x14, 0x75, 0xb5, 0xcf, 0x1c	**EFI_DECOMPRESS_PROTOCOL.Decompress** - Calling **Decompress()** with *SourceSize* < 8 returns **EFI_INVALID_PARAMETER**.	1. Read the Compressed file into memory and save the buffer pointer. 2. Call **GetInfo()** to retrieve the decompression info. 3. Call **Decompress()** with *SourceSize* < 8. The returned status should be **EFI_INVALID_PARAMETER**.

16 Protocols Debugger Support Test

16.1 EFI_DEBUG_SUPPORT_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_DEBUG_SUPPORT_PROTOCOL Section.

16.1.1 GetMaximumProcessorIndex()

Number	GUID	Assertion	Test Description
5.12.1.1.1	0x2ac7927c, 0xd9df, 0x4c32, 0x87, 0xb4, 0xad, 0x0a, 0xc4, 0xbb, 0xd5, 0x92	**`EFI_DEBUG_SUPPORT_PROTOCOL.GetMaximumProcessorIndex`** - Invokes **`GetMaximumProcessorIndex()`** returns **`EFI_SUCCESS`** and the out parameter contains a UINTN value.	Call **`GetMaximumProcessorIndex()`**. It should return **`EFI_SUCCESS`** and the out parameter should contain a UINTN value.

16.1.2 RegisterPeriodicCallback()

Number	GUID	Assertion	Test Description
5.12.1.2.1	0x1e43071e, 0xa00d, 0x46eb, 0xbd, 0xdd, 0x8f, 0x54, 0x22, 0xef, 0x24, 0x30	**`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterPeriodicCallback`** - Invokes **`RegisterPeriodicCallback()`** installs an interrupt handler function and returns **`EFI_SUCCESS`**.	Call **`RegisterPeriodicCallback()`** with a valid interrupt handler function. The return code should be **`EFI_SUCCESS`**.
5.12.1.2.2	0x792e517a, 0xf006, 0x46e6, 0xb3, 0x19, 0xc0, 0xc8, 0x7e, 0x43, 0x8b, 0x32	**`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterPeriodicCallback`** - The SYSTEM_TIMER_VECTOR interrupt invokes the **`PeriodicCallback()`**.	Wait for the **`PeriodicCallback()`** to be invoked by the SYSTEM_TIMER_VECTOR interrupt. The **`PeriodicCallback()`** should be invoked.

Number	GUID	Assertion	Test Description
5.12.1.2.3	0xef21928d, 0xa7c3, 0x4c92, 0xaa, 0x22, 0x97, 0xc3, 0x3d, 0x4d, 0xd2, 0x00	`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterPeriodicCallback` - The `PeriodicCallback()` is invoked earlier than the time event callback function.	Create a time event and register a callback function for it with less time than the machine clock. Wait for two callback functions to be invoked. The `PeriodicCallback()` should be invoked earlier than the time event callback function.
5.12.1.2.4	0x9f3d4d83, 0xee41, 0x41dd, 0x83, 0x13, 0x6c, 0xc0, 0x59, 0x7f, 0x22, 0x21	`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterPeriodicCallback` - Invokes `RegisterPeriodicCallback()` installs another interrupt handler function and returns `EFI_ALREADY_STARTED`.	Call `RegisterPeriodicCallback()` with a valid interrupt handler function. The return code should be `EFI_ALREADY_STARTED`.
5.12.1.2.5	0x29778e36, 0x09ad, 0x47db, 0x82, 0x4c, 0x5b, 0x46, 0x25, 0xd0, 0xe5, 0xb4	`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterPeriodicCallback` - Invokes `RegisterPeriodicCallback()` unstalls the interrupt handler function and returns `EFI_SUCCESS`.	Call `RegisterPeriodicCallback()` with a `NULL` interrupt handler function. The return code should be `EFI_SUCCESS`.
5.12.1.2.6	0xc34688c4, 0x9f84, 0x40a7, 0x90, 0x84, 0xe6, 0x5e, 0x2c, 0xbe, 0xae, 0x45	`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterPeriodicCallback` - The `PeriodicCallback()` is not invoked after the `SYSTEM_TIMER_VECTOR` interrupt.	Wait for the `SYSTEM_TIMER_VECTOR` interrupt. The `PeriodicCallback()` should not be invoked.

16.1.3 RegisterExceptionCallback()

Number	GUID	Assertion	Test Description
5.12.1.3.1	0x20bc4ac1, 0x8958, 0x446a, 0x8b, 0x5f, 0x27, 0xb3, 0xcc, 0x77, 0x41, 0x06	`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterExceptionCallback` - Invokes `RegisterExceptionCallback()` installs an interrupt handler function.	Call `RegisterExceptionCallback()` with a valid InterrruptHandler function, the exception type is EXCEPT_IA32_BREAKPOINT. The return code should be `EFI_SUCCESS`.
5.12.1.3.2	0xfbfa47e8, 0xbd32, 0x4f81, 0x89, 0x38, 0xb7, 0x36, 0x47, 0x08, 0xa2, 0xb9	`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterExceptionCallback` - Calling INT3 invokes the interrupt handler function.	Use “INT 3” instruction to invokes the interrupt. After “INT 3” is called, the interrupt handler function should be invoked.
5.12.1.3.3	0x14362c36, 0xf284, 0x4a95, 0xab, 0x1b, 0x3b, 0x67, 0xa9, 0x6e, 0x1d, 0xe8	`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterExceptionCallback` - Invokes `RegisterPeriodicCallback()` installs the Periodic interrupt handler function and two callback functions are invoked.	Call `RegisterPeriodicCallback()` with a valid InterrruptHandler function.Use “INT 3” instruction to invokes the Exception callback function, and wait for the periodic callback function to be invoked. The return code of `RegisterPeriodicCallback()` should be `EFI_SUCCESS`.Two callback functions should be invoked successfully.
5.12.1.3.4	0x0cf314a2, 0xfe51, 0x4093, 0xb4, 0x22, 0x9f, 0x4a, 0x90, 0x98, 0xd2, 0x89	`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterExceptionCallback` - Invokes `RegisterExceptionCallback()` installs another interrupt handler function.	Call `RegisterExceptionCallback()` with a valid InterrruptHandler function. The return code should be `EFI_ALREADY_STARTED`.
5.12.1.3.5	0x28e232bd, 0xfe72, 0x4963, 0xb3, 0x33, 0x1e, 0x83, 0x61, 0x5e, 0x1e, 0x2e	`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterExceptionCallback` - Invokes `RegisterExceptionCallback()` uninstalls the interrupt handler function.	Call `RegisterExceptionCallback()` with `NULL` InterrruptHandler function. The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.12.1.3.6	0x59efd2fb, 0x2f7d, 0x4535, 0xa2, 0x1c, 0x39, 0x25, 0xcb, 0xb3, 0x0b, 0x87	`EFI_DEBUG_SUPPORT_PROTOCOL.RegisterExceptionCallback` - Using “INT 3“ instruction does not invokes the previously installed (but now uninstalled) interrupt handler function.	Use “INT 3“ instruction to invokes the interrupt. After “INT 3” is called, the previously installed (but now uninstalled) interrupt handler function should not be invoked.

16.1.4 InvalidateInstructionCache()

Number	GUID	Assertion	Test Description
5.12.1.4.1	0x41c3bc2c, 0xf066, 0x4272, 0xac, 0xa7, 0xb9, 0x48, 0x9f, 0xac, 0x94, 0x2b	`EFI_DEBUG_SUPPORT_PROTOCOL.InvalidateInstructionCache` - Invokes `InvalidateInstructionCache()` returns `EFI_SUCCESS`, verifying interface correctness.	Call `InvalidateInstructionCache()`.The return code should be `EFI_SUCCESS`.

16.1.5 Isa

Number	GUID	Assertion	Test Description
5.12.1.5.1	0x701d9223, 0x1123, 0x40a2, 0xa8, 0x81, 0x5f, 0xd6, 0x68, 0xeb, 0x32, 0x87	`EFI_DEBUG_SUPPORT_PROTOCOL.Isa` – The instruction is IA32, IPF, or EBC.	Get the Isa value, it should be IA32 (0x014C), IPF (0x0200), or EBC (0xEBC).

16.2 EFI_DEBUGPORT_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_DEBUGPORT_PROTOCOL Section.

16.2.1 Reset()

Number	GUID	Assertion	Test Description

5.12.2.1.1	0x6aca7c62, 0x7bbe, 0x4d1b, 0x9c, 0x8a, 0xc7, 0x7a, 0x6c, 0x68, 0x74, 0x76	`EFI_DEBUGPORT_PROTOCOL.Reset` - Invokes `Reset()` returns `EFI_SUCCESS`, verifying interface correctness within test case.	Call `Reset()`.It should return `EFI_SUCCESS`.

16.2.2 Write()

No automatic test is designed to verify this function.

16.2.3 Read()

No automatic test is designed to verify this function.

16.2.4 Poll()

Number	GUID	Assertion	Test Description
5.12.2.4.1	0x4bf087b2, 0xe914, 0x4056, 0x8e, 0x1a, 0x25, 0xf0, 0x13, 0x54, 0x31, 0x26	`EFI_DEBUGPORT_PROTOCOL.Poll` - Calling `Poll()` when the debug port has data returns `EFI_SUCCESS`.	Call `Write()` to send data to the debug port. Call `Poll()` to check the debug port to see if any data is available to be read. The return code of `Poll()` should be `EFI_SUCCESS`.
5.12.2.4.2	0x838a1da2, 0x9640, 0x47f3, 0xba, 0xc1, 0x39, 0x26, 0xf3, 0x1d, 0x00, 0xc2	`EFI_DEBUGPORT_PROTOCOL.Poll` - Calling `Poll()` when the debug port does not have data returns `EFI_NOT_READY`.	Call `Reset()` to reset the debug port. Call `Poll()` to check the debug port to see if any data is available to be read. The return code of `Poll()` should be `EFI_NOT_READY`.

17 Protocols ACPI Test

17.1 EFI_ACPI_TABLE_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_ACPI_TABLE_PROTOCOL Section.

17.1.1 InstallAcpiTable ()

Number	GUID	Assertion	Test Description
5.16.1.1.1	0x278963cf, 0x0c77, 0x47b5, 0xa9, 0x1f, 0x2b, 0xa7, 0xde, 0x9d, 0xa3, 0x75	**ACPI_TABLE_PROTOCOL.InstallAcpiTable - InstallAcpiTable()** returns **EFI_INVALID_PARAMETER** with **NULL** AcpiTableBuffer.	Call **InstallAcpiTable()** with **NULL AcpiTableBuffer**. The return status should be **EFi_INVALID_PARAMETER**.
5.16.1.1.2	0xa3f1e4b1, 0xe8d9, 0x4516, 0xa2, 0xbc, 0x3d, 0xef, 0x20, 0x15, 0xec, 0x7d	**ACPI_TABLE_PROTOCOL.InstallAcpiTable - InstallAcpiTable()** returns **EFI_INVALID_PARAMETER** with **NULL** TableKey.	Call **InstallAcpiTable()** with **NULL TableKey**. The return status should be **EFi_INVALID_PARAMETER**.
5.16.1.1.3	0xb03fa7b4, 0xeb94, 0x4f56, 0x8a, 0x69, 0x5a, 0x13, 0x59, 0xcf, 0x57, 0x3f	**ACPI_TABLE_PROTOCOL.InstallAcpiTable - InstallAcpiTable()** returns **EFI_INVALID_PARAMETER** with AcpiTableBufferSize is different with the size field in AcpiTableBuffer.	Call **InstallAcpiTable()** with the size of **AcpiTableBuffer** not the same as the **AcpiTableBufferSize**. The return status should be **EFi_INVALID_PARAMETER**.
5.16.1.1.4	0x40949ceb, 0x734b, 0x468d, 0x88, 0xca, 0xfe, 0xc2, 0x7e, 0x4c, 0x19, 0xd2	**ACPI_TABLE_PROTOCOL.InstallAcpiTable - InstallAcpiTable()** returns **EFI_SUCCESS** with valid parameters	Call **InstallAcpiTable()** with valid parameter. The return status should be **EFI_SUCCESS**. Call **UninstallAcpiTable()** to restore the environment.

Number	GUID	Assertion	Test Description
5.16.1.1.5	0xfd58070a, 0xcefe, 0x4aea, 0x90, 0x3b, 0xa7, 0xa9, 0xbe, 0x53, 0x9c, 0xaf	`ACPI_TABLE_PROTOCOL.InstallAcpiTable-InstallAcpiTable()` returns `EFI_SUCCESS` and automatically correct `AcpiTable` checksum	1. Call `InstallAcpiTable()` with `AcpiTable` with wrong checksum. 2. The return status should be `EFI_SUCCESS` & `AcpiTable` checksum corrected. 3. Call `UninstallAcpiTable()` to restore the environment.

17.1.2 UninstallAcpiTable ()

Number	GUID	Assertion	Test Description
5.16.1.2.1	0x5c72198c, 0x74d2, 0x4c55, 0xb9, 0xcf, 0x17, 0xdc, 0x02, 0x30, 0xac, 0x71	`ACPI_TABLE_PROTOCOL.UninstallAcpiTable - UninstallAcpiTable()` returns `EFI_NOT_FOUND` with `TableKey` not refer to a table entry.	Call `InstallAcpiTable()` with valid parameter. The return status should be `EFI_SUCCESS`. Call `UninstallAcpiTable()`. The return status should be `EFI_SUCCESS`. Call `UninstallAcpiTable()` again. The return status should `EFI_NOT_FOUND`..
5.16.1.2.2	0xf1c7de32, 0xd0fe, 0x4d67, 0xb0, 0x28, 0x06, 0xb4, 0xa0, 0x84, 0x06, 0xc4	`ACPI_TABLE_PROTOCOL.UninstallAcpiTable - UninstallAcpiTable()` returns `EFI_SUCCESS` with `TableKey` refer to a table entry.	Call `InstallAcpiTable()` with valid parameter. The return status should be `EFI_SUCCESS`. Call `UninstallAcpiTable()`. The return status should be `EFI_SUCCESS`.

18 Network Protocols Managed Network

18.1 EFI_MANAGED_NETWORK_PROTOCOL Test

Reference Document:

UEFI 2.0 Specification, Section 21.

18.1.1 GetModeData()

Number	GUID	Assertion	Test Description
5.23.1.1.1	0xfd5600b1, 0x958d, 0x4cf3, 0x9a, 0x6a, 0xb4, 0x5e, 0x26, 0x73, 0x19, 0xc6	**EFI_MANAGED_NETWORK_PROTOCOL.GetModeData** – invokes **GetModeData()** with a *MnpConfigData* value other than **NULL** when the MNP child has not been configured.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.GetModeData()** with a *MnpConfigData* value other than **NULL** when the MNP child has not been configured. The return status should be **EFI_NOT_STARTED**, and the default values are returned in *MnpConfigData*. 3. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.1.2	0xf39fc5b4, 0xcea9, 0x498d, 0xb7, 0xe4, 0xce, 0x0a, 0x7c, 0x9e, 0x0b, 0x35	**EFI_MANAGED_NETWORK_PROTOCOL.GetModeData** – invokes **GetModeData()** to get the previously configured data.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameter for the child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.GetModeData()** to get the previously configured data in step 2, 4. Verify the data. The return status should be **EFI_SUCCESS**. 5. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.1.3	0x5b579cdd, 0xae9b, 0x4415, 0xbd, 0xc0, 0x39, 0xb0, 0x14, 0xcf, 0x29, 0xe2	**EFI_MANAGED_NETWORK_PROTOCOL.GetModeData** – invokes **GetModeData()** with a *MnpConfData* value of **NULL** and a *SnpModeData* value of **NULL**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameter for the child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.GetModeData()** with a *MnpConfData* value of **NULL** and a *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.1.4	0xd34ce9f5, 0x8fb5, 0x4f50, 0xac, 0x68, 0x64, 0x0e, 0xc9, 0x3b, 0xc0, 0xbf	**EFI_MANAGED_NETWORK_PROTOCOL.GetModeData** – invokes **GetModeData()** with a *MnpConfData* value of **NULL**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameter for the child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.GetModeData()** with a *MnpConfData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.1.5	0xbde40b90, 0xf94f, 0x4c26, 0xac, 0x32, 0x21, 0x07, 0xa4, 0x19, 0x82, 0xde	**EFI_MANAGED_NETWORK_PROTOCOL.GetModeData** – invokes **GetModeData()** with a *SnpModeData* value of **NULL**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameter for the child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.GetModeData()** with a *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

18.1.2 Configure()

Number	GUID	Assertion	Test Description
5.23.1.2.1	0x4c4b70cd, 0x5492, 0x440f, 0x87, 0xd8, 0xc8, 0x4d, 0x0b, 0x61, 0x02, 0x9f	**EFI_MANAGED_NETWORK_PROTOCOL.Configure** – invokes **Configure()** with an invalid *MnpConfigData.ProtocolTypeFilter* value.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** with an invalid *MnpConfigData.ProtocolTypeFilter* value. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.2.2	0x437bdc0d, 0xe159, 0x4535, 0x92, 0xe0, 0x56, 0x59, 0xd7, 0xa4, 0xc7, 0xfc	**EFI_MANAGED_NETWORK_PROTOCOL.Configure** – invokes **Configure()** after creating a new MNP child.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild**() to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. The return status should be **EFI_SUCCESS.** 3. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.2.3	0x3d69e8d4, 0x34fa, 0x4a15, 0xaa, 0xb1, 0x95, 0x48, 0x13, 0x9a, 0x62, 0x59	**EFI_MANAGED_NETWORK_PROTOCOL.Configure** – invokes **Configure()** with unicast and broadcast disabled, which means set the parameter *EnableUnicastReceive* and *EnableBroadcaseReceive* set to **FALSE**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** with the parameter *EnableUnicastReceive* and *EnableBroadcaseReceive* a set to **FALSE**. The return status should be **EFI_SUCCESS.** 3. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.2.4	0x5e075f02, 0x708d, 0x4c3d, 0x8e, 0xc6, 0x53, 0x91, 0x6c, 0x30, 0xf4, 0x2b	**EFI_MANAGED_NETWORK_PROTOCOL.Configure** – invokes **Configure()** when the configuration data is reset to **NULL.**	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** when the configuration data is reset to **NULL.** The return status should be **EFI_SUCCESS.** 3. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.2.5	0xfbbaf8a7, 0x91ac, 0x497a, 0x9f, 0x9d, 0xec, 0x0a, 0x35, 0x34, 0xa1, 0xd7	EFI_MANAGED_NETWORK_PROTOCOL.Configure – invokes Configure() when *ReceiveQueueTimeout* is enabled.	1. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new MNP child. 2. Call EFI_MANAGED_NETWORK_PROTOCOL.Configure() when *ReceiveQueueTimeout* is enabled. The return status should be EFI_SUCCESS. 3. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created MNP child and clean up the environment.

18.1.3 McastIpToMac()

Number	GUID	Assertion	Test Description
5.23.1.3.1	0x5902f01b, 0x124a, 0x4fe9, 0x98, 0xfa, 0x07, 0x97, 0x71, 0x4b, 0x39, 0xc3	EFI_MANAGED_NETWORK_PROTOCOL.McastIpToMac – invokes McastIpToMac() when the child has not been configured.	1. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new MNP child. 2. Call EFI_MANAGED_NETWORK_PROTOCOL.McastIpToMac() when the child has not been configured. The return status should be EFI_NOT_STARTED. 3. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created MNP child and clean up the environment.

5.23.1.3.2	0x0b2990e3, 0xc947, 0x4121, 0xb8, 0xa5, 0x9c, 0x47, 0x7b, 0xac, 0x28, 0xf7	**EFI_MANAGED_NETWORK_PROTOCOL. McastIpToMac –** invokes **McastIpToMac()** with an *IpAddress* value of **NULL.**	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.McastIpToMac()** with an *IpAddress* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.3.3	0x0227a52e, 0x22b9, 0x4c6a, 0x8e, 0x13, 0x06, 0x62, 0x4c, 0x92, 0x39, 0x7f	**EFI_MANAGED_NETWORK_PROTOCOL. McastIpToMac –** invokes **McastIpToMac()** with an *IpAddress* value that is an invalid multicast IP address**.**	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.McastIpToMac()** with an *IpAddress* value that is an invalid multicast IP address. The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

5.23.1.3.4	0x318eae7a, 0xa94d, 0x4eec, 0xbf, 0xde, 0x4e, 0x04, 0x04, 0xe3, 0x2c, 0x34	**EFI_MANAGED_NETWORK_PROTOCOL. McastIpToMac –** invokes **McastIpToMac()** with a *MacAddress* value of **NULL.**	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.McastIpToMac()** with a *MacAddress* value of **NULL.** The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.3.5	0x8571d2b8, 0xe8e9, 0x450a, 0x84, 0x58, 0xf8, 0xb4, 0xa4, 0xa4, 0xc6, 0x5d	**EFI_MANAGED_NETWORK_PROTOCOL. McastIpToMac –** invokes **McastIpToMac()** with the parameter *Ipv6Flag* set to **TRUE.**	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.McastIpToMac()** with the parameter *Ipv6Flag* set to **TRUE.** The return status should be **EFI_UNSUPPORTED.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

5.23.1.3.6	0xa6a2d468, 0x07b3, 0x47d7, 0x82, 0xec, 0x76, 0x85, 0x92, 0x6a, 0x78, 0x09	**EFI_MANAGED_NETWORK_PROTOCOL.McastIpToMac –** invokes **McastIpToMac()** to change multicast IPv4 address to MAC.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.McastIpToMac()** to change multicast IPv4 address to MAC. The return status should be **EFI_SUCCESS.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

18.1.4 Groups()

Number	GUID	Assertion	Test Description
5.23.1.4.1	0xdae4ffb7, 0x4cc2, 0x4d04, 0xbe, 0x90, 0xef, 0xd1, 0x9e, 0x62, 0x94, 0xd8	**EFI_MANAGED_NETWORK_PROTOCOL.Groups –** invokes **Groups()** when the child has not been configured.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Groups()** when the child has not been configured. The return status should be **EFI_NOT_STARTED.** 3. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.4.2	0x60fffa21, 0x3c10, 0x427a, 0xaf, 0x6e, 0xee, 0x78, 0x39, 0x14, 0xc5, 0xbe	**EFI_MANAGED_NETWORK_PROTOCOL.Groups** – invokes **Groups()** with the parameter *JoinFlag* set to **TRUE** and a *MacAddress* value of **NULL**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Groups()** with the parameter *JoinFlag* set to **TRUE** and a *MacAddress* value of **NULL.** The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.4.3	0x8e49561e, 0x667b, 0x4da2, 0xae, 0x57, 0xa3, 0x51, 0x07, 0xaa, 0xb0, 0xce	**EFI_MANAGED_NETWORK_PROTOCOL.Groups** – invokes **Groups()** with a **MacAddress* value that is an invalid multicast MAC address.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Groups()** with a **MacAddress* value that is an invalid multicast MAC address. The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.4.4	0xbf473ce1, 0x8bf5, 0x4386, 0x81, 0x3b, 0x73, 0x34, 0xff, 0xc1, 0x8b, 0xb2	`EFI_MANAGED_NETWORK_PROTOCOL.Groups` – invokes `Groups()` when the supplied multicast group has already been joined.	1. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new MNP child. 2. Call `EFI_MANAGED_NETWORK_PROTOCOL.Configure()` to configure the parameters for the new child. 3. Call `EFI_MANAGED_NETWORK_PROTOCOL.Groups()` to join a multicast group. The return status should be `EFI_SUCCESS.` 4. Call `EFI_MANAGED_NETWORK_PROTOCOL.Groups()` to join the same multicast group joined in step 3. The return status should be `EFI_ALREADY_STARTED.` 5. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created MNP child and clean up the environment.
5.23.1.4.5	0x0ea6fd9b, 0xb4d3, 0x46d0, 0xa9, 0xb5, 0xe3, 0x41, 0x8f, 0x76, 0x59, 0x9e	`EFI_MANAGED_NETWORK_PROTOCOL.Groups` – invokes `Groups()` to remove a multicast group that has not been joined.	1. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new MNP child. 2. Call `EFI_MANAGED_NETWORK_PROTOCOL.Configure()` to configure the parameters for the new child. 3. Call `EFI_MANAGED_NETWORK_PROTOCOL.Groups()` to remove a multicast group that has not been joined. The return status should be `EFI_NOT_FOUND`. 4. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.4.6	0x10e81796, 0x75df, 0x4998, 0x95, 0x3b, 0xf6, 0x6a, 0x73, 0x65, 0xa6, 0xdf	EFI_MANAGED_NETWORK_PROTOCOL.Groups – invokes Groups() to join a multicast group.	1. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new MNP child. 2. Call EFI_MANAGED_NETWORK_PROTOCOL.Configure() to configure the parameters for the new child. 3. Call EFI_MANAGED_NETWORK_PROTOCOL.Groups () to join a multicast group. The return status should be EFI_SUCCESS. 4. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created MNP child and clean up the environment.
5.23.1.4.7	0x86d023ea, 0xcd2a, 0x4641, 0x82, 0x38, 0x19, 0x4c, 0x5e, 0x1c, 0x72, 0x07	EFI_MANAGED_NETWORK_PROTOCOL.Groups – invokes Groups() to delete a multicast group.	1. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new MNP child. 2. Call EFI_MANAGED_NETWORK_PROTOCOL.Configure() to configure the parameters for the new child. 3. Call EFI_MANAGED_NETWORK_PROTOCOL.Groups() to delete the multicast group. The return status should be EFI_SUCCESS. 4. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.4.8	0x28419ce8, 0xe2d3, 0x4434, 0x90, 0xd3, 0xc2, 0xe3, 0xb5, 0x34, 0x50, 0x52	**EFI_MANAGED_NETWORK_PROTOCOL.Groups** – invokes **Groups()** to delete all groups.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **Groups()** to delete all groups. The return status should be **EFI_SUCCESS**. 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

18.1.5 Transmit()

Number	GUID	Assertion	Test Description
5.23.1.5.1	0x5ae0ea70, 0x50d7, 0x49ab, 0xb7, 0x78, 0xb9, 0x12, 0xa9, 0xab, 0x5b, 0x91	**EFI_MANAGED_NETWORK_PROTOCOL.Transmit** – invokes **Transmit()** with a *Token* value of **NULL**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Transmit()** with a *Token* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.5.2	0x254e59ae, 0x6184, 0x4885, 0x84, 0x9d, 0xd9, 0x96, 0x75, 0x12, 0xd2, 0x5f	**EFI_MANAGED_NETWORK_PROTOCOL.Transmit** – invokes **Transmit()** with a *Token.Event* value of **NULL**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Transmit()** with a *Token.Event* value of **NULL.** The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.5.3	0xbcf56099, 0x84e9, 0x464b, 0xb8, 0x50, 0x64, 0x26, 0x5f, 0x91, 0x69, 0x6b	**EFI_MANAGED_NETWORK_PROTOCOL.Transmit** – invokes **Transmit()** with a *TxData.FragmentCount* value of 0.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Transmit()** with a *TxData.FragmentCount* value of **0**. The rerurn status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.5.4	0x8612aa9b, 0x2c0d, 0x4512, 0xbf, 0xf9, 0xfd, 0x70, 0xae, 0x62, 0xaf, 0xfa	**EFI_MANAGED_NETWORK_PROTOCOL.Transmit –** invokes **Transmit()** when *(Token.TxData.HeaderLength + Token.TxData.DataLength)* is not equal to the sum of the *Token.TxData.FragmentTable[].FragmentLength* fields.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Transmit()** when *(Token.TxData.HeaderLength + Token.TxData.DataLength)* is not equal to the sum of the *Token.TxData.FragmentTable[].FragmentLength* fields. The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.5.5	0xab47d163, 0x05ef, 0x4aac, 0xaa, 0x45, 0xae, 0x93, 0x8e, 0xf8, 0x25, 0x95	**EFI_MANAGED_NETWORK_PROTOCOL.Transmit –** invokes **Transmit()** with one or more *Token.TxData.FragmentTable[].FragmentLength* fields with values of **0**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Transmit()** with one or more *Token.TxData.FragmentTable[].FragmentLength* fields with values of **0**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.5.6	0x8030770d, 0x056a, 0x4780, 0x98, 0xbe, 0xef, 0x85, 0x46, 0x7f, 0xb2, 0xec	**EFI_MANAGED_NETWORK_PROTOCOL.Transmit** – invokes **Transmit()** with one or more *Token.TxData.FragmentTable[].FragmentBuffer* fields with values of **NULL**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Transmit()** with one or more *Token.TxData.FragmentTable[].FragmentBuffer* fields with values of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.5.7	0xcd7bf7fb, 0xf3be, 0x4cd7, 0x8a, 0xc3, 0x50, 0x2d, 0xca, 0xe5, 0xcc, 0x5a	**EFI_MANAGED_NETWORK_PROTOCOL.Transmit** – invokes **Transmit()** when the MNP child driver instance has not been configured.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Transmit()** when the MNP child driver instance has has not been configured. The return status should be **EFI_NOT_STARTED.** 3. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.5.8	0x5f54752c, 0xa297, 0x4609, 0x9b, 0x4b, 0x44, 0x77, 0x45, 0x04, 0x18, 0x2d	`EFI_MANAGED_NETWORK_PROTOCOL.Transmit` – invokes `Transmit()` with transmit specified data to check the correction of data transmission.	1. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new MNP child. 2. Call `EFI_MANAGED_NETWORK_PROTOCOL.Configure()` to configure the parameters for the new child. 3. Call `EFI_MANAGED_NETWORK_PROTOCOL.Transmit()` with transmit data specified. The return status should be `EFI_SUCCESS`. 4. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created MNP child and clean up the environment.
5.23.1.5.9	0x54a2a21b, 0x9acf, 0x4f61, 0x9c, 0xc1, 0x8e, 0x31, 0xa8, 0x3e, 0x9e, 0xc4	`EFI_MANAGED_NETWORK_PROTOCOL.Transmit` – invokes `Transmit()` with transmit data not specified.	1. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new MNP child. 2. Call `EFI_MANAGED_NETWORK_PROTOCOL.Transmit()` with transmit data not specified. The return status should be `EFI_SUCCESS`. 3. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created MNP child and clean up the environment.

18.1.6 Receive()

Number	GUID	Assertion	Test Description
5.23.1.6.1	0xf88f8d45, 0xedd2, 0x4adc, 0xb9, 0xd1, 0x8b, 0xec, 0x49, 0x25, 0xc5, 0x35	`EFI_MANAGED_NETWORK_PROTOCOL.Receive` – invokes `Receive()` when the child has not been configured.	1. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new MNP child. 2. Call `EFI_MANAGED_NETWORK_PROTOCOL.Receive()` when the child has not been configured. The return status should be `EFI_NOT_STARTED.` 3. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created MNP child and clean up the environment.
5.23.1.6.2	0xe0605ca4, 0x21d1, 0x4692, 0xa4, 0xcc, 0x90, 0x5f, 0xbe, 0xb0, 0xa9, 0xb5	`EFI_MANAGED_NETWORK_PROTOCOL.Receive` – invokes `Receive()` when the receive completion token is already in the receive queue.	1. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new MNP child. 2. Call `EFI_MANAGED_NETWORK_PROTOCOL.Configure()` to configure the parameters for the new MNP child. 3. Call `EFI_MANAGED_NETWORK_PROTOCOL.Receive()` to place the token into the receiving queue. 4. Call `EFI_MANAGED_NETWORK_PROTOCOL.Receive()` to receive the token which was placed in the receiving queue in step 3. The rerurn status should be `EFI_ACCESS_DENIED.` 5. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.6.3	0x9349ff52, 0x8bfb, 0x4018, 0xa8, 0x5a, 0x41, 0x71, 0xb8, 0x36, 0x9f, 0x28	**EFI_MANAGED_NETWORK_PROTOCOL.Receive** – invokes **Receive()** with a *Token* value of **NULL**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new MNP child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Receive()** with a *Token* value of **NULL.** The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.6.4	0xfdb1c2d3, 0xcc35, 0x4bc7, 0xac, 0xa6, 0x6d, 0x0f, 0xda, 0x79, 0x85, 0x55	**EFI_MANAGED_NETWORK_PROTOCOL.Receive** – invokes **Receive()** with a *Token.Event* value of **NULL**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new MNP child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Receive()** with a *Token.Event* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.6.5	0x23fb0e81, 0xe831, 0x40fa, 0x8c, 0xc9, 0xc4, 0x10, 0x2f, 0x7d, 0x8f, 0xdc	`EFI_MANAGED_NETWORK_PROTOCOL.Receive` – invokes `Receive()` to place an asynchronous receiving request into the receiving queue.	1. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new MNP child. 2. Call `EFI_MANAGED_NETWORK_PROTOCOL.Configure()` to configure the parameters for the new MNP child. 3. Call `EFI_MANAGED_NETWORK_PROTOCOL.Receive()` to place an asynchronous receiving request into the receiving queue. The return status should be `EFI_SUCCESS`. 4. Verify that the received data is correct. 5. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created MNP child and clean up the environment.
5.23.1.6.6	0x2c0e86ce, 0xec73, 0x4840, 0x9c, 0x07, 0xb5, 0xf1, 0x75, 0xc6, 0x81, 0x79	`EFI_MANAGED_NETWORK_PROTOCOL.Receive` – invokes `Cancel()` to abort the receive	1. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new MNP child. 2. Call `EFI_MANAGED_NETWORK_PROTOCOL.Configure()` to configure the parameters for the new MNP child. 3. Call `EFI_MANAGED_NETWORK_PROTOCOL.Receive()` to place an asynchronous receiving request into the receiving queue. The return status should be `EFI_SUCCESS`. 4. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.Cancel()` to abort the receive. The return status should be `EFI_SUCCESS`. 5. Call `EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.6.7	0x4e364693, 0xe0c7, 0x49d3, 0xa0, 0xe5, 0xb8, 0x43, 0xd4, 0x79, 0x84, 0xe6	EFI_MANAGED_NETWORK_PROTOCOL.Receive – invokes Receive() to place an asynchronous receiving request into the receiving queue.	1. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new MNP child. 2. Call EFI_MANAGED_NETWORK_PROTOCOL.Configure() to configure the parameters for the new MNP child. 3. Call EFI_MANAGED_NETWORK_PROTOCOL.Receive() to place an asynchronous receiving request into the receiving queue. The return status should be EFI_SUCCESS. 4. Verify source MAC address correction. 5. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created MNP child and clean up the environment.

18.1.7 Cancel()

Number	GUID	Assertion	Test Description
5.23.1.7.1	0xf8c7e036, 0xfb8e, 0x4fbb, 0x94, 0x7c, 0x1c, 0x72, 0x75, 0xf5, 0xb9, 0x1f	EFI_MANAGED_NETWORK_PROTOCOL.Cancle – invokes Cancel() when the child has not been configured.	1. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new MNP child. 2. Call EFI_MANAGED_NETWORK_PROTOCOL.Cancel() when the child has not been configured. The return status should be EFI_NOT_STARTED. 3. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.7.2	0x36ca4137, 0x5272, 0x469b, 0xad, 0x35, 0xba, 0xb4, 0x25, 0xb6, 0x4c, 0x27	**EFI_MANAGED_NETWORK_PROTOCOL.Cancle** – invokes **Cancel()** when the value of the *Token* parameter is not **NULL** but the asynchronous I/O request was not found in the transmit or receive queues.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new MNP child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Cancel()** when the value of the *Token* parameter is not **NULL** but the asynchronous I/O request was not found in the transmit or receive queues. The return status should be **EFI_NOT_FOUND**. 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.7.3	0xe873ef06, 0x2a4c, 0x4679, 0xa3, 0xf8, 0xd1, 0x02, 0x17, 0x1c, 0x11, 0xeb	**EFI_MANAGED_NETWORK_PROTOCOL.Cancle –** invokes **Cancel()** when the value of the *Token* parameter is **NULL.**	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new MNP child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Cancel()** when the value of the *Token* parameter is **NULL.** The return status should be **EFI_SUCCESS.** 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

Number	GUID	Assertion	Test Description
5.23.1.7.4	0x21288fe0, 0x7c33, 0x423c, 0xaa, 0xd7, 0x95, 0x79, 0xa7, 0xec, 0xc6, 0x04	**EFI_MANAGED_NETWORK_PROTOCOL.Cancle –** invokes **Cancel()** to abort an asynchronous transmit or receive request.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new MNP child. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Receive()** to place a asynchronous request into the receive queue. 4. Call **EFI_MANAGED_NETWORK_PROTOCOL.Cancel()** to abort an asynchronous transmit or receive request. The return status should be **EFI_SUCCESS**. 5. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

18.1.8 Poll()

Number	GUID	Assertion	Test Description
5.23.1.8.1	0xf87f9d7f, 0xbe91, 0x4b28, 0xb6, 0x8d, 0x49, 0x4e, 0x28, 0x18, 0x07, 0xca	**EFI_MANAGED_NETWORK_PROTOCOL.Poll –** invokes **Poll()** when the child has not been configured.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_PROTOCOL.Poll()** when the child has not been configured. The return status should be **EFI_NOT_STARTED.** 3. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

18.1.9 CreateChild()

Number	GUID	Assertion	Test Description
5.23.1.9.1	0x026c7391, 0x7ebe, 0x4715, 0xba, 0xe4, 0xc5, 0x1b, 0x2e, 0x9a, 0x99, 0xf4	**EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild – ** invokes CreateChild() with a *ChildHandle* value of **NULL**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** with a *ChildHandle* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER.**
5.23.1.9.2	0x48b5ff0b, 0xd688, 0x4644, 0x86, 0x62, 0xa9, 0x63, 0x6f, 0x2f, 0x4c, 0x1c	**EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild – ** invokes CreateChild() with a *ChildHandle* value of **NULL**.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()** with a *ChildHandle* value of **NULL**. The return status should be **EFI_SUCCESS**. 2. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.
5.23.1.9.3	0x27da9434, 0x20fa, 0x42af, 0x8b, 0xdf, 0x87, 0x8e, 0xc9, 0x8b, 0x3b, 0xb9	**EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild – ** invokes CreateChild() when the *ChildHandle* value is an existing instance handle.	1. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()**with valid parameter to create a new MNP child. 2. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()**with the parameter *ChildHandle* pointing to the handle created in step 1. The return status should be **EFI_INVALID_PARAMETE**. 3. Call **EFI_MANAGED_NETWORK_PROTOCOL.Configure()** to configure the parameters for the new child. 4. Call **EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created MNP child and clean up the environment.

18.1.10 DestroyChild()

Number	GUID	Assertion	Test Description
5.23.1.10.1	0xc400df8b, 0x61d0, 0x4244, 0xb2, 0xec, 0xed, 0x2f, 0xc6, 0x54, 0x8c, 0x7e	EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild – invokes DestroyChild() when the child does not exist.	1.Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild() when the parameter *ChildHandle* is NULL. The return status should be EFI_INVALID_PARAMETER.
5.23.1.10.2	0x9ed9c819, 0x95fc, 0x4b00, 0x99, 0x7c, 0x36, 0x20, 0xfa, 0x9f, 0xad, 0xb3	EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild – invokes DestroyChild() to destroy an existing child.	1. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()to create a new MNP child. 2. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the child handle created in step 1. The return status should be EFI_SUCCESS.
5.23.1.10.3	0x8182f56c, 0x3fe6, 0x4583, 0x9b, 0xb7, 0xfd, 0x8a, 0xe2, 0x1b, 0xe6, 0xac	EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild – invokes DestroyChild()twice to destroy one child handle created before.	1. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.CreateChild()to create a new MNP child. 2. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the child handle created in step 1. The return status should be EFI_SUCCESS. 2. Call EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the child handle created in step 1 again. The return status should be EFI_UNSUPPORTED.

19 EFI Byte Code Virtual Machine Test

19.1 EFI_EBC_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_EBC_PROTOCOL Section.

19.1.1 CreateThunk()

Number	GUID	Assertion	Test Description
5.15.1.1.1	0x5de39abd, 0xe9d4, 0x4fee, 0xb4, 0xdd, 0x31, 0x73, 0xb7, 0x35, 0xe3, 0x20	**EFI_EBC_PROTOCOL.CreateThunk** - Calling **CreateThunk()** with an invalid Parameters returns **EFI_INVALID_PARAMETER**.	Call **CreateThunk()** when the EBC image entry point is not 2-byte aligned. The return code should be **EFI_INVALID_PARAMETER**.
5.15.1.1.2	0x6f19a253, 0xc6ff, 0x41a3, 0xa5, 0x8b, 0xa4, 0x57, 0x16, 0xe1, 0x2f, 0x4c	**EFI_EBC_PROTOCOL.CreateThunk** - Calling **CreateThunk()** to create ebc thunk returns **EFI_SUCCESS**.	Call **CreateThunk()** to create thunk for the EBC image. The return code should be **EFI_SUCCESS**.
5.15.1.1.3	0xcabc5c1e, 0x75a0, 0x4349, 0xab, 0xd8, 0x41, 0x17, 0x7b, 0x25, 0x9e, 0x8a	**EFI_EBC_PROTOCOL.CreateThunk** – Calling **CreateThunk()** invokes the Ebc entry point.	Call **CreateThunk()** to create thunk for the EBC image and invokes the thunk. The entry point of EBC image must be invoked.

19.1.2 UnloadImage()

Number	GUID	Assertion	Test Description
5.15.1.2.1	0x99c53b53, 0x0998, 0x4fda, 0xaa, 0x4e, 0x9c, 0xc4, 0x9a, 0x1c, 0x8a, 0x19	EFI_EBC_PROTOCOL.UnloadImage - Calling UnloadImage() with an invalid Parameters returns EFI_INVALID_PARAMETER.	Call UnloadImage() when the image handle is not recognized as belonging to an EBC image that has been executed. The return code should be EFI_INVALID_PARAMETER.
5.15.1.2.2	0xecea2853, 0xe14e, 0x493b, 0x9a, 0xb3, 0xcd, 0xa4, 0xc8, 0x32, 0x2c, 0x3e	EFI_EBC_PROTOCOL.UnloadImage - Calling UnloadImage() unloads ebc thunk.	Call UnloadImage() to unload the EBC image from memory. The return code should be EFI_SUCCESS.

19.1.3 RegisterICacheFlush()

Number	GUID	Assertion	Test Description
5.15.1.3.1	0xf362b36f, 0x819d, 0x45a4, 0xa5, 0xc7, 0xa0, 0x0a, 0x81, 0x2b, 0xf3, 0x5f	EFI_EBC_PROTOCOL.RegisterICacheFlush - Calling RegisterICacheFlush() registers an ebc callback function.	Call RegisterICacheFlush() to register a callback function. The return code should be EFI_SUCCESS.
5.15.1.3.2	0x26480c1d, 0xac79, 0x46e5, 0xa4, 0xff, 0xec, 0x3e, 0xd5, 0x99, 0x87, 0xec	EFI_EBC_PROTOCOL.RegisterICacheFlush - Callback function is invoked after calling CreateThunk().	1. Call RegisterICacheFlush() to register a callback function. 2. Call CreateThunk() to create thunk for an EBC image. The callback function should be invoked.

19.1.4 GetVersion()

Number	GUID	Assertion	Test Description
5.15.1.4.1	0xce787a92, 0x1ee8, 0x4f65, 0xb7, 0x7c, 0xb4, 0xcd, 0xcf, 0xcd, 0xd3, 0xf2	**EFI_EBC_PROTOCOL.GetVersion** - Calling **GetVersion()** when version pointer is **NULL** and returns **EFI_INVALID_PARAMETER**.	Call **GetVersion()** when version pointer is **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.15.1.4.2	0x57100f81, 0xe05a, 0x4abf, 0x93, 0xc2, 0x49, 0x1c, 0xf8, 0xd4, 0xb6, 0x7c	**EFI_EBC_PROTOCOL.GetVersion** - Calling **GetVersion()** to get ebc interpreter version returns **EFI_SUCCESS**.	Call **GetVersion()** to get the version of the EBC interpreter. The return code should be **EFI_SUCCESS**.

20 Network Protocols ARP and DHCP

20.1 EFI_ARP_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_ARP_PROTOCOL Section.

20.1.1 Add()

Number	GUID	Assertion	Test Description
5.24.1.1.1	0xf6fa3bd8, 0xd8d0, 0x4c54, 0x88, 0xc2, 0x1f, 0xcf, 0x27, 0x62, 0xc5, 0xd4	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_INVALID_PARAMETER** with both the *DenyFlag* and *TargetAddress* value of **NULL**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** with both the *DenyFlag* and *TargetAddress* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.1.2	0x6404caf6, 0x9020, 0x4272, 0xa2, 0x79, 0x6f, 0x53, 0x8d, 0x42, 0x5c, 0x35	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_INVALID_PARAMETER** with a *DenyFlag* value of **FALSE** and the *TargetHwAddress /TargetSwAddress* value of **NULL**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **FALSE** and the *TargetHwAddress / TargetSwAddress* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.3	0x138858cd, 0x40fe, 0x4b05, 0xb4, 0x8c, 0xb5, 0x9f, 0xf2, 0xfd, 0xee, 0x5e	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_INVALID_PARAMETER** with a *DenyFlag* value of **FALSE** and a *TargetHwAddress* value of **NULL**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** with a **DenyFlag** value of **FALSE** and a *TargetHwAddress* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.1.4	0x48a946f4, 0x8ff7, 0x4b50, 0xa1, 0xb2, 0xc6, 0x82, 0xcd, 0xa5, 0x78, 0x62	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_INVALID_PARAMETER** with a *DenyFlag* value of **FALSE** and a *TargetSwAddress* value of **NULL**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **FALSE** and a *TargetSwAddress* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.5	0x32deb7c7, 0x9e67, 0x459f, 0xbf, 0x4c, 0xbc, 0x80, 0x33, 0x31, 0x36, 0x05	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_INVALID_PARAMETER** with a *DenyFlag* value of **TRUE** and both *TargetHwAddress* and *TargetSwAddress* value of **NULL**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** when *DenyFlag* is **TRUE** and both *TargetHwAddress* and *TargetSwAddress* are not **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.1.6	0x87d47f39, 0x8d82, 0x40c4, 0xb9, 0x36, 0x2c, 0xf5, 0x8b, 0xa2, 0xd9, 0x32	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_ACCESS_DENIED** when the ARP cache entry of the same *TargetSwAddress* already exists and *Overwrite* is **FALSE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **FALSE** and with valid *TargetSwAddress* / *TargetHwAddress* values. 4. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **TRUE** and with the same *TargetSwAddress* as the one used in the last call while *Overwrite* is **FALSE**. The return status must be **EFI_ACCESS_DENIED.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.7	0xecc2942f, 0xd23e, 0x421e, 0x8a, 0x31, 0x3c, 0xe2, 0xdf, 0xee, 0x82, 0xcb	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_ACCESS_DENIED** when the ARP cache entry of the same *TargetHwAddress* already exists and Overwrite is **FALSE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **FALSE** and with valid *TargetSwAddress* / *TargetHwAddress* values. 4. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **TRUE** and with the same *TargetHwAddress* as the one used in the last call while Overwrite is **FALSE**. The return status must be **EFI_ACCESS_DENIED.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.1.8	0x31b66402, 0x4c9a, 0x486f, 0x9e, 0x68, 0xf5, 0xb1, 0x8b, 0x7b, 0xb4, 0xbf	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_ACCESS_DENIED** when the ARP cache entry of the same *TargetHwAddress* already exists and *Overwrite* is **FALSE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **FALSE** and with valid *TargetSwAddress/ TargetHwAddress* . 4. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **FALSE** and with the same *TargetSwAddress/ TargetHwAddress* as the ones used in the last call while Overwrite is **FALSE**. The return status must be **EFI_ACCESS_DENIED.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.9	0x14c76af4, 0x29ca, 0x4018, 0x85, 0x6d, 0xfb, 0xfa, 0xfb, 0xae, 0x02, 0xa6	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_NOT_STARTED** when the ARP driver instance has not been configured and *TargetHwAddress* is valid, while *DenyFlag* is **TRUE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **TRUE** and with valid *TargetHwAddress* . The return status must be **EFI_NOT_STARTED.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.1.10	0x8f07a21d, 0xfca8, 0x4d4a, 0xa7, 0x18, 0xaf, 0x80, 0x27, 0x46, 0x84, 0x40	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_NOT_STARTED** when the ARP driver instance has not been configured and *TargetSwAddress* is valid, while *DenyFlag* is **TRUE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **TRUE** and a valid *TargetSwAddress* value. The return status must be **EFI_NOT_STARTED.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.1.11	0xf7e1b57e, 0x8499, 0x49b7, 0xa1, 0x35, 0xe0, 0x25, 0x7a, 0x68, 0x7c, 0xca	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_NOT_STARTED** when the ARP driver instance has not been configured and *TargetSwAddress/ TargetHwAddress* are valid, while *DenyFlag* is **FALSE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **FALSE** and with valid *TargetSwAddress*/ *TargetHwAddress* . The return status must be **EFI_NOT_STARTED.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.12	0x203039cb, 0xbfce, 0x472f, 0x9d, 0x46, 0xfe, 0x53, 0xcd, 0x47, 0x42, 0xb6	**`EFI_ARP_PROTOCOL.Add()`** - returns **`EFI_SUCCESS`** when Adding normal entry.	1. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Arp child handle. 2. Call **`EFI_ARP_PROTOCOL.Configure()`** with all valid parameters. 3. Call **`EFI_ARP_PROTOCOL.Add()`** with a *`DenyFlag`* value of **`FALSE`** and with valid *`TargetSwAddress`*/ *`TargetHwAddress`* . The return status must be **`EFI_SUCCESS.`** 4. Call **`EFI_ARP_PROTOCOL.Request()`** with the same *`TargetSwAddress`* as the one added. 5. Call **`EFI_ARP_PROTOCOL.Request()`** with the *`TargetHwAddress`* added into the entry cache, and compare the *`TargetHwAddress`* brought back by it, then verify if they are the same. 6. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.13	0x7e93dc4e, 0x2731, 0x41d4, 0x96, 0x89, 0x27, 0x3a, 0xfe, 0xdc, 0x26, 0x40	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_SUCCESS** When overwrite is **TRUE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **FALSE** and with valid *TargetSwAddress*/ *TargetHwAddress* . 4. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **FALSE** and with the same *TargetSwAddress* as the one used in the last call and another different *TargetHwAddress* , while overwrite is **TRUE**. The return status must be **EFI_SUCCESS.** 5. Call **EFI_ARP_PROTOCOL.Request()** with the same *TargetSwAddress* as the one added. 6. Call **EFI_ARP_PROTOCOL.Request()** with the *TargetHwAddress* added at the second time, and compare the *TargetHwAddress* brought back by it, then verify if they are the same. 7. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.14	0xa00cc3c8, 0x005c, 0x4aed, 0xa1, 0x5c, 0x3e, 0x91, 0xca, 0x56, 0x33, 0xe5	**EFI_ARP_PROTOCOL .Add()** - returns **EFI_SUCCESS** when adding normal entry with Timeout set.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** with a *DenyFlag* value of **FALSE** and with valid *TargetSwAddress*/ *TargetHwAddress* . 4. Call **EFI_ARP_PROTOCOL.Add()** to overwrite the exist entry with TimeoutValue set to be 50 seconds. The return status must be **EFI_SUCCESS.** 5. Call **EFI_ARP_PROTOCOL.Request()** with the same *TargetSwAddress* as the same one added. 6. Call **EFI_ARP_PROTOCOL.Request()** with the *TargetHwAddress* added at the second time, and compare the *TargetHwAddress* brought back, then verify if they are the same. 7. Stall 30 seconds and then call **EFI_ARP_PROTOCOL.Request()** and verify if the Address is correct again. 8. Stall 20 seconds to let entry timeout, then call **EFI_ARP_PROTOCOL.Request(),** and now the return status must be EFI_NOT_READY. 9. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.15	0x46eee5b0, 0x7a16, 0x4be3, 0x87, 0x9e, 0xb6, 0x4f, 0xaa, 0xd0, 0xc0, 0x65	`EFI_ARP_PROTOCOL.Add()` - returns `EFI_SUCCESS` when adding normal entry after the request call.	1. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Arp child handle. 2. Call `EFI_ARP_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_ARP_PROTOCOL.Request()` with valid *`TargetSwAddress`*. The return status must be `EFI_NOT_READY`. 4. Call `EFI_ARP_PROTOCOL.Add()` to add a normal entry with the same *`TargetSwAddress`* as the one used in `Request().` The return status must be `EFI_SUCCESS.` 5. Call `EFI_ARP_PROTOCOL.Request()` with the *`TargetHwAddress`* added, and compare the *`TargetHwAddress`* brought back by it, then verify if they are the same. 6. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.16	0x01321dca, 0xe8d4, 0x4022, 0xb7, 0xa1, 0xd6, 0x69, 0xca, 0xcb, 0x52, 0x0b	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_SUCCESS** when adding denied entry.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a deny entry with the valid *TargetSwAddress*. The return status must be **EFI_SUCCESS.** 4. Call **EFI_ARP_PROTOCOL.Request()** with the same *TargetSwAddress* as the one used in the last call. The return status must be **EFI_ACCESS_DENIED**. 5. Call **EFI_ARP_PROTOCOL.Request()** with a *TargetHwAddress* value of “0.0.0.0.0.0”, and compare the *TargetHwAddress* brought back by it, then verify if they are the same. 6. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.17	0x7856bfd5, 0x758a, 0x4bcf, 0x9d, 0xc9, 0x2e, 0x36, 0x9a, 0xea, 0xf7, 0xdf	**`EFI_ARP_PROTOCOL.Add()`** - returns **`EFI_SUCCESS`** when adding denied entry with a overwrite value of **`TRUE`**.	1. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Arp child handle. 2. Call **`EFI_ARP_PROTOCOL.Configure()`** with all valid parameters. 3. Call **`EFI_ARP_PROTOCOL.Add()`** to add a deny entry with the valid *`TargetHwAddress`* (0:2:3:4:5:6) and overwrite value of **`TRUE`**. The return status must be **`EFI_SUCCESS.`** 4. Call **`EFI_ARP_PROTOCOL.Request()`** with valid *`TargetSwAddress`* (172.16.210.161). The OS side should capture the request packet sent from the EUT side. 5. If captured, the OS side configures the ARP reply packet with source IP "172.16.210.161", source Mac "0:2:3:4:5:6". Then send the packet back to EUT side. 6. Then the OS side configures another ARP reply packet with source IP "172.16.210.161", source Mac "0:2:3:4:5:7". Then sends the second packet back to EUT side. The return status must be **`EFI_ACCESS_DENIED`**. 7. Call **`EFI_ARP_PROTOCOL.Request()`** with a *`TargetHwAddress`* value of "0:2:3:4:5:7" and compare the *`TargetHwAddress`* brought back by it, then verify if they are the same. 8. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.18	0xefcdb906, 0xa43a, 0x437f, 0x81, 0x35, 0xe0, 0xef, 0xea, 0xd3, 0xdc, 0x0a	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_SUCCESS** – Add denied entry with overwrite is **TRUE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry with the valid *TargetSwAddress*”172.16.210.161” and *TargetHwAddress* “0:2:3:4:5:6”. 4. Call **EFI_ARP_PROTOCOL.Add()** to overwrite the existed entry with a deny entry and the *TargetHwAddress* is still “0:2: 3:4:5:6”. The return status must be **EFI_SUCCESS.** 5. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* (172.16.210.161). The OS side should capture the request packet sent from the EUT side. 6. If having captured, the OS side configures the ARP reply packet with sender IP “172.16.210.161”, sender Mac “0:2:3:4:5:6”. Then send the packet back to EUT side. 7. Then the OS side configures another ARP reply packet with sender IP “172.16.210.161”, sender Mac “0:2:3:4:5:7”. Then send the second packet back to EUT side. The return status must be **EFI_ACCESS_DENIED**. 8. Compare the *TargetHwAddress* brought back by **EFI_ARP_PROTOCOL.Request()** with “0:2:3:4:5:7” and verify if they are same. 8. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.19	0xccf3f6de, 0x5d43, 0x4dfa, 0xbe, 0x65, 0xe8, 0xc5, 0x3d, 0xe0, 0xdf, 0x95	**EFI_ARP_PROTOCOL .Add()** – returns **EFI_SUCCESS** when adding denied entry with overwrite value of **TRUE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry with the valid *TargetSwAddress*"172.16.210.161" and *TargetHwAddress* "0:2:3:4:5:6". 4. Call **EFI_ARP_PROTOCOL.Add()** to overwrite the existed entry with a deny entry and the *TargetHwAddress* is still "0:2: 3:4:5:6". The return status must be **EFI_SUCCESS.** 5. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* (172.16.210.161). The OS side should capture the request packet sent from the EUT side. 6. If having captured, the OS side configures the ARP reply packet with sender IP "172.16.210.161", sender Mac "0:2:3:4:5:6". Then send the packet back to EUT side. 7. Then the OS side configures another ARP reply packet with sender IP "172.16.210.161", sender Mac "0:2:3:4:5:7". Then send the second packet back to EUT side. The return status must be **EFI_ACCESS_DENIED**. 8. Compare the *TargetHwAddress* brought back by **EFI_ARP_PROTOCOL.Request()** with "0:2:3:4:5:7" and verify if they are same. 8. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment. 9. Call **EFI_ARP_PROTOCOL.Add()** to overwrite the exist entry with a deny entry whose *TargetSwAddress* is "172.16.210.161". The return status must be **EFI_SUCCESS.**

Number	GUID	Assertion	Test Description
5.24.1.1.19 (continued)			10. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* (172.16.210.161). The return status must be **EFI_ACCESS_DENIED**.
5.24.1.1.20	0xb294d2a8, 0xb3f7, 0x4ec0, 0xa1, 0x4c, 0x74, 0xa9, 0x6d, 0xcc, 0x56, 0xb7	**EFI_ARP_PROTOCOL.Add()** - returns **EFI_SUCCESS** when adding denied entry with Timeout set.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a deny entry whose *TargetSwAddress* is "172.16.210.161" and a Timeout value of set to be 50. The return status must be **EFI_SUCCESS.** 4. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* (172.16.210.161). The return status must be **EFI_ACCESS_DENIED** and the return *TargetHwAddress* must be "0:0:0:0:0:0". 5. Stall 30 seconds, call **EFI_ARP_PROTOCOL.Request()** again with valid *TargetSwAddress*"172.16.210.161". The return status must be **EFI_ACCESS_DENIED** and the return *TargetHwAddress* must be "0:0:0:0:0:0". 6. Stall 20 seconds, call **EFI_ARP_PROTOCOL.Request()** again with valid *TargetSwAddress*"172.16.210.161".This time the return status must be **EFI_NOT_READY**. 7. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.1.21	0x48d3af46, 0x09db, 0x4c34, 0xb9, 0x1e, 0xb0, 0x48, 0xe0, 0x1a, 0x9d, 0x17	**`EFI_ARP_PROTOCOL.Add()`** - returns **`EFI_SUCCESS`** when adding denied entry.	1. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Arp child handle. 2. Call **`EFI_ARP_PROTOCOL.Configure()`** with all valid parameters. 3. Call **`EFI_ARP_PROTOCOL.Request()`** with valid *TargetSwAddress* (172.16.210.161). The return status must be **`EFI_NOT_READY.`** 4. Call **`EFI_ARP_PROTOCOL.Add()`** to add a deny entry whose *`TargetSwAddress`* is ”172.16.210.161”. The return status must be **`EFI_SUCCESS.`** 5. Verify if the return *`TargetHwAddress`* is “0:0:0:0:0:0”. 6. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created Arp child handle and clean up the environment.

20.1.2 Cancel()

Number	GUID	Assertion	Test Description
5.24.1.2.1	0x56539533, 0xee7d, 0x4e57, 0xaf, 0x89, 0x2a, 0xa7, 0x3d, 0x82, 0x36, 0x61	**EFI_ARP_PROTOCOL.Cancel()** - returns **EFI_INVALID_PARAMETER** with a *TargetSwAddress* value of invalid.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* (172.16.210.161). The return status must be **EFI_NOT_READY.** 4. Call **EFI_ARP_PROTOCOL.Cancel()** with a *TargetSwAddress* value of **NULL** and a *ResolvedEvent* value other than **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.2.2	0xe9118c8c, 0x1e0e, 0x451b, 0x8f, 0x4f, 0xd6, 0x37, 0x8b, 0x82, 0xf3, 0x6a	**EFI_ARP_PROTOCOL.Cancel()** - returns **EFI_INVALID_PARAMETER** with an invalid *Event* value.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* (172.16.210.161). The return status must be **EFI_NOT_READY**. 4. Call **EFI_ARP_PROTOCOL.Cancel()** with an *Event* value of **NULL** and a *TargetSwAddress* value of not **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.2.3	0x8b6cee26, 0x52c3, 0x45fe, 0xae, 0x7e, 0xfa, 0xa6, 0xd9, 0xc1, 0x80, 0xc7	**EFI_ARP_PROTOCOL.Cancel()** - returns **EFI_NOT_FOUND** with *Event* not found.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* (172.16.210.161). The return status must be **EFI_NOT_READY.** 4. Call **EFI_ARP_PROTOCOL.Cancel()** with valid *TargetSwAddress* while *Event* is not issued by the **EFI_ARP_PROTOCOL.Request().** The return status must be **EFI_NOT_FOUND.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.2.4	0x09e570d8, 0xdc54, 0x4458, 0xb9, 0xa3, 0x58, 0x4f, 0xeb, 0x64, 0xc0, 0xdb	**`EFI_ARP_PROTOCOL.Cancel()`** - returns **`EFI_NOT_FOUND`** with *`TargetSwAddress`* not found.	1. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Arp child handle. 2. Call **`EFI_ARP_PROTOCOL.Configure()`** with all valid parameters. 3. Call **`EFI_ARP_PROTOCOL.Request()`** with valid *`TargetSwAddress`* (172.16.210.161). The return status must be **`EFI_NOT_READY.`** 4. Call **`EFI_ARP_PROTOCOL.Cancel()`** with a *`TargetSwAddress`* value of “172.16.210.160” which is not issued by the **`EFI_ARP_PROTOCOL.Request().`** The return status must be **`EFI_NOT_FOUND.`** 5. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.2.5	0xbecb34c1, 0xbfed, 0x43c1, 0x81, 0xfe, 0xc5, 0x9f, 0x8d, 0xf4, 0xf2, 0x5a	**EFI_ARP_PROTOCOL.Cancel()** - returns **EFI_NOT_FOUND** with *TargetSwAddress* not found.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* (172.16.210.161). The return status must be **EFI_NOT_READY.** 4. Call **EFI_ARP_PROTOCOL.Cancel()** with a *TargetSwAddress* value of "172.16.210.160" which is not issued by the **EFI_ARP_PROTOCOL.Request().** The return status must be **EFI_NOT_FOUND.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.2.6	0x9511bd75, 0x971b, 0x4e14, 0xb2, 0xd1, 0x44, 0x9b, 0x2e, 0x0a, 0x90, 0x78	**EFI_ARP_PROTOCOL.Cancel()** - returns **EFI_NOT_FOUND** with both the *TargetSwAddress* and *Event* value of **NULL**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* (172.16.210.161). The return status must be **EFI_NOT_READY.** 4. Call **EFI_ARP_PROTOCOL.Cancel()** with both the *TargetSwAddress* and *Event* value of **NULL**. The return status must be **EFI_NOT_FOUND.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.2.7	0xd45a3a11, 0xf14c, 0x4dc2, 0x8d, 0x91, 0xfe, 0x0b, 0xa7, 0x14, 0xac, 0x97	**EFI_ARP_PROTOCOL.Cancel()** - returns **EFI_NOT_STARTED** when the ARP driver instance has not been configured.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* (172.16.210.161). The return status must be **EFI_NOT_READY.** 4. Call **EFI_ARP_PROTOCOL.Configure()** with a *ConfigData* value of **NULL** to reset the ARP driver instance. 5. Call **EFI_ARP_PROTOCOL.Cancel()** with valid parameters which Request () had issued. The return status must be **EFI_NOT_STARTED.** 6. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.2.8	0x1b5f4fbb, 0x0d7d, 0x4b4c, 0xad, 0x29, 0x7b, 0x8b, 0xa5, 0x3e, 0xab, 0xc6	`EFI_ARP_PROTOCOL.Cancel()` - returns `EFI_SUCCESS` when canceling request.	1. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Arp child handle. 2. Call `EFI_ARP_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_ARP_PROTOCOL.Request()` with valid *`TargetSwAddress`* (172.16.210.161). The return status must be `EFI_NOT_READY.` 4. Call `EFI_ARP_PROTOCOL.Cancel()` with parameters issued by `EFI_ARP_PROTOCOL.Request()`. The return status must be `EFI_SUCCESS.` 5. Then the OS side shouldn't capture any packet sent from the EUT side. 6. Call `EFI_ARP_PROTOCOL.Request()` again, the return status should be `EFI_NOT_READY` and the return *`TargetHwAddress`* should be "0:0:0:0:0:0". 7. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created Arp child handle and clean up the environment.

20.1.3 Configure()

Number	GUID	Assertion	Test Description
5.24.1.3.1	0xcdbd6b40, 0x3b1f, 0x4cd5, 0x8b, 0xd9, 0x33, 0x99, 0x63, 0x8e, 0x80, 0x35	**EFI_ARP_PROTOCOL.Configure() -** returns **EFI_INVALID_PARAMETER** with invalid *SwAddressLength*.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with a *SwAddressLength* value of 0. The return status must be **EFI_INVALID_PARAMETER.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.3.2	0x072fb583, 0x5885, 0x4b2e, 0x99, 0x72, 0xe7, 0x2c, 0x5b, 0xd3, 0x34, 0xd5	**EFI_ARP_PROTOCOL.Configure() -** returns **EFI_INVALID_PARAMETER** with invalid *StationAddress*.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with a *StationAddress* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.3.3	0x3a8fde87, 0x1d5d, 0x462e, 0x8e, 0x3c, 0x01, 0xec, 0x3b, 0x9f, 0xf7, 0x5b	**EFI_ARP_PROTOCOL.Configure() -** returns **EFI_ACCESS_DENIED**when the *StationAddress* is different from the one that has already registered.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Configure()** with different *StationAddress* with the one that has already registerd. The return status must be **EFI_ACCESS_DENIED.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.3.4	0x2747e156, 0xee8d, 0x4533, 0xb4, 0x63, 0xa8, 0xb0, 0x5f, 0xe0, 0x6b, 0xc1	**EFI_ARP_PROTOCOL.Configure() –** returns **EFI_ACCESS_DENIED** when the *SwAddressLength* is different from the one that has already registered.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Configure()** with a different *SwAddressLength* from the one that has already registered. The return status must be **EFI_ACCESS_DENIED.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.3.5	0x790466e9, 0x0f6e, 0x4a3d, 0xa7, 0xdb, 0x5c, 0xb5, 0x6b, 0x59, 0x01, 0xef	**EFI_ARP_PROTOCOL.Configure() –** returns **EFI_ACCESS_DENIED** when the *SwAddressLength* is different from the one that has already registered.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Configure()** with different *SwAddressLength* from the one that has already registerd. The return status must be **EFI_ACCESS_DENIED.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.3.6	0xab90d4d0, 0xa0ac, 0x44c3, 0xb7, 0x03, 0x12, 0xdd, 0x10, 0x37, 0x74, 0x1d	**`EFI_ARP_PROTOCOL.Configure()`** - returns **`EFI_ACCESS_DENIED`** when the *`SwAddressType`* is different from the one that has already registered.	1. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Arp child handle. 2. Call **`EFI_ARP_PROTOCOL.Configure()`** with valid parameters. 3. Call **`EFI_ARP_PROTOCOL.Configure()`** with different *`SwAddressType`* from the one that has already registered. The return status must be **`EFI_ACCESS_DENIED.`** 4. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.3.7	0xf41970a5, 0x733f, 0x47d4, 0x8f, 0x52, 0xd5, 0x5c, 0x86, 0xd7, 0x96, 0x9f	**`EFI_ARP_PROTOCOL.Configure()`** - returns **`EFI_ACCESS_DENIED`** when the *`SwAddressType`* is different from the one that has already registered.	1. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Arp child handle. 2. Call **`EFI_ARP_PROTOCOL.Configure()`** with valid parameters. 3. Call **`EFI_ARP_PROTOCOL.Configure()`** with different *`SwAddressType`* from the one that has already registerd. The return status must be **`EFI_ACCESS_DENIED.`** 4. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.3.8	0x8b9bcd53, 0x9a83, 0x45c0, 0x9b, 0x5f, 0xf2, 0x99, 0x2c, 0x78, 0xf8, 0x1b	**`EFI_ARP_PROTOCOL.Configure()`** - returns **`EFI_SUCCESS`** with valid parameters.	1. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Arp child handle. 2. Call **`EFI_ARP_PROTOCOL.Configure()`** with valid parameters. 3. Call **`EFI_ARP_PROTOCOL.Request()`** with *`TargetSwAddress`* “172.16.210.161”. 4. The OS side should capture the request packet, and send back the ARP reply packet filled with source IP”172.16.210.161” and source MAC “0:2:3:4:5:6”. 5. The return status must be **`EFI_NOT_READY`** and the return *`TargetHwAddress`* should be“0:2:3:4:5:6”. 6. The OS side sends a request packet to resolve IP “172.16.210.102” with the source IP”172.16.210.161” and source MAC”0:2:3:4:5:7”. 7. Then OS should capture the ARP reply packet sent from the EUT side. 8. If having captured, call **`EFI_ARP_PROTOCOL.Request()`** with *`TargetSwAddress`* “172.16.210.161”. 9. The return *`TargetHwAddress`* must be “0:2:3:4:5:7”, and The return status must be **`EFI_SUCCESS.`** 10. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.3.9	0xeee99be3, 0xa701, 0x4612, 0x98, 0x1a, 0xad, 0x8c, 0x06, 0x4a, 0xd7, 0xa5	**EFI_ARP_PROTOCOL.Configure() -** returns **EFI_SUCCESS** with valid parameters.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Configure()** with a *ConfigData* value of **NULL** to reset the ARP driver instance. 4. Call **EFI_ARP_PROTOCOL.Request()**, the return status should be **EFI_NOT_STARTED**. 5. Call **EFI_ARP_PROTOCOL.Configure()** again with valid parameters. 6. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* “172.16.210.161”. 7. The OS side should capture the request packet, and send back the ARP reply packet filled with source IP”172.16.210.161” and source MAC “0:2:3:4:5:6”. The return status must be **EFI_SUCCESS**. 8. Verify if the return *TargetHwAddress* is “0:2:3:4:5:6”. 9 Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.3.10	0x4423e5b6, 0x6f3c, 0x41c3, 0x8c, 0x50, 0xea, 0x71, 0xd8, 0x52, 0x3b, 0x74	**EFI_ARP_PROTOCOL.Configure()** – returns **EFI_SUCCESS** with parameter timeout set.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters with timeout set to be 50. 3. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* "172.16.210.161". 4. The OS side should capture the request packet, and send back the ARP reply packet filled with source IP"172.16.210.161" and source MAC "0:2:3:4:5:6". The return status must be **EFI_SUCCESS**. 5. Verify if the return *TargetHwAddress* is "0:2:3:4:5:6". 6. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* "172.16.210.161" again. 7. The return status should be **EFI_SUCCESS** and the *TargetHwAddress* be "0:2:3:4:5:6". 8. Stall 30 seconds, call **EFI_ARP_PROTOCOL.Request()** like the step 6 again. 9. The return status should be **EFI_SUCCESS** and the *TargetHwAddress* be "0:2:3:4:5:6". 10. Stall 20 seconds, call **EFI_ARP_PROTOCOL.Request()** like the step 6 again. 11. This time the return status should be **EFI_NOT_READY** and the *TargetHwAddress* be "0:0:0:0:0:0". 12. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.3.11	0x79f9aacd, 0xfb79, 0x4746, 0x8f, 0x5c, 0x38, 0x4b, 0xf9, 0x2e, 0x0a, 0x53	**EFI_ARP_PROTOCOL.Configure() –** returns **EFI_SUCCESS** and packet count is correct when *ConfigData.RetryCount* is **5**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with The return status must be **EFI_SUCCESS**. 3. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* “172.16.210.161”. 4. The OS side should capture the request packet for 5 times. The return status should be **EFI_NOT_READY** and the *TargetHwAddress* should be “0:0:0:0:0:0”. 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.3.12	0x970634b0, 0x57a5, 0x40c5, 0x92, 0x01, 0xcb, 0xb2, 0x00, 0x8c, 0xbb, 0x43	**EFI_ARP_PROTOCOL.Configure() –** returns **EFI_SUCCESS** with valid parameters.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with *EntryTimeOut*, *RetryCount*, *RetryTimeOut* value of 0. 3. Call **EFI_ARP_PROTOCOL.Configure()** with a *EntryTimeOut* value of 5000000, a *RetryCount* value of 30, and a *RetryTimeOut* value of 5000000. 4. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* “172.16.210.161”. 4. The OS side should capture the request packet. 5. If having captured, the OS side sends an ARP reply back with source IP “172.16.210.161”, source MAC “0:2:3:4:5:6”. The return status must be **EFI_SUCCESS**. In addition, the *TargetHwAddress* should be “0:2:3:4:5:6”. 6. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.3.13	0xc6c2e0c3, 0x9715, 0x48a8, 0x86, 0xba, 0x36, 0xbd, 0xac, 0x70, 0x71, 0x6d	**EFI_ARP_PROTOCOL.Configure() -** returns **EFI_SUCCESS** when *SwAddressLength* is 1.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with a *SwAddressLength* value of 1. The return status must be **EFI_SUCCESS**. 3. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* “171.16.210.161”. 4. The OS side should capture the request packet. 5. If having captured, the OS side sends an ARP reply back with source IP “171”, source MAC “0:2:3:4:5:6”, Target IP “172”. The return status should be **EFI_NOT_READY** and the *TargetHwAddress* should be “0:2:3:4:5:6”. 6. The OS sends an ARP request to the broadcast address with source IP “171” and source MAC” 0:2:3:4:5:6” to resolve Target IP “172”. 7. The OS should capture the packet. 8. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.3.14	0xf4972462, 0x1dc5, 0x484f, 0xb6, 0x55, 0x8b, 0x2e, 0x89, 0xec, 0x2c, 0x46	**EFI_ARP_PROTOCOL.Configure()** – returns **EFI_SUCCESS** when *SwAddressLength* is 16.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with a *SwAddressLength* value of 16. The return status must be **EFI_SUCCESS**. 3. Call **EFI_ARP_PROTOCOL.Request()** with valid *TargetSwAddress* “171.16.210.161”. 4. The OS side should capture the request packet. 5. If having captured, the OS side sends an ARP reply back filled with source IP “172.16.210.161.0.0.0.0.0.0.0.0.0.0.0.0”, source MAC “0:2:3:4:5:6”, Target IP “172.16.210.102.0.0.0.0.0.0.0.0.0.0.0.0”. The return status should be “EFI_NOT_READY” and the *TargetHwAddress* “0:2:3:4:5:6”. 6. The OS sends an ARP request to the broadcast address with source IP “172.16.210.161.0.0.0.0.0.0.0.0.0.0.0.0” and source MAC” 0:2:3:4:5:6” to resolve Target IP “172.16.210.102.0.0.0.0.0.0.0.0.0.0.0.0”. 7. The OS should capture the packet. 8. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

20.1.4 Delete()

Number	GUID	Assertion	Test Description
5.24.1.4.1	0x1ba44874, 0x8e16, 0x422e, 0x97, 0x73, 0x43, 0x6f, 0x06, 0x2f, 0x6f, 0x01	**EFI_ARP_PROTOCOL.Delete() –** returns **EFI_NOT_FOUND** when the specified deletion key of MacAddress is not found.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Delete()** to delete the added entry with key specified as IpAddress which is the same as the *TargetSwAddress* of added entry. 5. Call **EFI_ARP_PROTOCOL.Delete()** again with key specified as MacAddress which is the same with the *TargetHwAddress* of added entry. The return status must be **EFI_NOT_FOUND.** 6. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.4.2	0xab90c68f, 0xa0af, 0x4188, 0x9a, 0x74, 0x66, 0x5a, 0x9c, 0x8a, 0x4b, 0x92	**EFI_ARP_PROTOCOL.Delete()** – returns **EFI_NOT_FOUND** when the specified deletion key of IpAddress was not found.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Delete()** to delete the added entry with key specified as MacAddress which is the same with the *TargetHwAddress* of the added entry. 5. Call **EFI_ARP_PROTOCOL.Delete()** again with key specified as IpAddress which is the same as the *TargetSwAddress* of the added entry. The return status must be **EFI_NOT_FOUND.** 6. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.4.3	0xe03b088c, 0x8cf0, 0x4db9, 0xa0, 0xc1, 0x77, 0xa9, 0xf4, 0x1a, 0xce, 0x0a	**EFI_ARP_PROTOCOL.Delete()** – returns **EFI_NOT_STARTED** when ARP driver instance has not been configured.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Delete()** with key specified as IpAddress. The return status must be **EFI_NOT_STARTED.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.4.4	0x4b8b9c7f, 0x96fc, 0x41fb, 0xbc, 0x58, 0x32, 0x1d, 0x13, 0x75, 0xed, 0x7b	`EFI_ARP_PROTOCOL.Delete()` – returns `EFI_NOT_STARTED` when ARP driver instance has not been configured.	1. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Arp child handle. 2. Call `EFI_ARP_PROTOCOL.Delete()` with key specified as MacAddress. The return status must be `EFI_NOT_STARTED`. 3. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created Arp child handle and clean up the environment.
5.24.1.4.5	0x494278d5, 0x4ff5, 0x4ac5, 0x9e, 0xd7, 0xfa, 0x53, 0xa1, 0x7e, 0x03, 0xed	`EFI_ARP_PROTOCOL.Delete()` – returns `EFI_SUCCESS` when deleting the normal entry.	1. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Arp child handle. 2. Call `EFI_ARP_PROTOCOL.Configure()` with valid parameters. 3. Call `EFI_ARP_PROTOCOL.Add()` to add a normal entry. 4. Call `EFI_ARP_PROTOCOL.Delete()` to delete the added entry with key specified as IpAddress which is the same as the *`TargetSwAddress`* of the added entry. The return status must be `EFI_SUCCESS`. 5. Call `EFI_ARP_PROTOCOL.Request()` with *`TargetSwAddress`* which is the same as the *`TargetSwAddress`* of the added entry. The return status must be `EFI_NOT_READY`. 6. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.4.6	0xd2477a4f, 0xef0d, 0x46a2, 0x9a, 0x86, 0x32, 0x82, 0x3f, 0x2c, 0x4b, 0xa3	**EFI_ARP_PROTOCOL.Delete() –** returns **EFI_SUCCESS** when deleting the normal entry.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Delete()** to delete the added entry with key specified as MacAddress which is the same as the *TargetHwAddress* of the added entry. The return status must be **EFI_SUCCESS.** 5. Call **EFI_ARP_PROTOCOL.Request()** with *TargetSwAddress* which is the same as the *TargetSwAddress* of the added entry. The return status must be **EFI_NOT_READY**. 6. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.4.7	0x1e618ee9, 0x40b9, 0x4f79, 0xb9, 0x26, 0xee, 0x2b, 0xa3, 0x73, 0x51, 0x4c	**EFI_ARP_PROTOCOL.Delete() -** returns **EFI_SUCCESS** when deleting all entries.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Delete()** with AddressBuffer set to **NULL** and *BySwAddress* set to **TRUE** to delete all entries. The return status must be **EFI_SUCCESS.** 5. Call **EFI_ARP_PROTOCOL.Request()** with *TargetSwAddress* which is the same as the *TargetSwAddress* of the added entry. The return status must be **EFI_NOT_READY**. 6. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.4.8	0x34a1c3fa, 0xf335, 0x471d, 0x83, 0x03, 0xef, 0x50, 0x98, 0xa3, 0x05, 0x30	`EFI_ARP_PROTOCOL.Delete()` – returns `EFI_SUCCESS` when deleting all entries.	1. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Arp child handle. 2. Call `EFI_ARP_PROTOCOL.Configure()` with valid parameters. 3. Call `EFI_ARP_PROTOCOL.Add()` to add a normal entry. 4. Call `EFI_ARP_PROTOCOL.Delete()` with AddressBuffer set to `NULL` and *`BySwAddress`* set to `FALSE` to delete all entries. The return status must be `EFI_SUCCESS.` 5. Call `EFI_ARP_PROTOCOL.Request()` with *`TargetSwAddress`* which is the same as the *`TargetSwAddress`* of the added entry. The return status must be `EFI_NOT_READY`. 6. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created Arp child handle and clean up the environment.

20.1.5 Find()

Number	GUID	Assertion	Test Description
5.24.1.5.1	0x16bcb5a1, 0xf2c1, 0x419a, 0x8a, 0xf1, 0xea, 0x4b, 0xd9, 0x89, 0x5f, 0xda	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_INVALID_PARAMETER** when both *EntryLength* and *EntryCount* are **NULL** and *Refresh* is **FALSE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Find()** with both *EntryLength* and *EntryCount* are **NULL**, *BySwAddress* is **TRUE**,and *Refresh* is **FALSE**. The return status must be **EFI_INVALID_PARAMETER.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.5.2	0x210ce61b, 0xa76d, 0x4c56, 0xbe, 0x24, 0xe7, 0xb8, 0x11, 0x50, 0xd7, 0x10	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_INVALID_PARAMETER** when both *EntryLength* and *EntryCount* are **NULL** and *Refresh* is **FALSE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Find()** with both the *EntryLength* and *EntryCount* value of **NULL** and a *BySwAddress* value of **FALSE** while *Refresh* is **FALSE**. The return status must be **EFI_INVALID_PARAMETER.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.3	0xf6244c19, 0x6e26, 0x4b9e, 0x84, 0xd3, 0x43, 0x65, 0xb7, 0x6c, 0x17, 0x39	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_INVALID_PARAMETER** when both *EntryLength* and *EntryCount* are **NULL** and Entries are not **NULL** while *Refresh* is **TRUE**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Find()** with both *EntryLength* and *EntryCount* are **NULL** and Entries are not **NULL** while *Refresh* is **TRUE**. The return status must be **EFI_INVALID_PARAMETER.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.5.4	0x5508b3bb, 0x7062, 0x46e7, 0xa4, 0x31, 0xf2, 0xed, 0x67, 0x0b, 0xee, 0x61	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_NOT_FOUND** when no matching entries were found.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Find()** with the specified IpAddress. The return status must be **EFI_NOT_FOUND.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.5	0x9d95d0d7, 0x8e23, 0x4db4, 0xb1, 0xb6, 0x76, 0xc2, 0xee, 0xdc, 0x0f, 0x4f	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_NOT_FOUND** when no matching entries were found.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Find()** with the specified MacAddress. The return status must be **EFI_NOT_FOUND.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.5.6	0x056e9bc8, 0xb221, 0x4063, 0xa2, 0x59, 0x19, 0xe0, 0x08, 0xff, 0x86, 0xda	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_NOT_FOUND** when no matching entries were found.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Find()** with the specified IpAddress and a *Refresh* value of **TRUE**. The return status must be **EFI_NOT_FOUND.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.7	0xc8b3f76f, 0x5ec3, 0x40f6, 0x98, 0x72, 0x31, 0xea, 0x23, 0x6f, 0xc8, 0x08	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_NOT_FOUND** when no matching entries were found.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Find()** with the specified MacAddress and a *Refresh* value of **TRUE**. The return status must be **EFI_NOT_FOUND.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.5.8	0xe0814da9, 0x47fb, 0x443d, 0x84, 0xce, 0xaf, 0x65, 0x01, 0x33, 0x3f, 0x69	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_NOT_FOUND** when no matching entries were found.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Find()** with AddressBuffer set to **NULL** and *BySwAddress* set to **FALSE** while *Refresh* is **TRUE** so as to *refresh* all the entries. The return status must be **EFI_NOT_FOUND.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.9	0xdb367aca, 0xbc94, 0x4c36, 0x92, 0xbd, 0x3b, 0xba, 0x16, 0x9e, 0xc0, 0x6e	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_NOT_FOUND** when no matching entries were found.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Find()** with AddressBuffer set to **NULL** and *BySwAddress* set to **TRUE** while *Refresh* is **TRUE** so as to *refresh* all the entries. The return status must be **EFI_NOT_FOUND.** 4. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.5.10	0x883abd28, 0xd498, 0x4868, 0xb1, 0xa7, 0xe3, 0x22, 0xd1, 0x22, 0x6a, 0x12	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_NOT_STARTED** when the ARP driver instance has not been configured.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Find()** with specified key of IpAddress when *Refresh* is **FALSE**. The return status must be **EFI_NOT_STARTED.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.5.11	0x9301dc5d, 0xc1f2, 0x4858, 0x93, 0xcf, 0xda, 0x77, 0x96, 0xa6, 0x2a, 0x8f	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_NOT_STARTED** when the ARP driver instance has not been configured.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Find()** with specified key of MacAddress and *Refresh* is **TRUE**. The return status must be **EFI_NOT_STARTED.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.12	0x6350837b, 0x0e0e, 0x4241, 0xbd, 0x10, 0x87, 0x77, 0xb3, 0x35, 0xa7, 0xd3	`EFI_ARP_PROTOCOL.Find()` – returns `EFI_SUCCESS` when finding the entry.	1. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Arp child handle. 2. Call `EFI_ARP_PROTOCOL.Configure()` with valid parameters. 3. Call `EFI_ARP_PROTOCOL.Add()` to add a normal entry. 4. Call `EFI_ARP_PROTOCOL.Add()` to add another normal entry with the same *TargetHwAddress* as the one used in the first call to `EFI_ARP_PROTOCOL.Add()`, while the *TargetSwAddress* is different. 5. Call `EFI_ARP_PROTOCOL.Find()` with specified key of MacAddress that is the same as the *TargetHwAddress* in the `EFI_ARP_PROTOCOL.Add()` call. The return status must be `EFI_SUCCESS`. In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2.

Number	GUID	Assertion	Test Description
5.24.1.5.13	0x81716a64, 0x63db, 0x4625, 0xad, 0x87, 0xf1, 0x23, 0x46, 0x94, 0x9f, 0xa9	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_SUCCESS** when finding the entry.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Add()** to add another normal entry with the same *TargetHwAddress* as the first **EFI_ARP_PROTOCOL.Add()**, while the *TargetSwAddress* is different. 5. Call **EFI_ARP_PROTOCOL.Find()** with specified key of MacAddress that is the same as the *TargetHwAddress* in the **EFI_ARP_PROTOCOL.Add()** call. The return status must be **EFI_SUCCESS**. In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2. 6. Call **EFI_ARP_PROTOCOL.Delete()** to delete the entry added in the second time. 7. Call **EFI_ARP_PROTOCOL.Find()** with specified key of IpAddress that is the same as the *TargetSwAddress* in the first **EFI_ARP_PROTOCOL.Add()** call. The return status must be **EFI_SUCCESS**. In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x1. 8. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.14	0x34fd32ad, 0x8e3e, 0x4f49, 0xa0, 0xd7, 0xcc, 0xca, 0xac, 0xa3, 0xce, 0x1f	**`EFI_ARP_PROTOCOL.Find()`** – returns **`EFI_SUCCESS`** when finding the entry.	1. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Arp child handle. 2. Call **`EFI_ARP_PROTOCOL.Configure()`** with valid parameters. 3. Call **`EFI_ARP_PROTOCOL.Add()`** to add a normal entry. 4. Call **`EFI_ARP_PROTOCOL.Add()`** to add another normal entry with the same *`TargetHwAddress`* as the one used in the first call to **`EFI_ARP_PROTOCOL.Add()`**, while the *`TargetSwAddress`* is different. 5. Call **`EFI_ARP_PROTOCOL.Find()`** with AddressBuffer set to **`NULL`** to find all the entries. The return status must be **`EFI_SUCCESS`**. In addition, the return *`EntryLength`* should be 0x16 and the return *`EntryCount`* should be 0x2.

Number	GUID	Assertion	Test Description
5.24.1.5.15	0x3b98d05b, 0x0cd1, 0x41a3, 0xa4, 0x8b, 0x2c, 0xe3, 0x37, 0x6e, 0x0f, 0x09	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_SUCCESS** when finding the entry.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Add()** to add another normal entry with the same *TargetHwAddress* as the first **EFI_ARP_PROTOCOL.Add()**, while the *TargetSwAddress* is different. 5. Call **EFI_ARP_PROTOCOL.Find()** with specified key of MacAddress that is the same as the *TargetHwAddress* in the **EFI_ARP_PROTOCOL.Add()** call. The return status must be **EFI_SUCCESS**. In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2. 6. Call **EFI_ARP_PROTOCOL.Delete()** to delete the entry added in the second time. 7. Call **EFI_ARP_PROTOCOL.Find()** with AddressBuffer set to **NULL** to find all the entries. The return status must be **EFI_SUCCESS**. In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x1. 8. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.16	0x0c8090e4, 0xa0c5, 0x427f, 0xa2, 0xf9, 0x34, 0xd8, 0x10, 0x91, 0x11, 0x2f	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_SUCCESS** when finding the entry with refreshing.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry with timeout set to 50s. 4. Call **EFI_ARP_PROTOCOL.Add()** to add another normal entry with the same *TargetHwAddress* as the one used in the first all to **EFI_ARP_PROTOCOL.Add()**,while the *TargetSwAddress* is different. In addition, timeout is set to 50s. 5. Stall 20 s. 6. Call **EFI_ARP_PROTOCOL.Find()** with specified key of MacAddress the same as *TargetHwAddress* in the **EFI_ARP_PROTOCOL.Add( )** call when **refresh** is **TRUE**. The return status must be **EFI_SUCCESS**. In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2. 7. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.17	0x89474dd0, 0x461b, 0x49c3, 0xa8, 0x5e, 0xaa, 0x16, 0x74, 0xad, 0x6f, 0x9d	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_SUCCESS** when finding the entry without refreshing.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry with timeout set to 50s. 4. Call **EFI_ARP_PROTOCOL.Add()** to add another normal entry with the same *TargetHwAddress* as the one used in the first all to **EFI_ARP_PROTOCOL.Add()**,while the *TargetSwAddress* is different. In addition, timeout is set to 50s. 5. Stall 20 s. 6. Call **EFI_ARP_PROTOCOL.Find()** with specified key of MacAddress the same as *TargetHwAddress* in the **EFI_ARP_PROTOCOL.Add**() call when *refresh* is **TRUE**. The return status must be **EFI_SUCCESS**. In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2. 7. Stall 35 s. 8. Call **EFI_ARP_PROTOCOL.Find()** with specified key of MacAddress the same as *TargetHwAddress* in the **EFI_ARP_PROTOCOL.Add()** call and a *refresh* value of **FALSE**. The return status must be **EFI_SUCCESS**. In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2. 9. Stall 20 s. 10. Call **EFI_ARP_PROTOCOL.Find()** with specified key of MacAddress the same as *TargetHwAddress* in the **EFI_ARP_PROTOCOL.Add( )** call with a *refresh* value of **FALSE**. The return status must be **EFI_NOT_FOUND**. 11. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.18	0x97fbb88f, 0x0566, 0x4b4b, 0x93, 0xfe, 0x5e, 0xc9, 0xad, 0x60, 0x8d, 0x7e	`EFI_ARP_PROTOCOL.Find()` – returns `EFI_SUCCESS` when finding the entry with a *`SwAddressLength`* value of 16.	1. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Arp child handle. 2. Call `EFI_ARP_PROTOCOL.Configure()` with a *`SwAddressLength`* value of 16. 3. Call `EFI_ARP_PROTOCOL.Add()` to add a normal entry. 4. Call `EFI_ARP_PROTOCOL.Add()` to add another normal entry with the same *`TargetHwAddress`* as the one used in the first call to `EFI_ARP_PROTOCOL.Add()`, while *`TargetSwAddress`* is different. 5. Call `EFI_ARP_PROTOCOL.Find()` with specified key of MacAddress the same as *`TargetHwAddress`* in the `EFI_ARP_PROTOCOL.Add()` call. The return status must be `EFI_SUCCESS`. In addition, the return *`EntryLength`* should be 0x22 and the return *`EntryCount`* should be 0x2. 6. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.19	0xcbd6f47d, 0x2edc, 0x4235, 0x91, 0x50, 0x1f, 0xba, 0xe9, 0x07, 0xac, 0x26	**EFI_ARP_PROTOCOL.Find()** – returns **EFI_SUCCESS** when finding the entry with a *SwAddressLength* value of 16.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with a *SwAddressLength* value of 16. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Add()** to add another normal entry with the same *TargetHwAddress* as the one used in the first call to **EFI_ARP_PROTOCOL.Add()**, while *TargetSwAddress* is different. 5. Call **EFI_ARP_PROTOCOL.Find()** with specified key of MacAddress the same as *TargetHwAddress* in the **EFI_ARP_PROTOCOL.Add()** call. The return status must be **EFI_SUCCESS**. In addition, the return **EntryLength** should be 0x22 and the return **EntryCount** should be 0x2. 6. Call **EFI_ARP_PROTOCOL.Delete()** to delete the entry added in the second time. 7. Call **EFI_ARP_PROTOCOL.Find()** with specified key of IpAddress the same as *TargetSwAddress* in the first **EFI_ARP_PROTOCOL.Add()** call. The return status must be **EFI_SUCCESS**. In addition, the return *EntryLength* should be 0x22 and the return *EntryCount* should be 0x1. 8. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.20	0x630e139e, 0x287a, 0x456c, 0xa5, 0xf7, 0x58, 0x35, 0xaf, 0x42, 0xf7, 0x7d	**`EFI_ARP_PROTOCOL.Find()`** – returns **`EFI_SUCCESS`** when finding the entry with a *`SwAddressLength`* value of 1.	1. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Arp child handle. 2. Call **`EFI_ARP_PROTOCOL.Configure()`** with a *`SwAddressLength`* value of 1. 3. Call **`EFI_ARP_PROTOCOL.Add()`** to add a normal entry. 4. Call **`EFI_ARP_PROTOCOL.Add()`** to add another normal entry with the same *`TargetHwAddress`* as the first **`EFI_ARP_PROTOCOL.Add()`**, while the *`TargetSwAddress`* is different. 5. Call **`EFI_ARP_PROTOCOL.Find()`** with specified key of MacAddress the same as the *`TargetHwAddress`* in the **`EFI_ARP_PROTOCOL.Add()`** call. The return status must be **`EFI_SUCCESS`**. In addition, the return *`EntryLength`* should be 0x13 and the return *`EntryCount`* should be 0x2. 8. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.5.21	0xf7c0f95a, 0xfaa2, 0x4577, 0x8c, 0x66, 0xb4, 0x76, 0x82, 0x00, 0x85, 0x5d	**`EFI_ARP_PROTOCOL.Find()`** – returns **`EFI_SUCCESS`** when finding the entry with a *`SwAddressLength`* value of 1.	1. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Arp child handle. 2. Call **`EFI_ARP_PROTOCOL.Configure()`** with a *`SwAddressLength`* value of 1. 3. Call **`EFI_ARP_PROTOCOL.Add()`** to add a normal entry. 4. Call **`EFI_ARP_PROTOCOL.Add()`** to add another normal entry with the same *`TargetHwAddress`* as the first **`EFI_ARP_PROTOCOL.Add()`**, while the *`TargetSwAddress`* is different. 5. Call **`EFI_ARP_PROTOCOL.Find()`** with specified key of MacAddress the same as the *`TargetHwAddress`* in the **`EFI_ARP_PROTOCOL.Add()`** call. The return status must be **`EFI_SUCCESS`**. In addition, the return *`EntryLength`* should be 0x13 and the return *`EntryCount`* should be 0x2. 6. Call **`EFI_ARP_PROTOCOL.Delete()`** to delete the entry added in the second time. 7. Call **`EFI_ARP_PROTOCOL.Find()`** with specified key of IpAddress the same as the *`TargetSwAddress`* in the first **`EFI_ARP_PROTOCOL.Add()`** call. The return status must be **`EFI_SUCCESS`**. In addition, the return *`EntryLength`* should be 0x13 and the return *`EntryCount`* should be 0x1. 8. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created Arp child handle and clean up the environment.

20.1.6 Flush()

Number	GUID	Assertion	Test Description
5.24.1.6.1	0x057bd5b9, 0xc869, 0x4446, 0xa9, 0xd1, 0x79, 0x07, 0xdc, 0xf8, 0x74, 0xf0	**EFI_ARP_PROTOCOL.Flush() –** returns **EFI_NOT_FOUND** when flushing the entry again after the first flush.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal permanent entry 1. 4. Call **EFI_ARP_PROTOCOL.Add()** to add a normal dynamic entry 2. 5. Call **EFI_ARP_PROTOCOL.Add()** to add a normal permanent entry 3. 6. Call **EFI_ARP_PROTOCOL.Add()** to add a normal dynamic entry 4. 7. Call **EFI_ARP_PROTOCOL.Flush()** to remove all dynamic cache entries. 8. Call **EFI_ARP_PROTOCOL.Flush()** again. The return status must be **EFI_NOT_FOUND.** 9. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.6.2	0xe34bd9b5, 0x94b2, 0x422a, 0xb8, 0xd1, 0x6c, 0x18, 0x07, 0x6c, 0xef, 0xbb	**EFI_ARP_PROTOCOL.Flush() –** returns **EFI_NOT_STARTED** when the arp driver instance has not been configured.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Flush().** The return status must be **EFI_NOT_STARTED.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.6.3	0xf2cc7ff1, 0x9049, 0x4daa, 0xa3, 0x4d, 0xca, 0x55, 0xf5, 0xe9, 0x67, 0x55	`EFI_ARP_PROTOCOL.Flush()` – returns `EFI_SUCCESS` when flushing the entry.	1. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Arp child handle. 2. Call `EFI_ARP_PROTOCOL.Configure()` with valid parameters.e 3. Call `EFI_ARP_PROTOCOL.Add()` to add a normal permanent entry 1. 4. Call `EFI_ARP_PROTOCOL.Add()` to add a normal dynamic entry 2 – timeout is 50s. 5. Call `EFI_ARP_PROTOCOL.Add()` to add a normal permanent entry 3. 6. Call `EFI_ARP_PROTOCOL.Add()` to add a normal dynamic entry 4 – timeout is 50s. 7. Call `EFI_ARP_PROTOCOL.Flush()` to remove all dynamic cache entries. The return status must be `EFI_SUCCESS.` 8. Call `EFI_ARP_PROTOCOL.Find()` with specified key of IpAddress the same as the *`TargetSwAddress`* in the first `EFI_ARP_PROTOCOL.Add()` call. The return status must be `EFI_SUCCESS`. 9. Call `EFI_ARP_PROTOCOL.Find()` with specified key of MacAddress the same as the *`TargetHwAddress`* in the first `EFI_ARP_PROTOCOL.Add()` call. The return status must be `EFI_SUCCESS`. 10. Call `EFI_ARP_PROTOCOL.Find()` with specified key of IpAddress the same as the *`TargetSwAddress`* in the second `EFI_ARP_PROTOCOL.Add()` call. The return status must be `EFI_NOT_FOUND`. 11. Call `EFI_ARP_PROTOCOL.Find()` with specified key of MacAddress as same as the *`TargetHwAddress`* in the second `EFI_ARP_PROTOCOL.Add()` call. The return status must be `EFI_NOT_FOUND`. 14. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created Arp child handle and clean up the environment.

20.1.7 Request()

Number	GUID	Assertion	Test Description
5.24.1.7.1	0x464366ea, 0xf5a5, 0x47a0, 0x8b, 0x3b, 0x67, 0x09, 0x89, 0xcf, 0x43, 0xd2	**EFI_ARP_PROTOCOL.Request()** – returns **EFI_INVALID_PARAMETER** when *TargetHwAddress* is **NULL**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Request()** with a *TargetHwAddress* value of **NULL** ,and both the *ResolvedEvent* and *TargetSwAdddress* value other than **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.7.2	0xb4df082c, 0xb895, 0x4ec8, 0xac, 0xc7, 0x26, 0x58, 0x87, 0xc7, 0xe3, 0xbb	**EFI_ARP_PROTOCOL.Request()** – returns **EFI_INVALID_PARAMETER** when *TargetHwAddress* is **NULL**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Request()** with a *TargetHwAddress* value of **NULL** ,a *ResolvedEvent* value of **NULL**, and a *TargetSwAdddress* value other than **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.7.3	0x58d0454a, 0xeed1, 0x4ccd, 0xa3, 0xd0, 0x10, 0xa5, 0xa8, 0x71, 0x46, 0x38	**EFI_ARP_PROTOCOL.Request() -** returns **EFI_INVALID_PARAMETER** when *TargetHwAddress* *TargetHwAddress* is **NULL**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Request()** with a *TargetHwAddress* value of **NULL**, and both the *ResolvedEvent* and *TargetSwAdddress* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.7.4	0xe726cb6e, 0x3ee3, 0x474e, 0x9c, 0x1c, 0xa7, 0xc7, 0xa6, 0x93, 0x85, 0x1d	**EFI_ARP_PROTOCOL.Request()** – returns **EFI_ACCESS_DENIED** when the requested Address is present in the deny address list.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry. 4. Call **EFI_ARP_PROTOCOL.Add()** to add a deny entry whose *TargetSwAddress* is the same as the one used in the first Add() call to overwrite the entry first added. 5. Call **EFI_ARP_PROTOCOL.Request()** with *TargetSwAddress* the same as the one used in the call to **EFI_ARP_PROTOCOL.Add()**. The return status must be **EFI_ACCESS_DENIED.** In addition, the return *TargetHwAddress* should be 0:0:0:0:0:0. 6. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.7.5	0xd774703f, 0x7ed8, 0x48da, 0x9f, 0x86, 0x5e, 0xf8, 0x19, 0x47, 0xb6, 0x47	**EFI_ARP_PROTOCOL.Request()** – returns **EFI_NOT_STARTED** when the ARP driver instance has not been configured.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Request()** when both *TargetSwAddress* and *ResolvedEvent* are not **NULL**. The return status must be **EFI_NOT_STARTED.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.7.6	0x122d41e6, 0x252a, 0x4afb, 0xa2, 0x47, 0x03, 0x56, 0xd5, 0x3c, 0x4a, 0x64	**EFI_ARP_PROTOCOL.Request() –** returns **EFI_NOT_STARTED** when the ARP driver instance has not been configured.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Request()** when *TargetSwAddress* is not **NULL** and *ResolvedEvent* is **NULL**. The return status must be **EFI_NOT_STARTED.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.
5.24.1.7.7	0xca3946d0, 0x64ff, 0x4139, 0x97, 0x66, 0x82, 0x91, 0xcb, 0xc1, 0x12, 0x09	**EFI_ARP_PROTOCOL.Request() –** returns **EFI_NOT_STARTED** – when the ARP driver instance has not been configured.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Request()** when both *TargetSwAddress* and *ResolvedEvent* are **NULL**. The return status must be **EFI_NOT_STARTED.** 3. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.7.8	0xf4b08f82, 0xdafd, 0x4618, 0x94, 0xed, 0x15, 0xf8, 0x54, 0xce, 0xe3, 0x9f	**EFI_ARP_PROTOCOL.Request()** – returns **EFI_NOT_READY** – when the request has been started and is not finished.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Request()** with *TargetSwAddress* “172.16.210.161”. 4. The OS side should capture the request packet and send back the reply packet with SourceIp “172.16.210.161”, SourceMac “0:2:3:4:5:6”. The return status must be **EFI_NOT_READY.** In addition, the return *TargetHwAddress* should be “0:2:3:4:5:6”. 10. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.7.9	0x3d6668d9, 0x631c, 0x4cee, 0xae, 0xc9, 0xc1, 0x0f, 0x3f, 0xe6, 0xee, 0x27	**EFI_ARP_PROTOCOL.Request() -** returns **EFI_NOT_READY** – when the request has been started and is not finished.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Request()** with *TargetSwAddress* “172.16.210.161”. 4. The OS side should capture the request packet and send back the reply packet with SourceIp “172.16.210.161”, SourceMac “0:2:3:4:5:6”. The return status must be **EFI_NOT_READY.** In addition, the return *TargetHwAddress* should be “0:2:3:4:5:6”. 5. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry with *TargetSwAddress* “172.16.210.161” and *TargetHwAddress* “0:2:3:4:5:6”. 6. Call **EFI_ARP_PROTOCOL.Add()** to add a deny entry with the same *TargetHwAddress* as the one used in the first Add( ) to overwrite the entry first added. 7. Call **EFI_ARP_PROTOCOL.Request()** with *TargetSwAddress* “172.16.210.161”. 8. The OS side should capture the request packet and send back the first reply packet with SourceIp “172.16.210.161”, SourceMac “0:2:3:4:5:6”. 9. Then OS sends back the second reply packet with SourceIp “172.16.210.161”, SourceMac “0:2:3:4:5:7”. The return status must be **EFI_NOT_READY.** In addition, the return *TargetHwAddress* should be “0:2:3:4:5:7”. 10. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.7.10	0xe37f681b, 0xab41, 0x4370, 0xab, 0x02, 0xf6, 0xd5, 0xfb, 0x0a, 0xf2, 0xb7	**EFI_ARP_PROTOCOL.Request() -** returns **EFI_SUCCESS** when the data was copied from the ARP cache into the *TargetHwAddress* buffer.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Request()** with *TargetSwAddress* “172.16.210.161”. 4. The OS side should capture the request packet and validate whether the packet is rightly sent from the EUT side. 5. The OS sends back the reply packet with SourceIp “172.16.210.161”, SourceMac “0:2:3:4:5:6”. The return status must be **EFI_SUCCESS.** In addition, the return *TargetHwAddress* “0:2:3:4:5:6”. 6. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.24.1.7.11	0x93e9a6d8, 0xb732, 0x40d7, 0x8d, 0x1e, 0xe5, 0xdb, 0xa6, 0xf6, 0x02, 0x1e	**`EFI_ARP_PROTOCOL.Request()`** – returns **`EFI_SUCCESS`** when the data was copied from the ARP cache into the *`TargetHwAddress`* buffer.	1. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Arp child handle. 2. Call **`EFI_ARP_PROTOCOL.Configure()`** with valid parameters. 3. Call **`EFI_ARP_PROTOCOL.Request()`** with *`TargetSwAddress`* "172.16.210.161". 4. The OS side should capture the request packet and validate whether the packet is rightly sent from the EUT side. 5. The OS sends back the reply packet with SourceIp "172.16.210.161", SourceMac "0:2:3:4:5:6". The return status must be **`EFI_SUCCESS.`** In addition, the return *`TargetHwAddress`* "0:2:3:4:5:6". 6. Call **`EFI_ARP_PROTOCOL.Request()`** with broadcast destination address to resolve *`TargetSwAddress`* "172.16.210.161". The return status must be **`EFI_SUCCESS.`** In addition, the return *`TargetHwAddress`* should be "0:2:3:4:5:6". 7. Call **`EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created Arp child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.24.1.7.12	0xa227797d, 0x00b5, 0x4ff0, 0xb4, 0x62, 0x46, 0x87, 0xa1, 0x31, 0xa0, 0x1c	**EFI_ARP_PROTOCOL.Request() –** returns **EFI_SUCCESS** when the data was copied from the ARP cache into the *TargetHwAddress* buffer.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry with *TargetSwAddress* "172.16.210.161" and TargeHwAddress "0:2:3:4:5:6" 4. Call **EFI_ARP_PROTOCOL.Request()** with *TargetSwAddress* "172.16.210.161". The return status must be **EFI_SUCCESS.** In addition, the return *TargetHwAddress* should be "0:2:3:4:5:6". 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.24.1.7.13	0xd958bbd5, 0x3429, 0x4b94, 0x9f, 0xe5, 0x8e, 0xe1, 0xf4, 0x8b, 0xfd, 0xd2	**EFI_ARP_PROTOCOL.Request() -** returns **EFI_SUCCESS** when requesting the entry whose *TargetSwAddress* is a multicast IP address.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry with *TargetSwAddress* “172.16.210.161” and TargeHwAddress “0:2:3:4:5:6” 4. Call **EFI_ARP_PROTOCOL.Request()** to resolve multicast IP address “224.0.1.2”. The return status must be **EFI_SUCCESS.** In addition, the return *TargetHwAddress* should be “1:0:5e:0:1:2”. 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.7.14	0x46146a28, 0x7af5, 0x43c5, 0xb7, 0xd1, 0x6f, 0xfb, 0xd6, 0xa4, 0x89, 0x97	**EFI_ARP_PROTOCOL.Request() -** returns **EFI_SUCCESS** when requesting the entry whose *TargetSwAddress* is a multicast IP address.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry with *TargetSwAddress* “172.16.210.161” and *TargetHwAddress* “0:2:3:4:5:6” 4. Call **EFI_ARP_PROTOCOL.Request()** to resolve multicast IP address “238.255.255.255”. The return status must be **EFI_SUCCESS.** In addition, the return *TargetHwAddress* is “1:0:5e:7f: ff: ff”. 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.24.1.7.15	0x50ecb99e, 0xfdab, 0x441c, 0x85, 0x08, 0x92, 0x5f, 0x1b, 0xdf, 0x42, 0x4b	**EFI_ARP_PROTOCOL.Request()** - returns **EFI_SUCCESS** when requesting the entry whose *TargetSwAddress* is **NULL**.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry with *TargetSwAddress* “172.16.210.161” and TargeHwAddress “0:2:3:4:5:6” 4. Call **EFI_ARP_PROTOCOL.Request()** when *TargetSwAddress* is **NULL**. The return status must be **EFI_SUCCESS.** In addition, the return *TargetHwAddress* should be “ff: ff: ff: ff: ff: ff”. (Network interface hardware broadcast address). 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.24.1.7.16	0x50d9cb20, 0x1177, 0x4b13, 0xbc, 0x41, 0xf0, 0xf3, 0x2a, 0x3d, 0xf9, 0x02	**EFI_ARP_PROTOCOL.Request() -** returns **EFI_SUCCESS** when requesting the entry whose *TargetSwAddress* is “255.255.255.255”.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Add()** to add a normal entry with *TargetSwAddress* “172.16.210.161” and TargeHwAddress “0:2:3:4:5:6” 4. Call **EFI_ARP_PROTOCOL.Request()** when *TargetSwAddress* is “255.255.255.255”. The return status must be **EFI_SUCCESS.** In addition, the return *TargetHwAddress* should be “ff: ff: ff: ff: ff: ff”. (Network interface hardware broadcast address). 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.24.1.7.17	0xf7140dcf, 0x0d15, 0x438a, 0xa3, 0x4d, 0x47, 0x23, 0x97, 0x6f, 0x0b, 0xc8	**EFI_ARP_PROTOCOL.Request() –** returns **EFI_SUCCESS** when calling **Request ()** twice with the same *TargetSwAddress*.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Call **EFI_ARP_PROTOCOL.Request()** when *TargetSwAddress* is “172.16.210.161”. The return status should be **EFI_NOT_READY**. 4. Call **EFI_ARP_PROTOCOL.Request()** again when *TargetSwAddress* is “172.16.210.161”. The return status should be **EFI_NOT_READY**. 5. The OS side should capture the request packet and send back the reply packet with SouceIP “172.16.210.161” and SourceMac ”0:2:3:4:5:6”. The return *EventContext* should be 2. The return status must be **EFI_SUCCESS.** In addition, the return *TargetHwAddress* should be “0:2:3:4:5:6”. 6. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment

20.1.8 CreateChild()

Number	GUID	Assertion	Test Description
5.24.1.8.1	0xd01e591b, 0x6b83, 0x417c, 0xbf, 0xe0, 0x1d, 0xb3, 0x78, 0xea, 0x2c, 0x78	`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` - returns `EFI_INVALID_PARAMETER` with `NULL` child handle.	1. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` with the parameter a *`ChildHandle`* value of `NULL`. The return status must be `EFI_INVALID_PARAMETER.`
5.24.1.8.2	0x51d66e16, 0x39f6, 0x4fff, 0x8a, 0x99, 0xf2, 0x95, 0x01, 0xe3, 0x4b, 0xe8	`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` - returns `EFI_INVALID_PARAMETER` with invalid child handle.	1. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` with a invalid *`ChildHandle`*. The return status must be `EFI_INVALID_PARAMETER.`
5.24.1.8.3	0x460a6262, 0xaa4d, 0x4e25, 0x92, 0x6b, 0x55, 0x1e, 0xf0, 0xb5, 0x6d, 0x37	`EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` – invokes `CreateChild()` to create different childs.	Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` to create childs three times and then destroy them.

20.1.9 DestroyChild()

Number	GUID	Assertion	Test Description
5.24.1.9.1	0xfaabc3ef, 0xc56f, 0x44d1, 0xbe, 0xb6, 0x53, 0x5b, 0x26, 0x4d, 0xba, 0x63	`EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()` - returns `EFI_UNSUPPORTED` with invalid child handle.	1. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Arp child handle. 2. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created Arp child handle The return status must be `EFI_SUCCESS.` 3. Call `EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()` again with value of Handle set to be 8 and clean up the environment. The return status must be `EFI_UNSUPPORTED.`

Number	GUID	Assertion	Test Description
5.24.1.9.2	0x7b8de1fe, 0x93e1, 0x48a4, 0xa0, 0x5e, 0x38, 0xad, 0x8f, 0x26, 0xf0, 0x83	EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild() - returns EFI_INVALID_PARAMETER with NULL child.	1. Call EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the NULL child.
5.24.1.9.3	0xf651081a, 0xb71f, 0x4617, 0x99, 0x7a, 0xd1, 0x87, 0x7a, 0x07, 0x03, 0x28	EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild() - returns EFI_INVALID_PARAMETER and inexistent child.	1. Call EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the inexistent child.
5.24.1.9.4	0x5772a154, 0xb8f5, 0x4fec, 0xaa, 0x80, 0xae, 0xb9, 0x0c, 0x4c, 0xd2, 0x5d	EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild() – invokes DestroyChild() to destroy different childs	Call EFI_ARP_SERVICE_BINDING_PROTOCOL. DestroyChild () to destroy the newly three created Arp childs.

20.1.10 RFC Related

Number	GUID	Assertion	Test Description
5.24.1.10.1	0x0f6557a8, 0xf383, 0x436e, 0x96, 0x2b, 0x88, 0x2a, 0x28, 0x3c, 0x4c, 0x64	EFI_ARP_PROTOCOL.Rfc – Send an ARP request and check the ARP reply .	1. Call EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Arp child handle. 2. Call EFI_ARP_PROTOCOL.Configure() with valid parameters. 3. Send ARP request to the broadcast address with sender ip “172.16.210.161” and sender Mac”0:2:3:4:5:7” to resolve the Target ip”172.16.210.102”. 4. Then the OS side should capture the reply packet. 5. If having captured, dump the reply packet and validate whether the sender Mac is the MacAddress of TargetMachine. 6. Call EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.1.10.2	0x842c7377, 0x04b6, 0x459f, 0x92, 0x56, 0x39, 0xbf, 0x2e, 0x2f, 0xc5, 0x93	**EFI_ARP_PROTOCOL.Rfc –** without reply when sending an ARP request with opcode invalid.	1. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Arp child handle. 2. Call **EFI_ARP_PROTOCOL.Configure()** with valid parameters. 3. Send ARP request to the broadcast address with sender ip "172.16.210.161" and sender Mac"0:2:3:4:5:7" to resolve the Target ip"172.16.210.102" – the opcode set to 255. 4. Then the OS side shouldn't capture the reply packet. 5. Call **EFI_ARP_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created Arp child handle and clean up the environment.

20.2 EFI_DHCP4_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_DHCP4_PROTOCOL Section.

20.2.1 GetModeData()

Number	GUID	Assertion	Test Description
5.24.2.1.1	0x52159e94, 0x4a67, 0x44f6, 0x9b, 0x0b, 0x83, 0x21, 0x93, 0x41, 0xe1, 0xf3	`EFI_DHCP4_PROTOCOL.GetModeData()` - invokes `GetModeData()` to get all mode data when the Dhcp4 child has not been configured.	1. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Dhcp4 child. 2. Call `EFI_DHCP4_PROTOCOL.GetModeData()` to get all mode data when the Dhcp4 child has not been configured. The ModeData.State should be Dhcp4Stopped.The return status should be `EFI_SUCCESS`. 3. Call `EFI_DHCP4_PROTOCOL.Stop()` to verify the Dhcp4 child in the Dhcp4Stopped state. 4. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.1.2	0x969e5dac, 0x2097, 0x4a3f, 0xaa, 0x15, 0xb0, 0x6d, 0xff, 0x26, 0x48, 0xec	**EFI_DHCP4_PROTOCOL.GetModeData()** - invokes **GetModeData()** to get DHCP4 mode data during the configuration process.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0". 3. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data when the Dhcp4 child has been configured. The ModeData.State should be Dhcp4Init.The return status should be **EFI_SUCCESS**. 4. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 5. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data during the configuration process. The ModeData.State should be Dhcp4Selecting.The return status should be **EFI_SUCCESS**. 6. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.1.3	0xca520116, 0x5097, 0x4cda, 0x80, 0x79, 0x4a, 0x9b, 0x8f, 0xdd, 0x88, 0x38	**EFI_DHCP4_PROTOCOL.GetModeData()** - invokes **GetModeData()** to get DHCP4 mode data during the configuration process.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.1.24". 3. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data when the Dhcp4 child has been configured. The ModeData.State should be Dhcp4InitReboot.The return status should be **EFI_SUCCESS**. 4. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 5. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data during the configuration process. The ModeData.State should be Dhcp4Rebooting.The return status should be **EFI_SUCCESS**. 6. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

20.2.2 Configure()

Number	GUID	Assertion	Test Description
5.24.2.2.1	0xbd919c90, 0x708b, 0x4502, 0xad, 0xd7, 0xd5, 0x85, 0x30, 0x4b, 0x84, 0x0e	**EFI_DHCP4_PROTOCOL.Configure()** - invokes **Configure()** when this driver instance was not in the Dhcp4Stopped, Dhcp4Init, Dhcp4InitReboot, or Dhcp4Bound state.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data during the configuration process. The ModeData.State should be Dhcp4Selecting.The return status should be **EFI_SUCCESS**. 5. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure the child during the configuration process. The return status should be **EFI_ACCESS_DENIED**. 6. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.2.2	0x57b62321, 0x14a8, 0x4412, 0xb4, 0x20, 0xad, 0x49, 0x5d, 0x6a, 0xab, 0xbb	**EFI_DHCP4_PROTOCOL.Configure()** - invokes **Configure()** when Another instance is already in a valid configured state.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child1. 2. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child2. 3. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child1. 4. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child2. The return status should be **EFI_ACCESS_DENIED**. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.2.3	0x5101b2b6, 0x8021, 0x4a04, 0x90, 0x83, 0xf6, 0x6b, 0x9f, 0x4d, 0x10, 0x1f	**EFI_DHCP4_PROTOCOL.Configure()** - invokes **Configure()** with invalid parameters, among which *DiscoverTryCount* is positive and *DiscoverTimeout* is **NULL**.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure the new instance with a *DiscoverTryCount* value of positive and a *DiscoverTimeout* value of **NULL**. The return status should be **EFI_INVALID_PATAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.2.4	0x50f034a4, 0x2aa4, 0x4d1a, 0x8a, 0x8c, 0x9d, 0x7c, 0x06, 0x48, 0xc9, 0x35	**EFI_DHCP4_PROTOCOL.Configure()** - invokes **Configure()** with invalid parameters, among which *RequestTryCount* is positive and *RequestTimeout* is **NULL**.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure the new instance with a *RequestTryCount* value of positive and a *RequestTimeout* value of **NULL**. The return status should be **EFI_INVALID_PATAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.2.5	0xc199419b, 0x62b1, 0x4cda, 0xb4, 0x38, 0x9d, 0xcd, 0xed, 0x4d, 0x83, 0x6d	**EFI_DHCP4_PROTOCOL.Configure()** - invokes **Configure()** with invalid parameters, among which *OptionCount* is positive and *OptionList* is **NULL**.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure the new instance with a *OptionCount* value of positive and a *OptionList* value of **NULL**. The return status should be **EFI_INVALID_PATAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.2.6	0xada01077, 0x4869, 0x4c21, 0x8f, 0x6d, 0x6e, 0x65, 0x93, 0x41, 0xbc, 0xa6	**EFI_DHCP4_PROTOCOL.Configure()** - invokes **Configure()** with invalidparameters, except that *ClientAddress* is an invalid unicast address.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure the new instance with a *ClientAddress* value of an invalid unicast address. The return status should be **EFI_INVALID_PATAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.2.7	0xde6079f0, 0x4aa4, 0x4665, 0x80, 0x8b, 0xa0, 0x22, 0x3c, 0x8b, 0xf6, 0x40	`EFI_DHCP4_PROTOCOL.Configure()` - invokes `Configure()` to Validate the configuration data effect before and after calling Dhcp.start() to start the Configuration.	1. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Dhcp4 child. 2. Call `EFI_DHCP4_PROTOCOL.Configure()` to configure child with setting *ClientAddress* "0.0.0.0". 3. Call `EFI_DHCP4_PROTOCOL.GetModeData()` to check the configuration data effect. 4. Call `EFI_DHCP4_PROTOCOL.Start()` to start the configuration process with a *CompletionEvent* value of `NULL`. 5. Call `EFI_DHCP4_PROTOCOL.GetModeData()` to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init. 6. Call Call `EFI_DHCP4_PROTOCOL.Configure()` to configure child with setting *ClientAddress* "192.168.2.3". 7. Call `EFI_DHCP4_PROTOCOL.GetModeData()` to check the configuration data effect. 8. Call `EFI_DHCP4_PROTOCOL.Start()` to start the configuration process with a *CompletionEvent* value of `NULL.` 9. Call `EFI_DHCP4_PROTOCOL.GetModeData()` to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init. 10. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.2.8	0x73401b2e, 0x30aa, 0x422d, 0xa3, 0xca, 0x9f, 0x36, 0x78, 0x1c, 0xfa, 0x94	**EFI_DHCP4_PROTOCOL.Configure()** - invokes **Configure()** to Validate the configuration data effect before and after calling **Dhcp->start** to start the Configuration, Call **Dhcp.stop()** before calling **Dhcp.start()** again.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0". 3. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to check the configuration data effect. 4. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 5. Call **EFI_DHCP4_PROTOCOL.Stop()** to stop the configuration. 6. Call Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting **ClientAddress** "192.168.2.3". 7. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to check the configuration data effect. 8. Call **EFI_DHCP4_PROTOCOL.Start()** again to start the configuration process with a *CompletionEvent* value of **NULL**. 9. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init. 10. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.2.9	0x1a27208e, 0x08a8, 0x42a6, 0xb9, 0x3f, 0x8b, 0x95, 0x94, 0x24, 0x46, 0xb7	**EFI_DHCP4_PROTOCOL.Configure()** - invokes **Configure()** with the following condition: if one instance wants to make it possible for another instance to configure successfully, it must call **EFI_DHCP4_PROTOCOL.Configure()** with DhcpCfgData set to **NULL**.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child1. 2. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child2. 3. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child1. 4. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child2. The return status should be **EFI_ACCESS_DENIED**. 5. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child1 with setting *ConfigData* to **NULL**. The return status should be **EFI_SUCCESS**. 6. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child2. The return status should be **EFI_SUCCESS**. 7. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

20.2.3 Start()

Number	GUID	Assertion	Test Description
5.24.2.3.1	0xbac2be63, 0xd705, 0x4667, 0x9d, 0x1b, 0x04, 0xe0, 0x5e, 0xeb, 0xcb, 0x3a	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** when the driver instance is in the Dhcp4Stopped state.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value other than **NULL**. The return status should be **EFI_NOT_STARTED**. 3. Call **EFI_DHCP4_PROTOCOL.Stop()** to stop the configuration process. 4. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.3.2	0xc67ae0d7, 0x3401, 0x4daf, 0xa6, 0x4c, 0xb9, 0xa6, 0x0e, 0xea, 0x17, 0x71	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** with no response during the specified timeout value.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0". 2. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value other than **NULL**. The return status should be **EFI_TIMEOUT**. 3. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init. 4. Call **EFI_DHCP4_PROTOCOL.Stop()** to stop the configuration process. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.3.3	0xd7cd1980, 0x7509, 0x4612, 0x80, 0xc0, 0x5c, 0x21, 0x5b, 0x9e, 0x8e, 0x10	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** while the user aborts the DHCP process.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0" and Dhcp4Callback=1(Callbackfunctionlist[1]=Aborted) 2. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value other than **NULL**. The return status should be **EFI_ABORTED**. 3. Call **EFI_DHCP4_PROTOCOL.Stop()** to stop the configuration process. 4. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.3.4	0x580e7e81, 0x506d, 0x4339, 0xb7, 0xc2, 0x9f, 0x05, 0x53, 0x8f, 0xf5, 0xde	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** to start configuration process while another instance has already started the DHCP process.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child1. 2. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child2. 3. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child1. 4. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process of child1 with a *CompletionEvent* value other than **NULL**. The return status should be **EFI_SUCCESS**. 5. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process of child2 with a *CompletionEvent* value of **NULL**. The return status should be **EFI_ALREADY_STARTED**. 6. Call **EFI_DHCP4_PROTOCOL.Stop()** to stop the configuration process of child1. 7. Call **EFI_DHCP4_PROTOCOL.Stop()** to stop the configuration process of child2. 8. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.3.5	0x8bd59e83, 0x3f3a, 0x4649, 0xb8, 0x61, 0x36, 0x56, 0x23, 0x5c, 0x8f, 0x7d	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** in Dhcp4Init State and Asynchronous Mode. (Calling functions in sequence A).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of not **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.3.6	0xaca2403d, 0x458b, 0x4c8e, 0x8f, 0x77, 0x1f, 0x87, 0x85, 0x31, 0x08, 0xed	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** in Dhcp4Init State and Asynchronous Mode. (Calling functions in sequence B).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value other than **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.3.7	0x7344b984, 0x306d, 0x467b, 0xa4, 0x3d, 0x36, 0x77, 0xf8, 0xc9, 0x79, 0x78	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** in Dhcp4Init State and Asynchronous Mode. (Calling functions in sequence C).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value other than **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPNAK packet. The ModeData.State should be Dhcp4Init. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.3.8	0xf9a23299, 0xeb65, 0x472b, 0xbe, 0x96, 0xe5, 0xea, 0x77, 0x2e, 0x03, 0xc0	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** in Dhcp4InitReboot State and Asynchronous Mode. (Calling functions in sequence A).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value other than **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.3.9	0x723e3088, 0x5f48, 0x4b09, 0x9b, 0x17, 0x80, 0x45, 0x86, 0xf9, 0x9a, 0xad	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** in Dhcp4InitReboot State and Asynchronous Mode. (Calling functions in sequence B).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value other than **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver havng stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.3.10	0xa8fcde55, 0x522b, 0x49ea, 0xbc, 0xe8, 0x6b, 0xea, 0x80, 0x57, 0x91, 0x21	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** in Dhcp4InitReboot State and Asynchronous Mode. (Calling functions in sequence C).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value other than **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPNAK packet. The ModeData.State should be Dhcp4Init. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.3.11	0x941de4e1, 0xc289, 0x417b, 0x87, 0xeb, 0xef, 0x3e, 0x1e, 0xc0, 0x12, 0x3d	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** in Dhcp4Init State and synchronous Mode. (Calling functions in sequence A).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.3.12	0xff3f4b6d, 0x2b40, 0x49b5, 0xb9, 0xe0, 0x7e, 0x11, 0x8a, 0x73, 0x70, 0x0a	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** in Dhcp4Init State and synchronous Mode. (Calling functions in sequence B).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.3.13	0x35972f03, 0x90dc, 0x41ae, 0x8e, 0x1e, 0x27, 0x72, 0x47, 0x3b, 0x06, 0xb6	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** in Dhcp4Init State and synchronous Mode. (Calling functions in sequence C).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPNAK packet. The ModeData.State should be Dhcp4Init. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.3.14	0x90924db4, 0x1237, 0x4d59, 0x88, 0xf3, 0x11, 0x8b, 0xed, 0x01, 0x80, 0xae	**EFI_DHCP4_PROTOCOL.Start()** - invokes **Start()** in Dhcp4InitReboot State and synchronous Mode. (Calling functions in sequence A).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.3.15	0x434f1845, 0xd940, 0x4129, 0xaa, 0xeb, 0x7a, 0x1b, 0xe7, 0xe1, 0x39, 0x48	**`EFI_DHCP4_PROTOCOL.Start()`** - invokes **Start()** in Dhcp4InitReboot State and synchronous Mode. (Calling functions in sequence B).	1. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Dhcp4 child. 2. Call **`EFI_DHCP4_PROTOCOL.Configure()`** to configure child with setting *`ClientAddress`* "192.168.2.4". 3. Call **`EFI_DHCP4_PROTOCOL.Start()`** to start the configuration process with a *`CompletionEvent`* value of **`NULL`**. 4. Call **`EFI_DHCP4_PROTOCOL.GetModeData()`** to get Dhcp4 mode data after the driver having stopped DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.3.16	0x340ff4c6, 0x7412, 0x44d4, 0x8f, 0x33, 0xeb, 0xc2, 0x6f, 0x22, 0x1d, 0x0c	**`EFI_DHCP4_PROTOCOL.Start()`** - invokes **Start()** in Dhcp4InitReboot State and synchronous Mode. (Calling functions in sequence C).	1. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Dhcp4 child. 2. Call **`EFI_DHCP4_PROTOCOL.Configure()`** to configure child with setting *`ClientAddress`* "192.168.2.4". 3. Call **`EFI_DHCP4_PROTOCOL.Start()`** to start the configuration process with a *`CompletionEvent`* value of **`NULL`**. 4. Call **`EFI_DHCP4_PROTOCOL.GetModeData()`** to get Dhcp4 mode data after the driver having stopped the DHCPNAK packet. The ModeData.State should be Dhcp4Init. 5. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Dhcp4 child and clean up the environment.

20.2.4 RenewRebind()

Number	GUID	Assertion	Test Description
5.24.2.4.1	0x15bdc212, 0xbad5, 0x4213, 0xb2, 0x38, 0x50, 0xac, 0x76, 0x18, 0xdc, 0x90	**EFI_DHCP4_PROTOCOL.RenewRebind()** - invokes **RenewRebind()** when the driver instance is in the Dhcp4Stopped state.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.RenewRebind()** with a *RebindRequest* value of **TRUE**. The return status should be **EFI_NOT_STARTED**. 3. Call **EFI_DHCP4_PROTOCOL.RenewRebind()** with a *RebindRequest* value of **FALSE**. The return status should be **EFI_NOT_STARTED**. 4. Call **EFI_DHCP4_PROTOCOL.Stop()** to stop the configuration process. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.4.2	0x2949dc87, 0xdbcd, 0x4d64, 0x8f, 0x10, 0x68, 0x2f, 0xa2, 0x27, 0xe0, 0x88	**EFI_DHCP4_PROTOCOL.RenewRebind()** - invokes **RenewRebind()** while getting no response during the specified time.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **EFI_DHCP4_PROTOCOL.RenewRebind()** with a *RebindRequest* value of **FALSE** and a *CompletionEvent* value of **NULL**. The return status should be **EFI_TIMEOUT**. 6. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.4.3	0xd7f4cb11, 0xc3dc, 0x421f, 0x98, 0x80, 0x5c, 0x2a, 0x2d, 0x73, 0x06, 0x02	**EFI_DHCP4_PROTOCOL.RenewRebind()** - invokes **RenewRebind()** when the driver instance is not in the Dhcp4Bound state.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value other than **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having sent the DHCPREQUEST packet. The ModeData.State should be Dhcp4Rebooting. 5. Call **EFI_DHCP4_PROTOCOL.RenewRebind()** with a *RebindRequest* value of **TRUE** and a *CompletionEvent* value of **NULL**. The return status should be **EFI_ACCESS_DENIED**. 6. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.4.4	0x38bb70ba, 0xb05c, 0x4431, 0xb4, 0xf9, 0x8f, 0x4e, 0x9b, 0x10, 0xc7, 0x54	**`EFI_DHCP4_PROTOCOL.RenewRebind()`** - invokes **RenewRebind()** with the driver instance extending lease time in Asynchronous Mode using unicast. (Calling functions in sequence A).	1. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Dhcp4 child. 2. Call **`EFI_DHCP4_PROTOCOL.Configure()`** to configure child with setting *`ClientAddress`* "192.168.2.4". 3. Call **`EFI_DHCP4_PROTOCOL.Start()`** to start the configuration process with a *`CompletionEvent`* value of **`NULL`**. 4. Call **`EFI_DHCP4_PROTOCOL.GetModeData()`** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **`EFI_DHCP4_PROTOCOL.RenewRebind()`** with a *`RebindRequest`* value of **`FALSE`** and a *CompletionEvent* value of not **`NULL`**. The return status should be **`EFI_SUCCESS`**. 6. Call **`EFI_DHCP4_PROTOCOL.GetModeData()`** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 7. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.4.5	0x432ccefe, 0x8586, 0x4358, 0xb7, 0xee, 0xf1, 0x36, 0xe3, 0x8a, 0xd8, 0x30	**EFI_DHCP4_PROTOCOL.RenewRebind()** - invokes **RenewRebind()** with the driver instance extending lease time in Asynchronous Mode using unicast. (Calling functions in sequence B).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **EFI_DHCP4_PROTOCOL.RenewRebind()** with a *RebindRequest* value of **FALSE** and a *CompletionEvent* value of not **NULL**. The return status should be **EFI_SUCCESS**. 6. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Bound. 7. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.4.6	0xc0b17d39, 0x32bb, 0x41f8, 0xbd, 0x44, 0x6b, 0xb8, 0x53, 0x0f, 0xa4, 0xaf	**EFI_DHCP4_PROTOCOL.RenewRebind()** - invokes **RenewRebind()** with the driver instance extending lease time in Asynchronous Mode using broadcast. (Calling functions in sequence A).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **EFI_DHCP4_PROTOCOL.RenewRebind()** with a *RebindRequest* value of **TRUE** and a *CompletionEvent* value of not **NULL**. The return status should be **EFI_SUCCESS**. 6. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 7. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.4.7	0x819f530e, 0x0d51, 0x43ce, 0x83, 0x73, 0x0b, 0x27, 0xc6, 0x36, 0x3b, 0x63	**EFI_DHCP4_PROTOCOL.RenewRebind()** - invokes **RenewRebind()** with the driver instance extending lease time in Asynchronous Mode using broadcast. Sequence B.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **EFI_DHCP4_PROTOCOL.RenewRebind()** with a *RebindRequest* value of **TRUE** and a *CompletionEvent* value of not **NULL**. The return status should be **EFI_SUCCESS**. 6. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Bound. 7. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.4.8	0x982b5d48, 0x2d87, 0x40ea, 0xbe, 0x60, 0x44, 0x60, 0x49, 0xfe, 0x08, 0x98	**EFI_DHCP4_PROTOCOL.RenewRebind()** - invokes **RenewRebind()** with the driver instance extending lease time in synchronous Mode using unicast. Sequence A.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **EFI_DHCP4_PROTOCOL.RenewRebind()** with a *RebindRequest* value of **FALSE** and a *CompletionEvent* value of **NULL**. The return status should be **EFI_SUCCESS**. 6. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 7. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.4.9	0x4cc9abee, 0xd9e8, 0x444b, 0xb8, 0x34, 0x3e, 0xd4, 0x57, 0x96, 0x25, 0xc9	`EFI_DHCP4_PROTOCOL.RenewRebind()` - invokes **RenewRebind()** with the driver instance extending lease time in synchronous Mode using unicast. Sequence B.	1. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Dhcp4 child. 2. Call `EFI_DHCP4_PROTOCOL.Configure()` to configure child with setting *`ClientAddress`* "192.168.2.4". 3. Call `EFI_DHCP4_PROTOCOL.Start()` to start the configuration process with a *`CompletionEvent`* value of not `NULL`. 4. Call `EFI_DHCP4_PROTOCOL.GetModeData()` to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call `EFI_DHCP4_PROTOCOL.RenewRebind()` with a *`RebindRequest`* value of `TRUE` and a *`CompletionEvent`* value of `NULL`. The return status should be `EFI_SUCCESS`. 6. Call `EFI_DHCP4_PROTOCOL.GetModeData()` to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Bound. 7. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.4.10	0x061ca38f, 0x5092, 0x483b, 0xa4, 0xd2, 0xf3, 0x1f, 0x53, 0x3f, 0xe7, 0xac	**EFI_DHCP4_PROTOCOL.RenewRebind()** - invokes **RenewRebind()** with the driver instance extending lease time in synchronous Mode using broadcast. Sequence A.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **EFI_DHCP4_PROTOCOL.RenewRebind()** with a *RebindRequest* value of **FALSE** and a *CompletionEvent* value of **NULL**. The return status should be **EFI_SUCCESS**. 6. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 7. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.4.11	0xf9fa2078, 0x6283, 0x4510, 0xad, 0x21, 0xba, 0xe1, 0x15, 0x21, 0x56, 0xf9	**EFI_DHCP4_PROTOCOL.RenewRebind()** - invokes **RenewRebind()** with the driver instance extending lease time in synchronous Mode using broadcast. Sequence B.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **EFI_DHCP4_PROTOCOL.RenewRebind()** with a *RebindRequest* value of **TRUE** and a *CompletionEvent* value of not **NULL**. The return status should be **EFI_SUCCESS**. 6. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Bound. 7. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

20.2.5 Release()

Number	GUID	Assertion	Test Description
5.24.2.5.1	0xa80fa204, 0x87dd, 0x4e92, 0x8a, 0x5d, 0xee, 0x55, 0x6c, 0x83, 0xac, 0x7c	**EFI_DHCP4_PROTOCOL.Release()** - invokes **Release()** with the driver in the configuration process, but not in the Dhcp4Bound or Dhcp4InitReboot state.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "192.168.2.4". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of not **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.Release()** after Stop the REQUEST packet from the driver. The return status should be **EFI_ACCESS_DENIED**. 5. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.5.2	0x70f9485c, 0x4fef, 0x4bf3, 0xac, 0xd5, 0x2e, 0xe0, 0xba, 0x30, 0x3d, 0xd9	**EFI_DHCP4_PROTOCOL.Release()** - invokes **Release()** with the driver in the Dhcp4Stopped state, but not in the Dhcp4Bound or Dhcp4InitReboot state.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Stop()** to verify the driver in the Dhcp4Stopped state. 3. Call **EFI_DHCP4_PROTOCOL.Release()** when the driver is in the Dhcp4Stopped state. The return status should be **EFI_ACCESS_DENIED**. 4. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.5.3	0x67c1be03, 0xf9c4, 0x4419, 0x88, 0xf0, 0xb9, 0xfc, 0x6c, 0x1a, 0xd2, 0x67	**EFI_DHCP4_PROTOCOL.Release()** - invokes **Release()** when the driver is in the DhcpBound State.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Configure()** to configure child with setting *ClientAddress* "0.0.0.0". 3. Call **EFI_DHCP4_PROTOCOL.Start()** to start the configuration process with a *CompletionEvent* value of **NULL**. 4. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound. 5. Call **EFI_DHCP4_PROTOCOL.Release()** and capture ARPREQUEST packet from the driver, send ARPREPLY packet to the driver, then capture DHCPRELEASE packet from the driver. The return status should be **EFI_SUCCESS**. 6. Call **EFI_DHCP4_PROTOCOL.GetModeData()** to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init. 7. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.5.4	0x555d101b, 0xf86a, 0x4e6f, 0x95, 0x70, 0x1c, 0xfa, 0xe7, 0xd2, 0xd6, 0x8a	`EFI_DHCP4_PROTOCOL.Release()` - invokes **Release()** when the driver is in the DhcpInitReboot State.	1. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Dhcp4 child. 2. Call `EFI_DHCP4_PROTOCOL.Configure()` to configure child with setting *`ClientAddress`* "192.168.2.4". 3. Call `EFI_DHCP4_PROTOCOL.GetModeData()` to get Dhcp4 mode data. The ModeData.State should be DhcpInitReboot. 5. Call `EFI_DHCP4_PROTOCOL.Release()` and capture ARPREQUEST packet from the driver, send ARPREPLY packet to the driver, then capture DHCPRELEASE packet from the driver. The return status should be `EFI_SUCCESS`. 6. Call `EFI_DHCP4_PROTOCOL.GetModeData()` to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init. 7. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Dhcp4 child and clean up the environment.

20.2.6 Stop()

Number	GUID	Assertion	Test Description
5.24.2.6.1	0xda8661a5, 0x82d4, 0x4b1b, 0xa2, 0x68, 0xf3, 0x4f, 0xe5, 0xab, 0x03, 0x57	`EFI_DHCP4_PROTOCOL.Stop()` - invokes **Stop()** when the driver is in the DhcpInitReboot State.	1. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Dhcp4 child. 2. Call `EFI_DHCP4_PROTOCOL.Configure()` to configure child with setting *ClientAddress* "192.168.2.4". 3. Call `EFI_DHCP4_PROTOCOL.GetModeData()` to get Dhcp4 mode data after receiving REQUEST packet from the driver. The ModeData.State should be DhcpInitReboot. 4. Call `EFI_DHCP4_PROTOCOL.Stop()` to stop the configuration process. The return status should be `EFI_SUCCESS`. 5. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.6.2	0x0f6193fc, 0x21f7, 0x4831, 0xbf, 0x53, 0x39, 0x28, 0xc0, 0x49, 0x6b, 0x48	**`EFI_DHCP4_PROTOCOL.Stop()`** - invokes **Stop()** when the driver is in the configuration process.	1. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Dhcp4 child. 2. Call **`EFI_DHCP4_PROTOCOL.Configure()`** to configure child with setting *`ClientAddress`* "192.168.2.4". 3. Call **`EFI_DHCP4_PROTOCOL.Start()`** to start the configuration process with a *`CompletionEvent`* value other than **`NULL`**. 4. Call **`EFI_DHCP4_PROTOCOL.GetModeData()`** to get Dhcp4 mode data after receiving REQUEST packet from the driver. The ModeData.State should be Dhcp4Rebooting. 5. Call **`EFI_DHCP4_PROTOCOL.Stop()`** to stop the configuration process. The return status should be **`EFI_SUCCESS`**. 6. Call **`EFI_DHCP4_PROTOCOL.GetModeData()`** to get Dhcp4 mode data. The ModeData.State should be Dhcp4Stopped. 7. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Dhcp4 child and clean up the environment.

20.2.7 Build()

Number	GUID	Assertion	Test Description
5.24.2.7.1	0xc2aa2960, 0xdd52, 0x4e56, 0x87, 0x7e, 0x8c, 0x44, 0x6a, 0x5e, 0xea, 0x31	**EFI_DHCP4_PROTOCOL.Build()** - invokes **Build()** when the parameter *SeedPacket* is **NULL**.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Build()** with a *SeedPacket* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.7.2	0xf19cc8c3, 0x9a84, 0x4d62, 0x94, 0xae, 0xc3, 0x4b, 0x06, 0x3a, 0xea, 0x91	**EFI_DHCP4_PROTOCOL.Build()** - invokes **Build()** when the parameter *SeedPacket* is not a well-formed DHCP packet (Magic Number Error).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Build()** with a SeedPacket.**EFI_DHCP4_PROTOCOL**.Magik value of error magic cookie. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.7.3	0xc650067b, 0x4ab0, 0x4170, 0x9b, 0x4b, 0x4f, 0x7a, 0xeb, 0x77, 0xc0, 0x5e	**EFI_DHCP4_PROTOCOL.Build()** - invokes **Build()** when the parameter *AppendCount* is not 0 and *AppendList* is **NULL**.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Build()** with a *AppendCount* value other than **NULL** and *AppendList* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.7.4	0x1debfafe, 0xdfbe, 0x4ff5, 0x8a, 0xcd, 0x8f, 0xe1, 0x11, 0x82, 0x30, 0xe0	**EFI_DHCP4_PROTOCOL.Build()** - invokes **Build()** when the parameter *DeleteCount* is not 0 and *DeleteList* is **NULL**.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Build()** with a *DeleteCount* value of **NULL** and a *DeleteList* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.7.5	0xd0beca24, 0xa8f3, 0x4753, 0x8c, 0xdb, 0x96, 0xe6, 0x00, 0x92, 0x78, 0x47	**EFI_DHCP4_PROTOCOL.Build()** - invokes **Build()** when the parameter *NewPacket* is **NULL**.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Build()** with a *NewPacket* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.7.6	0x7d05c782, 0xccf3, 0x42d0, 0x9a, 0x6e, 0x0d, 0x6b, 0x5c, 0x8d, 0x9c, 0x20	**EFI_DHCP4_PROTOCOL.Build()** - invokes **Build()** when the parameter both *DeleteCount* and *OptionCount* are 0 and *NewPacket* is not **NULL**.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Build()** with both the *DeleteCount* and *OptionCount* value of 0 and a *NewPacket* value other than **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.7.7	0xf52d8032, 0xd5c6, 0x48e1, 0x86, 0xb0, 0xac, 0x47, 0xae, 0x82, 0x93, 0xed	**EFI_DHCP4_PROTOCOL.Build()** - invokes **Build()** when the parameter *AppendCount* and *AppendList* are not **NULL**, and build a new packet with DHCP options appended.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Build()** with both the *AppendCount* and *AppendList* value other than **NULL**. The return status should be **EFI_SUCCESS**. 3. Call **EFI_DHCP4_PROTOCOL.Parse()** to parse the packet returned by the parameter *NewPacket* of **EFI_DHCP4_PROTOCOL.Build()**. The *NewPacket* should include the DHCP options matching the parameter *AppendList* of **EFI_DHCP4_PROTOCOL.Build()**. 4. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.7.8	0x78dae7e2, 0x579a, 0x47a1, 0xb2, 0x45, 0x8c, 0xad, 0x39, 0xc8, 0x07, 0x27	**EFI_DHCP4_PROTOCOL.Build()** - invokes **Build()** to delete defined options.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Build()** with both the *DeleteCount* and *DeleteList* value other than **NULL**. The return status should be **EFI_SUCCESS**. 3. Call **EFI_DHCP4_PROTOCOL.Parse()** to parse the packet returned by the parameter *NewPacket* of **EFI_DHCP4_PROTOCOL.Build()**. The *NewPacket* should not include the DHCP options matching the parameter *DeleteList* of **EFI_DHCP4_PROTOCOL.Build()**. 4. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.7.9	0xfc1f9cb7, 0xed3d, 0x4e6d, 0x93, 0x2a, 0x63, 0xb5, 0xcf, 0xb4, 0xb3, 0x37	**EFI_DHCP4_PROTOCOL.Build()** - invokes **Build()** to delete an undefined option.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Build()** with both the *DeleteCount* and *DeleteList* value other than **NULL**, and *DeleteList* include an undefined option. The return status should be **EFI_SUCCESS**. 3. Call **EFI_DHCP4_PROTOCOL.Parse()** to parse the packet returned by the parameter *NewPacket* of **EFI_DHCP4_PROTOCOL.Build()**. The *NewPacket* should not include the DHCP options matching the parameter *DeleteList* of **EFI_DHCP4_PROTOCOL.Build()**. 4. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

20.2.8 TransmitReceive()

Number	GUID	Assertion	Test Description
5.24.2.8.1	0x6d1bb6a7, 0x5d67, 0x4982, 0x96, 0x35, 0x54, 0xeb, 0x4b, 0x0c, 0xfa, 0xd5	**EFI_DHCP4_PROTOCOL.TransmitReceive()** - invokes **TransmitReceive()** when the parameter *RemoteAddress* is 0.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.TransmitReceive()** with a *RemoteAddress* value of 0. The return status should be **EFI_UNSUPPORTED**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.8.2	0xd2bec02f, 0x8304, 0x4713, 0x8a, 0x95, 0x4b, 0xd3, 0x4c, 0x69, 0x89, 0xc0	**EFI_DHCP4_PROTOCOL.TransmitReceive()** - invokes **TransmitReceive()** when the parameter *Packet* is **NULL**.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.TransmitReceive()** with a *Packet* value of **NULL**. The return status should be **EFI_UNSUPPORTED**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.8.3	0x9dfd549b, 0x59eb, 0x4f5d, 0x99, 0x5f, 0xb8, 0x2d, 0xdd, 0x18, 0x02, 0xba	**EFI_DHCP4_PROTOCOL.TransmitReceive()** - invokes **TransmitReceive()** when the parameter *Packet* is not a well-formed DHCP packet(Magic Number error).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.TransmitReceive()** with a *Packet* value of not a well-formed DHCP packet(Magic Number error). The return status should be **EFI_UNSUPPORTE**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.8.4	0xce99ae23, 0x910a, 0x4818, 0xa0, 0x89, 0xf3, 0xf4, 0x5b, 0xc5, 0xeb, 0xa8	**EFI_DHCP4_PROTOCOL.TransmitReceive()** - invokes **TransmitReceive()** when the transaction ID in *Packet* is in used by another DHCP process.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.TransmitReceive()** when the transaction ID in *Packet* is in use by another DHCP process. The return status should be **EFI_UNSUPPORTED**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.8.5	0xbe6683bd, 0x807a, 0x4fb0, 0xbc, 0x7b, 0xf7, 0x51, 0x07, 0x0e, 0x0e, 0x66	`EFI_DHCP4_PROTOCOL.TransmitReceive()` - invokes `TransmitReceive()` when the previous call to this function has not finished yet. Try to call this function after collection process completed.	1. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Dhcp4 child. 2. Call `EFI_DHCP4_PROTOCOL.TransmitReceive()` with the previous call to this function not finished yet. The return status should be `EFI_UNSUPPORTE`. 3. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Dhcp4 child and clean up the environment.

20.2.9 Parse()

Number	GUID	Assertion	Test Description
5.24.2.9.1	0x7cca1a2c, 0x4136, 0x4ff0, 0xbc, 0x22, 0xca, 0x80, 0x56, 0x8d, 0xfd, 0xbf	`EFI_DHCP4_PROTOCOL.Parse()` - invokes `Parse()` when the parameter *`Packet`* is `NULL`.	1. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Dhcp4 child. 2. Call `EFI_DHCP4_PROTOCOL.Parse()` with a *`Packet`* value of `NULL`. The return status should be `EFI_INVALID_PARAMETER`. 3. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Dhcp4 child and clean up the environment.
5.24.2.9.2	0x225ddf1b, 0x9fb9, 0x4a9b, 0xb3, 0xb6, 0xca, 0x25, 0xeb, 0x31, 0x0d, 0xbb	`EFI_DHCP4_PROTOCOL.Parse()` - invokes `Parse()` when the parameter *`OptionCount`* is `NULL`.	1. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Dhcp4 child. 2. Call `EFI_DHCP4_PROTOCOL.Parse()` with a *`OptionCount`* value of `NULL`. The return status should be `EFI_INVALID_PARAMETER`. 3. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.9.3	0xea1a95dd, 0xdb6c, 0x4200, 0xb7, 0xc7, 0x19, 0xb0, 0xa3, 0x81, 0x06, 0x5d	**EFI_DHCP4_PROTOCOL.Parse()** - invokes **Parse()** when the *Packet* is not a well-formed DHCP packet (Magic Number error).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Parse()** with a *Packet* value other than a well-formed DHCP packet (Magic Number error). The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.9.4	0x91e4d243, 0x4ed6, 0x451a, 0xb0, 0x9c, 0x0a, 0x35, 0x6a, 0x06, 0x1d, 0xda	**EFI_DHCP4_PROTOCOL.Parse()** - invokes **Parse()** when the *Packet* is not well-formed DHCP packet (No End option).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Parse()** with a *Packet* value other than a well-formed DHCP packet (No End option). The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.9.5	0xd836cddd, 0x6bb4, 0x455e, 0x9e, 0xc4, 0x49, 0x9f, 0xc3, 0x27, 0xdd, 0x21	**EFI_DHCP4_PROTOCOL.Parse()** - invokes **Parse()** when the *Packet* is not a well-formed DHCP packet (Length < Header Size).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Parse()** with a *Packet* value other than a well-formed DHCP packet (Length < Header Size). The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.9.6	0xed5c8f2b, 0x0043, 0x4f43, 0xae, 0x83, 0xa6, 0xbf, 0xab, 0x5b, 0xa2, 0xba	**EFI_DHCP4_PROTOCOL.Parse()** - invokes **Parse()** when the *Packet* is not a well-formed DHCP packet (Size < Length).	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Parse()** with a *Packet* value other than a well-formed DHCP packet (Size < Length). The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.9.7	0x4bd82a66, 0xcede, 0x4132, 0xa8, 0xca, 0xd9, 0x95, 0xe8, 0xe7, 0x9a, 0xb2	**EFI_DHCP4_PROTOCOL.Parse()** - invokes **Parse()** when the parameter *OptionCount* is smaller than the number of options that were found in the *Packet*.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Parse()** with the parameter **OptionCount** smaller than the number of options that were found in the *Packet*. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.9.8	0xa73ac67a, 0xe5c9, 0x41e7, 0xb6, 0xc0, 0x80, 0xa2, 0x6f, 0x27, 0x7e, 0xc0	**EFI_DHCP4_PROTOCOL.Parse()** - invokes *Parse()* when the parameter *PacketOptionList* is **NULL**.	1. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Dhcp4 child. 2. Call **EFI_DHCP4_PROTOCOL.Parse()** with a *PacketOptionList* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.24.2.9.9	0xc84a412c, 0x702a, 0x40e1, 0xa3, 0x9c, 0x55, 0xa8, 0x8c, 0xbe, 0x60, 0x5a	**`EFI_DHCP4_PROTOCOL.Parse()`** - invokes **Parse()** when options exist in packet.	1. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Dhcp4 child. 2. Call **`EFI_DHCP4_PROTOCOL.Parse()`** to check the *`PacketOptionList`* when options exist in packet. The return status should be **`EFI_SUCCESS`**. 3. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.9.10	0x2ba25811, 0x4069, 0x45da, 0xb3, 0x9e, 0xfa, 0x05, 0x14, 0x42, 0x4a, 0x4c	**`EFI_DHCP4_PROTOCOL.Parse()`** - invokes **Parse()** when no options exist in packet	1. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Dhcp4 child. 2. Call **`EFI_DHCP4_PROTOCOL.Parse()`** to check the *`PacketOptionList`* when no options exist in packet. The return status should be **`EFI_SUCCESS`**. 3. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Dhcp4 child and clean up the environment.
5.24.2.9.11	0x6ce744e5, 0x9e5a, 0x4fb5, 0xa5, 0xf2, 0x3b, 0xe8, 0xf5, 0xb5, 0xad, 0x42	**`EFI_DHCP4_PROTOCOL.Parse()`** - invokes **Parse()** with Pad Option included in packet	1. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Dhcp4 child. 2. Call **`EFI_DHCP4_PROTOCOL.Parse()`** to check the *`PacketOptionList`* with Pad Option included in packet. The return status should be **`EFI_SUCCESS`**. 3. Call **`EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Dhcp4 child and clean up the environment.

20.2.10 CreateChild()

Number	GUID	Assertion	Test Description

5.24.2.10.1	0x4b66733f, 0xd324, 0x4af9, 0x9d, 0x92, 0x91, 0x4f, 0x5f, 0x77, 0x2e, 0xf0	`EFI_DHCP4_PROTOCOL.CreateChild()` - invokes CreateChild() when Child Handle is `NULL`.	1. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Dhcp4 child with `NULL` Handle Pointer. The return status should be `EFI_INVALID_PATAMETER`.
5.24.2.10.2	0x1e0f5047, 0x1be9, 0x4db0, 0xa5, 0x71, 0xfc, 0x82, 0xbc, 0x2d, 0x0a, 0x06	`EFI_DHCP4_PROTOCOL.CreateChild()` - to test the function of `CreateChild()`.	Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create childs three times and then destroy them.

20.2.11 DestroyChild()

Number	GUID	Assertion	Test Description
5.24.2.11.1	0x1f92470a, 0x7aec, 0x4fb4, 0xa4, 0x0d, 0x5f, 0x0c, 0xd2, 0x40, 0x1f, 0x08	`EFI_DHCP4_PROTOCOL.DestroyChild()` – invokes `DestroyChild()` when Call this function twice.	1. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Dhcp4 child. 2. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Dhcp4 child. The return status should be `EFI_SUCCESS`. 3. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Dhcp4 child again. The return status should be `EFI_UNSUPPORTED`.
5.24.2.11.2	0x06b43e55, 0xd8af, 0x494f, 0x8b, 0x93, 0x78, 0xf8, 0xd0, 0x7a, 0xa4, 0xc8	`EFI_DHCP4_PROTOCOL.DestroyChild()` – invokes `DestroyChild()` when Child Handle is `NULL`.	1. Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to Destroy a Dhcp4 child with `NULL` Handle Pointer. The return status should be `EFI_INVALID_PATAMETER`.
5.24.2.11.3	0xc44a4b68, 0x1f16, 0x4098, 0xb2, 0x6d, 0x2c, 0x43, 0xcb, 0x27, 0x4d, 0xae	`EFI_DHCP4_PROTOCOL.DestroyChild()` – to test the function of `DestroyChild()`.	Call `EFI_DHCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly three created Dhcp4 childs.

20.3 EFI_DHCP6_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_DHCP6_PROTOCOL Section.

20.3.1 CreateChild()

Number	GUID	Assertion	Test Description
5.24.3.1.1	0xbd25610a, 0xa4b3, 0x412a, 0xbf, 0x03, 0xb0, 0xf7, 0xce, 0x80, 0x98, 0xbf	**EFI_DHCP6_SERVICE_BINDING_PROTOCOL.CreateChild() - CreateChild()** returns **EFI_INVALID_PARAMETER** with a **NULL** *ChildHandle*.	Call **CreateChild()** with a **NULL** *ChildHandle*, the return status should be **EFI_INVALID_PARAMETER**.
5.24.3.1.2	0xcbf5cb1d, 0xd74d, 0x45bc, 0x94, 0xd2, 0x72, 0xda, 0x7f, 0xf7, 0xbe, 0xda	**EFI_DHCP6_SERVICE_BINDING_PROTOCOL.CreateChild() - CreateChild()** returns **EFI_SUCCESS** with the 1st valid *ChildHandle*.	5.24.3.1.2 to 5.24.3.1.5 belong to one case. 1. Call **CreateChild()** with the 1st valid *ChildHandle*, the return status should be **EFI_SUCCESS**.
5.24.3.1.3	0xb9cfe63d, 0x2cc2, 0x4940, 0xb3, 0x01, 0x39, 0x22, 0xf3, 0xff, 0xdd, 0x35	**EFI_DHCP6_SERVICE_BINDING_PROTOCOL.CreateChild() - CreateChild()** returns **EFI_SUCCESS** with the 2nd valid *ChildHandle*.	2. Call **CreateChild()** with the 2nd valid *ChildHandle*, the return status should be **EFI_SUCCESS**.
5.24.3.1.4	0x2336ebe8, 0x4934, 0x4a6c, 0xae, 0x72, 0x06, 0x73, 0xb6, 0x7a, 0xa0, 0xa6	**EFI_DHCP6_SERVICE_BINDING_PROTOCOL.DestroyChild() - DestroyChild()** returns **EFI_SUCCESS** with the 2nd valid *ChildHandle*.	3. Call **DestroyChild()** with the 2nd valid *ChildHandle*, the return status should be **EFI_SUCCESS**.
5.24.3.1.5	0x0fe6555e, 0x3487, 0x4989, 0x89, 0x96, 0x18, 0xa7, 0x2a, 0x71, 0x52, 0xd5	**EFI_DHCP6_SERVICE_BINDING_PROTOCOL.DestroyChild() - DestroyChild()** returns **EFI_SUCCESS** with the 1st valid *ChildHandle*.	4. Call **DestroyChild()** with the 1st valid *ChildHandle*, the return status should be **EFI_SUCCESS**.

20.3.2 DestroyChild ()

Number	GUID	Assertion	Test Description
5.24.3.2.1	0x6e2206aa, 0xbee7, 0x4f16, 0xa7, 0xaa, 0x71, 0x54, 0xa2, 0xe9, 0x63, 0x65	`EFI_DHCP6_SERVICE_BINDING_PROTOCOL.DestroyChild() - DestroyChild()` returns `EFI_INVALID_PARAMETER` with a `NULL` *`ChildHandle`*.	Call `DestroyChild()` with a `NULL` *`ChildHandle`*, the return status should be `EFI_INVALID_PARAMETER`.
5.24.3.2.2	0x061893a7, 0x48de, 0x431a, 0xad, 0x5b, 0x56, 0x29, 0xb6, 0x9c, 0xe6, 0xce	`EFI_DHCP6_SERVICE_BINDING_PROTOCOL.DestroyChild() - DestroyChild()` returns `EFI_UNSUPPORTED` with a *`ChildHandle`* which has been destroyed.	Call `DestroyChild()` with a *`ChildHandle`* which has been destroyed, the return status should be `EFI_UNSUPPORTED`.

20.3.3 GetModeData()

Number	GUID	Assertion	Test Description
5.24.3.3.1	0x3678146a, 0x0596, 0x4661, 0x8e, 0x53, 0xf6, 0x61, 0xa6, 0xec, 0xe2, 0xf3	`EFI_DHCP6 PROTOCOL.GetModeData() - GetModeData()` returns `EFI_ACCESS_DENIED` with an instance which has not been configured.	Call `GetModeData()` with an instance which has not been configured, The return status should be `EFI_ACCESS_DENIED`.
5.24.3.3.2	0xf58195a9, 0x1924, 0x4490, 0x95, 0x4b, 0x17, 0x75, 0xfc, 0x1c, 0xbf, 0xb0	`EFI_ DHCP6 PROTOCOL.GetModeData() - GetModeData()` returns `EFI_INVALID_PARAMETER` with `NULL` `Dhcp6ConfigData` and `Dhcp6ModeData`	Call `GetModeData()` with `NULL` `Dhcp6ConfigData` and `Dhcp6ModeData,` The return status should be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.24.3.3.3	0x99d01c9a, 0x2bd6, 0x442f, 0x8f, 0xe5, 0xda, 0x8a, 0xa6, 0x88, 0x27, 0x29	`Dhcp6CfgData.CallbackContext` should be 5.	5.24.3.3.3 to 5.24.3.1.13 belong to one case. 1. Call `CreateChild()` to create an DHCP6 instance. 2. Create an event for the Dhcp6CfgData 3. Call `Configure()` to initialize the DHCP6 instance. 4. Call `Start()` to start S.A.R.R process. 5. The `Dhcp6CfgData.CallbackContext` should be 5. The reason is Callback() is called by SendSolict/RcvdAdvertise/ SelectAdvertise/SendRequest/ RcvdReply. Callback() add `Dhcp6CfgData.CallbackContext` with 1 each time.
5.24.3.3.4	0x46993cb1, 0xfb2c, 0x44b3, 0xad, 0xe1, 0x7e, 0xa1, 0xe8, 0x43, 0xbd, 0x2e	`Dhcp6CfgData.IaInfoEvent` should be signaled.	6. When `Start`() return, the `Dhcp6CfgData.IaInfoEvent` should be signaled.
5.24.3.3.5	0x6a6bd40b, 0xb963, 0x4313, 0x8b, 0x4f, 0x45, 0x0e, 0x11, 0x4b, 0x6e, 0xeb	`EFI_ DHCP6 PROTOCOL.GetModeData() - GetModeData()` returns `EFI_SUCCESS` with `Dhcp6ConfigData` and `Dhcp6ModeData`	7. Call `GetModeData()` with `Dhcp6ConfigData` and `Dhcp6ModeData`, The return status should be `EFI_INVALID_PARAMETER`.
5.24.3.3.6	0x24694dfa, 0x5cc6, 0x4358, 0x9a, 0x14, 0x5c, 0xf2, 0x5e, 0x3a, 0x1a, 0xa4	`Dhcp6ModeData.Ia.State` should be `Dhcp6Bound`	8. `Dhcp6ModeData.Ia.State` should be `Dhcp6Bound`
5.24.3.3.7	0x6a19ff82, 0x9ea9, 0x44c1, 0xb8, 0x71, 0x69, 0x05, 0xfe, 0x18, 0x58, 0xbc	`Dhcp6ConfigData.OptionCount` should be same with configured the value.	9. `Dhcp6ConfigData.OptionCount` should be same with the configured value.

Number	GUID	Assertion	Test Description
5.24.3.3.8	0x9fa4ae6e, 0x82b3, 0x4ed7, 0xb9, 0xfc, 0x69, 0x0f, 0x0a, 0x98, 0xe5, 0xde	`Dhcp6ConfigData.OptionList` should be same with configured the value.	10. `Dhcp6ConfigData.OptionList` should be same with the configured value.
5.24.3.3.9	0xa803b115, 0x47b7, 0x496f, 0x95, 0xdb, 0x38, 0xf2, 0x3e, 0x27, 0x3c, 0x20	`Dhcp6ConfigData.IaDescriptor` should be same with configured the value.	11. `Dhcp6ConfigData.IaDescriptor` should be same with the configured value.
5.24.3.3.10	0x2e4a61f7, 0x3a07, 0x4dd9, 0x8b, 0xf6, 0xc3, 0xef, 0xbb, 0x35, 0xb7, 0x90	`Dhcp6ConfigData.IaInfoEvent` should be same with configured the value.	12. `Dhcp6ConfigData.IaInfoEvent` should be same with the configured value.
5.24.3.3.11	0x32797b99, 0x3b8b, 0x4456, 0x9d, 0xca, 0x3f, 0x76, 0xc6, 0x3f, 0x1c, 0xbf	`Dhcp6ConfigData.ReconfigureAccept` should be same with configured the value.	13. `Dhcp6ConfigData.ReconfigureAccept` should be same with the configured value.
5.24.3.3.12	0xb2f4a83b, 0xe44d, 0x4770, 0x81, 0xef, 0xef, 0x06, 0x29, 0xbd, 0x7f, 0xd7	`Dhcp6ConfigData.RapidCommit` should be same with configured the value.	14. `Dhcp6ConfigData.RapidCommit` should be same with the configured value.
5.24.3.3.13	0x45ea153f, 0x2d5f, 0x40b4, 0xbd, 0x34, 0x04, 0x52, 0x27, 0xd9, 0xb5, 0xc3	`Dhcp6ConfigData.SolicitRetransmission` should be same with configured the value.	15. `Dhcp6ConfigData.SolicitRetransmission` should be same with the configured value.

20.3.4 Configure()

Number	GUID	Assertion	Test Description
5.24.3.4.1	0x8aa05b75, 0x4bdf, 0x45e6, 0x81, 0x74, 0x21, 0x85, 0x55, 0x88, 0x19, 0x74	`EFI_ DHCP6 PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` with `Dhcp6ConfigData.OptionCount` > 0 and `Dhcp6ConfigData.OptionList` is `NULL`	Call `Configure()` with `Dhcp6ConfigData.OptionCount` > 0 and `Dhcp6ConfigData.OptionList` is `NULL`, The return status should be `EFI_INVALID_PARAMETER.`
5.24.3.4.2	0xee84c2d5, 0xda69, 0x45ca, 0x9b, 0x62, 0x6c, 0x5f, 0x9a, 0xd9, 0x0d, 0xe2	`EFI_ DHCP6 PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` with `OptionList` containing ClientId option.	Call `Configure()` with `OptionList` containing ClientId option, The return status should be `EFI_INVALID_PARAMETER.`
5.24.3.4.3	0xd6cda19e, 0xcec6, 0x458a, 0xb9, 0xc7, 0x9d, 0x5e, 0xc8, 0x3d, 0xdd, 0x3f	`EFI_ DHCP6 PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` with `OptionList` containing ReconfigAccept option.	Call `Configure()` with `OptionList` containing ReconfigAccept option, The return status should be `EFI_INVALID_PARAMETER.`
5.24.3.4.4	0x8a694b28, 0x7d56, 0x4171, 0xa9, 0x91, 0x07, 0x89, 0x56, 0x08, 0xf3, 0xb2	`EFI_ DHCP6 PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` with `OptionList` containing RapidCommit option.	Call `Configure()` with `OptionList` containing RapidCommit option, The return status should be `EFI_INVALID_PARAMETER.`
5.24.3.4.5	0x671c33eb, 0x66ab, 0x46db, 0xac, 0x12, 0xb6, 0x41, 0xca, 0xf3, 0xc2, 0xad	`EFI_ DHCP6 PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` with `OptionList` containing IA for Non-temporary Addresses Option.	Call `Configure()` with `OptionList` containing IA for Non-temporary Addresses Option, The return status should be `EFI_INVALID_PARAMETER.`

Number	GUID	Assertion	Test Description
5.24.3.4.6	0x438764a3, 0x3419, 0x48c1, 0xbc, 0xb6, 0xa7, 0x82, 0x21, 0xaf, 0x4d, 0xb7	EFI_ DHCP6 PROTOCOL.Configure() - Configure() returns EFI_INVALID_PARAMETER with OptionList containing IA for temporary Addresses Option.	Call Configure() with OptionList containing IA for temporary Addresses Option, The return status should be EFI_INVALID_PARAMETER.
5.24.3.4.7	0x4ae68d37, 0x1f81, 0x41a9, 0xbf, 0x5a, 0xf7, 0x5f, 0xd6, 0xcf, 0x04, 0x11	EFI_ DHCP6 PROTOCOL.Configure() - Configure() returns EFI_INVALID_PARAMETER with an invalid IaDescriptor.Type (neither EFI_DHCP6_IA_TYPE_NA nor EFI_DHCP6_IA_TYPE_TA).	Call Configure() with an invalid IaDescriptor.Type (neither EFI_DHCP6_IA_TYPE_NA nor EFI_DHCP6_IA_TYPE_TA), The return status should be EFI_INVALID_PARAMETER.
5.24.3.4.8	0xffb74292, 0x6403, 0x4e09, 0xb3, 0x83, 0xe9, 0xa8, 0x14, 0x98, 0x54, 0xfa	EFI_ DHCP6 PROTOCOL.Configure() - Configure() returns EFI_INVALID_PARAMETER with an IaDescriptor is not unique.	Call Configure() with an IaDescriptor is not unique, The return status should be EFI_INVALID_PARAMETER.
5.24.3.4.9	0x286b8508, 0x13bc, 0x44cc, 0xaa, 0x6a, 0xc2, 0xd9, 0xac, 0xc7, 0xeb, 0x49	EFI_ DHCP6 PROTOCOL.Configure() - Configure() returns EFI_INVALID_PARAMETER with both IaInfoEvent and SolicitRetransmission NULL.	Call Configure() with both IaInfoEvent and SolicitRetransmission NULL, The return status should be EFI_INVALID_PARAMETER.
5.24.3.4.10	0xc74fd682, 0x5e75, 0x455d, 0xbf, 0xc2, 0x28, 0xe0, 0xf3, 0x54, 0x34, 0xfa	EFI_ DHCP6 PROTOCOL.Configure() - Configure() returns EFI_INVALID_PARAMETER with a non NULL SolicitRetransmission while Mrc and Mrd are zero.	Call Configure() with a non NULL Dhcp6ConfigData while Mrc and Mrd are zero, The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.24.3.4.11	0x49935e3b, 0xe516, 0x423f, 0xa9, 0xb1, 0x99, 0x97, 0xea, 0xd4, 0x1c, 0x96	`EFI_ DHCP6 PROTOCOL.Configure( ) - Configure()` returns `EFI_ACCESS_DENIED` with a non `NULL` `Dhcp6ConfigData` while the instance has already been configured.	Call `Configure()` with a non `NULL` `Dhcp6ConfigData` while the instance has already been configured, The return status should be `EFI_ACCESS_DENIED.`
5.24.3.4.12	0x59090898, 0x378c, 0x4555, 0xa6, 0xab, 0x14, 0x10, 0x96, 0xdc, 0x4f, 0xde	`EFI_ DHCP6 PROTOCOL.Configure( ) - Configure()` returns `EFI_SUCCESS` with a valid `Dhcp6ConfigData`	5.24.3.4.12 to 5.24.3.4.15 belong to one case. 1. Call `Configure()` with a valid `Dhcp6ConfigData`, The return status should be `EFI_SUCCESS.`
5.24.3.4.13	0x568406ba, 0xa297, 0x4917, 0x8e, 0x7f, 0x77, 0xbb, 0x73, 0x6b, 0x53, 0xae	`EFI_ DHCP6 PROTOCOL.Configure( ) - Configure()` returns `EFI_SUCCESS` with a `NULL` `Dhcp6ConfigData`	2. Call `Configure()` with a `NULL` `Dhcp6ConfigData`, The return status should be `EFI_SUCCESS.`
5.24.3.4.14	0x670d8a4d, 0x57e4, 0x424a, 0xbb, 0x72, 0x02, 0xb6, 0x72, 0xb0, 0x2d, 0x78	`Dhcp6ModeData.ClientId` should not be 0.	3. Call `GetModeData()` to get `GetModeData.` 4. `Dhcp6ModeData.ClientId` should not be 0.
5.24.3.4.15	0x93080b8e, 0x5908, 0x4c54, 0x8d, 0xa7, 0xf6, 0x73, 0x2c, 0x66, 0x68, 0x92	`Dhcp6ModeData.Ia` should be 0.	5. `Dhcp6ModeData.Ia` should be 0.

20.3.5 Start()

Number	GUID	Assertion	Test Description
5.24.3.5.1	0x2153bcbb, 0xd5d3, 0x487e, 0x80, 0x98, 0xea, 0x02, 0x22, 0x79, 0x60, 0x11	EFI_ DHCP6 PROTOCOL.Start() – Start() returns EFI_ACCESS_DENIED with the non configured instance.	Call Start() with the non configured instance, The return status should be EFI_ACCESS_DENIED.
5.24.3.5.2	0x5b1e8f26, 0x72e7, 0x429a, 0xbc, 0xbd, 0xff, 0xd0, 0x27, 0x91, 0x8a, 0x35	EFI_ DHCP6 PROTOCOL.Start() – Start() returns EFI_ALREADY_STARTED with the configured instance which has been started.	Call Start() with the configured instance which has been started, The return status should be EFI_ALREADY_STARTED.
5.24.3.5.3	0xc5eca119, 0x7635, 0x4c13, 0x98, 0x5d, 0xde, 0xed, 0xf6, 0x94, 0x83, 0x37	EFI_ DHCP6 PROTOCOL.Start() – Start() returns EFI_NO_RESPONSE while DHCPv6 S.A.R.R process failed because of no response.	Call Start() while DHCPv6 S.A.R.R process failed because of no response, The return status should be EFI_NO_RESPONSE.
5.24.3.5.4	0x23731450, 0xf84f, 0x43cc, 0xa6, 0x2a, 0x87, 0x6c, 0x10, 0xb7, 0xb2, 0x08	EFI_ DHCP6 PROTOCOL.Start() – Start() returns EFI_ABORTED when the user returns error status from callback function.	Call Configure() when the user returns error status from callback function, The return status should be EFI_ABORTED.
5.24.3.5.5	0xd5a092e9, 0xed43, 0x4e5e, 0x8d, 0x9f, 0xc9, 0xc4, 0x92, 0x65, 0x27, 0xce	EFI_ DHCP6 PROTOCOL.Start() - Start() returns EFI_SUCCESS when the S.A.R.R process successfully.	5.24.3.5.5 to 5.24.3.5.7 belong to one case. 1. Call CreateChild() to create an DHCP6 instance. 2. Create an event for the Dhcp6CfgData 3. Call Configure() to initialize the DHCP6 instance. 4. Call Start() to start S.A.R.R process. 5. Get the return status of Start(), it should be EFI_SUCCESS
5.24.3.5.6	0xbb8655d9, 0x8d41, 0x452a, 0x92, 0x6e, 0xc8, 0xe7, 0x92, 0xf8, 0xc4, 0xcc	GetModeData.Ia.State should be Dhcp6Bound.	6. Call GetModeData() to get the GetModeData 7. GetModeData.Ia.State should be Dhcp6Bound

Number	GUID	Assertion	Test Description
5.24.3.5.7	0xb7d13d3b, 0x6492, 0x4955, 0x9d, 0x51, 0xe0, 0xba, 0x96, 0x69, 0xfd, 0x43	Dhcp6ConfigData.IaInfoEvent should be signaled.	8. Dhcp6ConfigData.IaInfoEvent should be signaled
5.24.3.5.8	0x6e3cc768, 0x1a9c, 0x466f, 0xa6, 0x0f, 0xac, 0xd4, 0x58, 0x76, 0xdb, 0x7f	EFI_ DHCP6 PROTOCOL.Start() - Start() returns EFI_SUCCESS when the S.A.R.R process successfully.	5.24.3.5.8 to 5.24.3.5.9 belong to one case. 1. Call CreateChild() to create an DHCP6 instance. 2. Call Configure() to initialize the DHCP6 instance. 3. Call Start() to start S.A.R.R process. 4. Get the return status of Start(), it should be EFI_SUCCESS
5.24.3.5.9	0xf68a6461, 0x26cf, 0x4f37, 0xa5, 0xd2, 0x65, 0xb2, 0x65, 0xd1, 0x1a, 0x84	EFI_ DHCP6 PROTOCOL.Configure() - Configure() returns EFI_INVALID_PARAMETER with both IaInfoEvent and SolicitRetransmission NULL.	5. Call GetModeData() to get the GetModeData 6. GetModeData.Ia.State should be Dhcp6Bound

20.3.6 InfoRequest()

Number	GUID	Assertion	Test Description
5.24.3.6.1	0x5bf750bc, 0x349f, 0x4aa2, 0xa5, 0x9f, 0xfd, 0x09, 0xba, 0xf0, 0xcf, 0xc1	EFI_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest() returns EFI_INVALID_PARAMETER with NULL OptionRequest.	Call InfoRequest() with NULL OptionRequest, The return status should be EFI_INVALID_PARAMETER.
5.24.3.6.2	0x3e90fc45, 0x7a27, 0x4c9b, 0x88, 0x8b, 0xfc, 0xa8, 0x56, 0x9f, 0x80, 0xef	EFI_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest() returns EFI_INVALID_PARAMETER with non zero OptionCount and an NULL OptionList.	Call InfoRequest() with non zero OptionCount and an NULL OptionList, The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.24.3.6.3	0xa85f59d4, 0x3a09, 0x4a74, 0xa8, 0xd6, 0x71, 0xee, 0x08, 0x20, 0x2f, 0x7e	EFI_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest() returns EFI_INVALID_PARAMETER when OptionList contains client identity option.	Call InfoRequest() when OptionList contains client identity option, The return status should be EFI_INVALID_PARAMETER.
5.24.3.6.4	0x8647418d, 0xb3f9, 0x4bf5, 0xb5, 0x24, 0xf4, 0xc1, 0x7d, 0x36, 0x00, 0x20	EFI_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest() returns EFI_INVALID_PARAMETER with an NULL Retransmission.	Call InfoRequest() with an NULL Retransmission, The return status should be EFI_INVALID_PARAMETER.
5.24.3.6.5	0xf18e8693, 0xd00f, 0x497f, 0x86, 0xfe, 0xf9, 0x3a, 0x2f, 0x50, 0x38, 0x04	EFI_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest() returns EFI_INVALID_PARAMETER when both Retransmission.Mrd and Retransmission.Mrt are zero.	Call InfoRequest() when both Retransmission.Mrd and Retransmission.Mrt are zero, The return status should be EFI_INVALID_PARAMETER.
5.24.3.6.6	0x1669a032, 0x433a, 0x4dbc, 0x8c, 0x00, 0x81, 0xc4, 0xb6, 0x59, 0x78, 0x1f	EFI_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest() returns EFI_INVALID_PARAMETER when ReplyCallback is NULL.	Call InfoRequest() when ReplyCallback is NULL, The return status should be EFI_INVALID_PARAMETER.
5.24.3.6.7	0xaa884b5b, 0xb369, 0x46cc, 0x85, 0xa9, 0xfe, 0xb0, 0x33, 0xd1, 0xaa, 0x48	EFI_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest() returns EFI_NO_RESPONSE when Dhcp6 server doesn't response.	Call InfoRequest() when Dhcp6 server doesn't response, The return status should be EFI_NO_RESPONSE.
5.24.3.6.8	0x3ade8458, 0xd07a, 0x4f45, 0xbc, 0xc3, 0x49, 0x68, 0x20, 0xe9, 0x85, 0x0b	EFI_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest() returns EFI_ABORTED when the user returns error status from ReplyCallback function.	Call InfoRequest() when the user returns error status from ReplyCallback function, The return status should be EFI_ABORTED.

Number	GUID	Assertion	Test Description
5.24.3.6.9	0xc7cb2c53, 0xd008, 0x40b5, 0xb0, 0x53, 0xb2, 0x68, 0x08, 0xb8, 0x81, 0x3a	InfoRequestPacket should be received.	5.24.3.6.9 to 5.24.3.6.12 belong to one case. 1. Call `CreateChild()` to create an DHCP6 instance. 2. Create a timeout event. 3. Call `InfoRequest()` to obtain configuration information without ant IA address. 4. InfoRequestPacket should be received.
5.24.3.6.10	0x730310e5, 0x5df3, 0x41f9, 0xbf, 0x4a, 0x75, 0x1b, 0x01, 0xf9, 0x59, 0xef	The return status of `InfoRequest()` should be `EFI_SUCCESS`	5. Send the the Reply packet for the InfoRequest message. 6. The return status of `InfoRequest()` should be `EFI_SUCCESS`
5.24.3.6.11	0x1cb6efc5, 0x1d58, 0x4c8e, 0xb5, 0x7d, 0x83, 0x7d, 0xd2, 0x8c, 0xb0, 0xd3	The CallbackContext should be updated with `ReplyCallback()`	7. The CallbackContext should be updated with `ReplyCallback()`
5.24.3.6.12	0x5738bba8, 0xf1ad, 0x4889, 0x87, 0xed, 0x29, 0x21, 0x59, 0x17, 0x61, 0x48	The Timeout event should not be signaled.	8. The Timeout event should not be signaled.
5.24.3.6.13	0xa0995b80, 0x76ad, 0x4d99, 0xa5, 0xd3, 0x0d, 0x55, 0x1d, 0xb0, 0x94, 0x75	InfoRequestPacket should be received.	5.24.3.6.13 to 5.24.3.6.15 belong to one case. 1. Call `CreateChild()` to create an DHCP6 instance. 2. Call `InfoRequest()` to obtain configuration information without ant IA address. 3. InfoRequestPacket should be received.
5.24.3.6.14	0x46a40db0, 0x5b97, 0x4272, 0x98, 0x98, 0x9c, 0xbb, 0xe7, 0xa2, 0x22, 0x5f	The return status of `InfoRequest()` should be `EFI_SUCCESS`	4. Send the the Reply packet for the InfoRequest message. 5. The return status of `InfoRequest()` should be `EFI_SUCCESS`

Number	GUID	Assertion	Test Description
5.24.3.6.15	0x4b1612fa, 0x7561, 0x4b55, 0xb9, 0xa2, 0x76, 0x40, 0x02, 0xc6, 0x95, 0xe1	The CallbackContext should be updated with `ReplyCallback()`	6. The CallbackContext should be updated with `ReplyCallback()`

20.3.7 RenewRebind()

Number	GUID	Assertion	Test Description
5.24.3.7.1	0x613614f9, 0x2c96, 0x45ee, 0xad, 0xb8, 0xf0, 0x88, 0x72, 0xfd, 0x86, 0xf9	`EFI_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()` returns `EFI_ACCESS_DENIED` when the instance has not been configured.	Call `RenewRebind()` when the instance has not been configured, The return status should be `EFI_ACCESS_DENIED.`
5.24.3.7.2	0x28ce0a5d, 0x6f3d, 0x47ad, 0xb1, 0x95, 0xc2, 0x5f, 0xce, 0xd8, 0x98, 0xb5	`EFI_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()` returns `EFI_ACCESS_DENIED` when the instance is not in `Dhcp6Bound` state.	Call `RenewRebind()` when the instance is not in `Dhcp6Bound` state, The return status should be `EFI_ACCESS_DENIED.`
5.24.3.7.3	0x5c85dc0c, 0x634a, 0x4db3, 0x95, 0x81, 0x72, 0x0d, 0x1b, 0xda, 0x6c, 0x84	`EFI_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()` returns `EFI_ALREADY_STARTED` with `RebindRequest TRUE` when the instance in `Dhcp6Rebinding` state.	Call `RenewRebind()` with `RebindRequest TRUE` when the instance in `Dhcp6Rebinding` state, The return status should be `EFI_ALREADY_STARTED.`
5.24.3.7.4	0x94bc77a0, 0xb016, 0x4d71, 0x8f, 0x5b, 0xd0, 0x49, 0x1a, 0x2c, 0x4f, 0x0c	`EFI_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()` returns `EFI_ALREADY_STARTED` with `RebindRequest FALSE` when the instance in `Dhcp6Rebinding` state.	Call `RenewRebind()` with `RebindRequest FALSE` when the instance in `Dhcp6Rebinding` state, The return status should be `EFI_ALREADY_STARTED.`

5.24.3.7.5	0xcc0b1c38, 0x2b99, 0x4ef4, 0xb9, 0x35, 0x63, 0x2e, 0x12, 0x46, 0x4f, 0xf7	`EFI_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()` returns `EFI_ABORTED` when the user returns error status from callback function.	Call `RenewRebind()` when the user returns error status from callback function, The return status should be `EFI_ABORTED.`
5.24.3.7.6	0x2957725b, 0x7693, 0x40ac, 0xae, 0x81, 0x59, 0x54, 0x88, 0x25, 0xf7, 0x48	`EFI_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()` returns `EFI_SUCCESS` when the exchange process is executed successfully.	5.24.3.7.6 to 5.24.3.7.8 belong to one case. 1. Call `CreateChild()` to create an DHCP6 instance. 2. Call `Configure()` to initialize the DHCP6 instance. 3. Call `Start()` to start S.A.R.R process. 4. Get the return status of `Start()`, it should be `EFI_SUCCESS` 5. Call `GetModeData()` to get the `GetModeData` 6. `GetModeData.Ia.State` should be `Dhcp6Bound` 7. Call `RenewRebind()` and execute exchange process, including RENEW-REPLY, the return status should be `EFI_SUCCESS`
5.24.3.7.7	0xf495e992, 0xe807, 0x4a38, 0xbf, 0x42, 0x57, 0x1d, 0xd1, 0xfe, 0x8f, 0xc7	CallbackContext should updated.	8. CallbackContext should be updated.
5.24.3.7.8	0x23d22d31, 0x1852, 0x4527, 0x80, 0x73, 0xcf, 0x8a, 0x51, 0x16, 0xff, 0x92	The state is still `Dhcp6Bound`.	9. The state is still `Dhcp6Bound`

5.24.3.7.9	0x6ae394d7, 0xa5dc, 0x4147, 0x93, 0x5e, 0xf5, 0x07, 0xb2, 0xb8, 0xea, 0x35	`EFI_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()` returns `EFI_SUCCESS` when the exchange process is executed successfully.	5.24.3.7.9 to 5.24.3.7.10 belong to one case. 1. Call `CreateChild()` to create an DHCP6 instance. 2. Call `Configure()` to initialize the DHCP6 instance. 3. Call `Start()` to start S.A.R.R process. 4. Get the return status of `Start()`, it should be `EFI_SUCCESS` 5. Call `GetModeData()` to get the `GetModeData` 6. `GetModeData.Ia.State` should be `Dhcp6Bound` 7. Call `RenewRebind()` and execute exchange process, including RENEW-REBIND-REPLY, the return status should be `EFI_SUCCESS`
5.24.3.7.10	0x9f653dd2, 0x3edd, 0x47d6, 0xa6, 0x2e, 0x6c, 0x79, 0x99, 0x3d, 0xd9, 0x58	The state is still `Dhcp6Bound`.	8. The state is still `Dhcp6Bound`.

20.3.8 Decline()

Number	GUID	Assertion	Test Description
5.24.3.8.1	0x30c90eee, 0x69f1, 0x4a41, 0x88, 0x4d, 0x27, 0x6e, 0x9f, 0x6c, 0x0e, 0x33	`EFI_ DHCP6 PROTOCOL.Decline() - Decline()` returns `EFI_ACCESS_DENIED` when the instance has not been configured.	Call `Decline()` when the instance has not been configured, The return status should be `EFI_ACCESS_DENIED.`
5.24.3.8.2	0x2f3cd8a1, 0x8987, 0x434d, 0xa1, 0xbb, 0xfc, 0xb6, 0x83, 0x04, 0xf6, 0x0d	`EFI_ DHCP6 PROTOCOL.Decline() - Decline()` returns `EFI_ACCESS_DENIED` when the instance is not in `Dhcp6Bound` state.	Call `Decline()` when the instance is not in `Dhcp6Bound` state, The return status should be `EFI_ACCESS_DENIED.`

5.24.3.8.3	0x6224a781, 0xfa3a, 0x4190, 0xa4, 0xfa, 0x5b, 0xec, 0x33, 0xbf, 0x3f, 0xfc	**EFI_ DHCP6 PROTOCOL.Decline() - Decline()** returns **EFI_INVALID_PARAMETER** when the **AddressCount** is zero.	Call **Decline()** when the **AddressCount** is zero, The return status should be **EFI_INVALID_PARAMETER.**
5.24.3.8.4	0x1c8166c0, 0xbc5e, 0x4d1f, 0xa3, 0x8b, 0x65, 0x34, 0x7e, 0x76, 0x10, 0x69	**EFI_ DHCP6 PROTOCOL.Decline() - Decline()** returns **EFI_INVALID_PARAMETER** when the **Addresses** is **NULL**.	Call **Decline()** when the **Addresses** is **NULL**, The return status should be **EFI_INVALID_PARAMETER.**
5.24.3.8.5	0xc14f0d80, 0xe7e5, 0x4742, 0x9c, 0xc5, 0x27, 0xd0, 0x37, 0x79, 0x1b, 0x0d	**EFI_ DHCP6 PROTOCOL.Decline() - Decline()** returns **EFI_NOT_FOUND** when any specified address in **Addresses** is not correlated with the configured IA.	Call **Decline()** when any specified address in **Addresses** is not correlated with the configured IA, The return status should be **EFI_NOT_FOUND.**
5.24.3.8.6	0x44b4fcda, 0xf970, 0x4f3e, 0x88, 0xbb, 0x52, 0xf2, 0x52, 0xe9, 0x81, 0xdf	**EFI_ DHCP6 PROTOCOL.Decline() - Decline()** returns **EFI_ABORTED** when the user returns error status from callback function.	Call **Decline()** when the user returns error status from callback function, The return status should be **EFI_ABORTED.**
5.24.3.8.7	0x86606604, 0x5e2b, 0x4268, 0x91, 0xcd, 0x99, 0xc6, 0xb5, 0x7a, 0x42, 0xd8	**EFI_ DHCP6 PROTOCOL.Decline() - Decline()** returns **EFI_SUCCESS** with execute exchange process, including DECLINE- REPLY.	5.24.3.8.7 to 5.24.3.8.8 belong to one case. 1. Call **CreateChild()** to create an DHCP6 instance. 2. Call **Configure()** to initialize the DHCP6 instance. 3. Call **Start()** to start S.A.R.R process. 4. Get the return status of **Start()**, it should be **EFI_SUCCESS** 5. The CallbackContext is updated Call **GetModeData()** to get the **GetModeData** 6. **GetModeData.Ia.State** should be **Dhcp6Bound** 7. Call **Decline()** and execute exchange process, including DECLINE-REPLY, the return status should be **EFI_SUCCESS**

5.24.3.8.8	0x1119b246, 0x8627, 0x45a1, 0x87, 0x89, 0x5b, 0xba, 0x7b, 0x4c, 0x0b, 0x48	The state is still `Dhcp6Bound`.	8. The state is still `Dhcp6Bound`
5.24.3.8.9	0x554529cc, 0x30e2, 0x4269, 0x88, 0xb7, 0x72, 0x8e, 0x31, 0x1d, 0xbd, 0x1b	`EFI_ DHCP6 PROTOCOL.Decline() - Decline()` returns `EFI_SUCCESS` to decline all IP6 addresses of the configured IA and execute exchange process, including DECLINE-REPLY.	5.24.3.8.9 to 5.24.3.8.10 belong to one case. 1. Call `CreateChild()` to create an DHCP6 instance. 2. Call `Configure()` to initialize the DHCP6 instance. 3. Call `Start()` to start S.A.R.R process. 4. Get the return status of `Start()`, it should be `EFI_SUCCESS` 5. The CallbackContext is updated Call `GetModeData()` to get the `GetModeData` 6. `GetModeData.Ia.State` should be `Dhcp6Bound` 7. Call `Decline()` to decline all IP6 addresses of the configured IA and execute exchange process, including DECLINE- REPLY, the return status should be `EFI_SUCCESS`
5.24.3.8.10	0xf7449f19, 0x53e0, 0x4130, 0xba, 0x62, 0xea, 0x2b, 0x1f, 0x74, 0x8c, 0xa0	The state is still `Dhcp6Init`.	8. The state is still `Dhcp6Init`.

5.24.3.8.11	0xcdbd802e, 0x7647, 0x41bc, 0x9b, 0xe6, 0xe4, 0x11, 0x9f, 0x6c, 0x79, 0x2d	`EFI_ DHCP6 PROTOCOL.Decline()` - `Decline()` returns `EFI_SUCCESS` to decline all IP6 addresses of the configured IA and execute exchange process, including DECLINE-REPLY.	5.24.3.8.11 to 5.24.3.8.13 belong to one case. 1. Call `CreateChild()` to create an DHCP6 instance. 2. Create `IaInfoEvent` 3. Call `Configure()` to initialize the DHCP6 instance. 4. Call `Start()` to start S.A.R.R process. 5. Get the return status of `Start()`, it should be `EFI_SUCCESS` 6. The CallbackContext is updated Call `GetModeData()` to get the `GetModeData` 7. `GetModeData.Ia.State` should be `Dhcp6Bound` 8. Call `Decline()` to decline all IP6 addresses of the configured IA and execute exchange process, including DECLINE- REPLY, the return status should be `EFI_SUCCESS`
5.24.3.8.12	0xfce31eb4, 0xeb16, 0x4b22, 0xb3, 0x55, 0xa8, 0xb0, 0x82, 0x0f, 0x0d, 0x3d	After the Decline exchange process returns,the `IaInfoEvent` will be signaled.	9. After the Decline exchange process returns,the `IaInfoEvent` will be signaled.
5.24.3.8.13	0x313da4fc, 0xf2ce, 0x4ecc, 0xa9, 0x97, 0x03, 0xea, 0x77, 0xfb, 0xdb, 0x59	The state is still `Dhcp6Init`.	10. The state is still `Dhcp6Init`.

5.24.3.8.14	0x60c90ab2, 0x4372, 0x4b75, 0x84, 0x56, 0xe6, 0xe1, 0xfa, 0x34, 0x71, 0xad	`EFI_ DHCP6 PROTOCOL.Decline()` - `Decline()` returns `EFI_NO_RESPONSE` to decline all IP6 addresses of the configured IA without the response from server.	5.24.3.8.14 to 5.24.3.8.15 belong to one case. 1. Call `CreateChild()` to create an DHCP6 instance. 2. Call `Configure()` to initialize the DHCP6 instance. 3. Call `Start()` to start S.A.R.R process. 4. Get the return status of `Start()`, it should be `EFI_SUCCESS` 5. The CallbackContext is updated Call `GetModeData()` to get the `GetModeData` 6. `GetModeData.Ia.State` should be `Dhcp6Bound` 7. Call `Decline()` to decline all IP6 addresses of the configured IA without the response from server, the return status should be `EFI_NO_RESPONSE`
5.24.3.8.15	0x6af27ff2, 0xecb2, 0x4e96, 0xaf, 0xf7, 0xa7, 0x6b, 0x18, 0xe6, 0x38, 0xfa	The state is still `Dhcp6Init`.	8. The state is still `Dhcp6Init`.

20.3.9 Release()

Number	GUID	Assertion	Test Description
5.24.3.9.1	0xfd3f1c62, 0x37d9, 0x4f34, 0x85, 0xe5, 0x93, 0x85, 0x28, 0x2f, 0xd3, 0xc4	`EFI_ DHCP6 PROTOCOL.Release()` - `Release()` returns `EFI_ACCESS_DENIED` when the instance has not been configured.	Call `Release()` when the instance has not been configured, The return status should be `EFI_ACCESS_DENIED.`
5.24.3.9.2	0x38bc0e62, 0x4d8f, 0x4706, 0xb1, 0x39, 0xe0, 0xa7, 0x1c, 0xbd, 0x6d, 0x56	`EFI_ DHCP6 PROTOCOL.Release()` - `Release()` returns `EFI_ACCESS_DENIED` when the instance is not in `Dhcp6Bound` state.	Call `Release()` when the instance is not in `Dhcp6Bound` state, The return status should be `EFI_ACCESS_DENIED.`

5.24.3.9.3	0x8e214193, 0x3dfb, 0x48e3, 0xb6, 0xe3, 0xdb, 0x4b, 0xde, 0xa4, 0xbc, 0xef	**EFI_ DHCP6 PROTOCOL.Release() - Release()** returns **EFI_INVALID_PARAMETER** when the **AddressCount** is not zero and **Addresses** is **NULL**.	Call **Release()** when the **AddressCount** is not zero and **Addresses** is **NULL**, The return status should be **EFI_INVALID_PARAMETER.**
5.24.3.9.4	0x4b411cb3, 0x2427, 0x4315, 0xa3, 0x74, 0xa9, 0xdd, 0x29, 0xf7, 0x9a, 0xed	**EFI_ DHCP6 PROTOCOL.Release() - Release()** returns **EFI_NOT_FOUND** when any specified address in **Addresses** is not correlated with the configured IA.	Call **Release()** when any specified address in **Addresses** is not correlated with the configured IA, The return status should be **EFI_NOT_FOUND.**
5.24.3.9.5	0xa4b55b0e, 0x1037, 0x4717, 0x83, 0x53, 0x29, 0x24, 0xd3, 0x18, 0x23, 0x5d	**EFI_ DHCP6 PROTOCOL.Release() - Release()** returns **EFI_ABORTED** when the user returns error status from callback function.	Call **Release()** when the user returns error status from callback function, The return status should be **EFI_ABORTED.**
5.24.3.9.6	0x1459bb4e, 0xa926, 0x42cc, 0x99, 0x7d, 0xf8, 0x87, 0xf7, 0xd0, 0xbb, 0x71	**EFI_ DHCP6 PROTOCOL.Release() - Release()** returns **EFI_SUCCESS** to release one of the IPv6 address that has already been assigned to the configured IA.	5.24.3.9.6 to 5.24.3.9.7 belong to one case. 1. Call **CreateChild()** to create an DHCP6 instance. 2. Call **Configure()** to initialize the DHCP6 instance. 3. Call **Start()** to start S.A.R.R process. 4. Get the return status of **Start()**, it should be **EFI_SUCCESS** 5. The CallbackContext is updated Call **GetModeData()** to get the **GetModeData** 6. **GetModeData.Ia.State** should be **Dhcp6Bound** 7. Call **Release()** to release one of the IPv6 address that has already been assigned to the configured IA, the return status should be **EFI_SUCCESS**
5.24.3.9.7	0x7251daef, 0x57ae, 0x4fc6, 0x81, 0xf4, 0x10, 0xe2, 0x34, 0xa5, 0x87, 0xa4	The state is still **Dhcp6Bound**.	8. The state is still **Dhcp6Bound**

5.24.3.9.8	0x692e0cfb, 0x587d, 0x4906, 0x91, 0xa1, 0xcb, 0x20, 0x3b, 0x1e, 0xba, 0x2d	**EFI_ DHCP6 PROTOCOL.Decline() - Decline()** returns **EFI_SUCCESS** to release all IP6 addresses of the configured IA and execute exchange process, including RELEASE-REPLY.	5.24.3.9.8 to 5.24.3.9.9 belong to one case. 1. Call **CreateChild()** to create an DHCP6 instance. 2. Call **Configure()** to initialize the DHCP6 instance. 3. Call **Start()** to start S.A.R.R process. 4. Get the return status of **Start()**, it should be **EFI_SUCCESS** 5. The CallbackContext is updated Call **GetModeData()** to get the **GetModeData** 6. **GetModeData.Ia.State** should be **Dhcp6Bound** 7. Call **Release()** to release all IP6 addresses of the configured IA and execute exchange process, including RELEASE- REPLY, the return status should be **EFI_SUCCESS**
5.24.3.9.9	0x309de757, 0x2ab4, 0x4d5b, 0xb3, 0x7c, 0xb7, 0xdc, 0x46, 0x40, 0x4d, 0x1c	The state is still **Dhcp6Init**.	8. The state is still **Dhcp6Init**.
5.24.3.9.10	0x7b131129, 0x2fdb, 0x4a67, 0x8f, 0xaa, 0xe9, 0x0c, 0x1d, 0x08, 0xab, 0x94	**EFI_ DHCP6 PROTOCOL.Release() - Release()** returns **EFI_SUCCESS** to release all IP6 addresses of the configured IA and execute exchange process, including RELEASE-REPLY.	5.24.3.9.10 to 5.24.3.9.12 belong to one case. 1. Call **CreateChild()** to create an DHCP6 instance. 2. Create **IaInfoEvent** 3. Call **Configure()** to initialize the DHCP6 instance. 4. Call **Start()** to start S.A.R.R process. 5. Get the return status of **Start()**, it should be **EFI_SUCCESS** 6. The CallbackContext is updated Call **GetModeData()** to get the **GetModeData** 7. **GetModeData.Ia.State** should be **Dhcp6Bound** 8. Call **Release()** to release all IP6 addresses of the configured IA and execute exchange process, including RELEASE- REPLY, the return status should be **EFI_SUCCESS**

5.24.3.9.11	0x47d072fd, 0x5782, 0x413b, 0xb4, 0x62, 0xb3, 0x18, 0x58, 0x04, 0xad, 0x4e	After the Release exchange process returns,the `IaInfoEvent` will be signaled.	9. After the Release exchange process returns, the `IaInfoEvent` will be signaled.
5.24.3.9.12	0x22dc90e4, 0xd93c, 0x465d, 0x90, 0x27, 0x35, 0xe9, 0xab, 0x3f, 0x3a, 0x3a	The state is still `Dhcp6Init`.	10. The state is still `Dhcp6Init`.
5.24.3.9.13	0x52b03918, 0x1e8c, 0x4620, 0xa1, 0x44, 0x02, 0x09, 0xae, 0xf3, 0xc7, 0x9d	`EFI_ DHCP6 PROTOCOL.Release() - Release()` returns `EFI_NO_RESPONSE` to release all IP6 addresses of the configured IA without the response from server.	5.24.3.9.14 to 5.24.3.9.15 belong to one case. 1. Call `CreateChild()` to create an DHCP6 instance. 2. Call `Configure()` to initialize the DHCP6 instance. 3. Call `Start()` to start S.A.R.R process. 4. Get the return status of `Start()`, it should be `EFI_SUCCESS` 5. The CallbackContext is updated Call `GetModeData()` to get the `GetModeData` 6. `GetModeData.Ia.State` should be `Dhcp6Bound` 7. Call `Release()` to release all IP6 addresses of the configured IA without the response from server, the return status should be `EFI_NO_RESPONSE`
5.24.3.9.14	0xc65a96c1, 0x448c, 0x4d75, 0x81, 0x90, 0x19, 0x13, 0x76, 0x1e, 0x79, 0x3d	The state is still `Dhcp6Init`.	8. The state is still `Dhcp6Init`.

20.3.10 Stop()

Number	GUID	Assertion	Test Description

5.24.3.10.1	0x592d9e8d, 0x82cd, 0x44d8, 0xbf, 0x26, 0x0b, 0x40, 0x81, 0x25, 0x65, 0x17	**EFI_ DHCP6 PROTOCOL.Stop() - Stop()** returns **EFI_SUCCESS** when the instance has not been configured.	Call **Stop()** when the instance has not been configured, The return status should be **EFI_SUCCESS.**
5.24.3.10.2	0x69ac94c1, 0xb57f, 0x4251, 0xb9, 0x56, 0x20, 0xaa, 0x9f, 0x30, 0x0d, 0xc1	**EFI_ DHCP6 PROTOCOL.Stop() - Stop()** returns **EFI_SUCCESS** when the instance has been configured.	Call **Stop()** when the instance has been configured, The return status should be **EFI_SUCCESS.**
5.24.3.10.3	0x51255767, 0x7218, 0x400d, 0xa2, 0xd7, 0x3f, 0x3e, 0x50, 0x8c, 0x90, 0x64	**EFI_ DHCP6 PROTOCOL.Release() - Release()** returns **EFI_INVALID_PARAMETER** when the **AddressCount** is not zero and **Addresses** is **NULL**.	5.24.3.9.3 to 5.24.3.9.5 belong to one case. 1. Call **CreateChild()** to create an DHCP6 instance. 2. Create **IaInfoEvent** 3. Call **Configure()** to initialize the DHCP6 instance. 4. Call **Start()** to start S.A.R.R process. 5. Get the return status of **Start()**, it should be **EFI_SUCCESS** 6. The CallbackContext is updated Call **GetModeData()** to get the **GetModeData** 7. **GetModeData.Ia.State** should be **Dhcp6Bound** 8. **IaInfoEvent** should be signaled. 9. Call **Stop()** to stop all IP6 addresses of the configured IA and execute exchange process, including RELEASE-REPLY, the return status should be **EFI_SUCCESS**
5.24.3.10.4	0xd00b1578, 0x5f23, 0x4ab7, 0x99, 0x40, 0x98, 0x51, 0x8a, 0x30, 0x8c, 0x08	**IaInfoEvent** should be signaled.	**IaInfoEvent** should be signaled.
5.24.3.10.5	0xcfa8dc36, 0xc246, 0x45d7, 0x94, 0xf1, 0xc9, 0x18, 0x54, 0xd6, 0x38, 0xad	The state of IA should be **Dhcp6Init**.	The state of IA should be **Dhcp6Init**

20.3.11 Parse()

Number	GUID	Assertion	Test Description
5.24.3.11.1	0x15a7d1de, 0x4bf6, 0x4507, 0xa3, 0xe2, 0xa1, 0xa4, 0x2e, 0xdd, 0x43, 0x23	EFI_ DHCP6 PROTOCOL.Parse() - Parse() returns EFI_INVALID_PARAMETER when the Packet is NULL.	Call Parse() when the Packet is NULL, The return status should be EFI_INVALID_PARAMETER.
5.24.3.11.2	0x28a7d965, 0x82bf, 0x49c6, 0xb1, 0xd8, 0x56, 0x08, 0x37, 0x0b, 0xdd, 0x62	EFI_ DHCP6 PROTOCOL.Parse() - Parse() returns EFI_INVALID_PARAMETER when the Packet is not well-formed(length is too small).	Call Parse() when the Packet is not well-formed(length is too small), The return status should be EFI_INVALID_PARAMETER.
5.24.3.11.3	0x2228cc36, 0xa56b, 0x4aa8, 0xa2, 0x15, 0x06, 0x01, 0xce, 0xfe, 0x00, 0x94	EFI_ DHCP6 PROTOCOL.Parse() - Parse() returns EFI_INVALID_PARAMETER when the OptionCount is not zero and PacketOptionList is NULL.	Call Parse() when the OptionCount is not zero and PacketOptionList is NULL, The return status should be EFI_INVALID_PARAMETER.
5.24.3.11.4	0x444b0ef0, 0x0297, 0x4805, 0x8b, 0x2a, 0xc4, 0xa2, 0xf8, 0x82, 0xac, 0x2c	EFI_ DHCP6 PROTOCOL.Parse() - Parse() returns EFI_INVALID_PARAMETER when the OptionCount is NULL.	Call Parse() when the OptionCount is NULL, The return status should be EFI_INVALID_PARAMETER.
5.24.3.11.5	0x49182e78, 0x34dc, 0x4450, 0xb6, 0x2c, 0xfe, 0x28, 0x33, 0x51, 0xc1, 0x96	EFI_ DHCP6 PROTOCOL.Parse() - Parse() returns EFI_BUFFER_TOO_SMALL when the OptionCount is NULL.	Call Parse() when OptionCount is smaller than the number of option that were found in the Packet, The return status should be EFI_BUFFER_TOO_SMALL.
5.24.3.11.6	0x43dcf866, 0x9f05, 0x47d5, 0x92, 0xa1, 0x1e, 0x6f, 0x26, 0xf4, 0x1f, 0x61	OptionCount should be update to the right number of option that is found in the packet.	OptionCount should be update to the right number of option that is found in the packet.
5.24.3.11.7	0xacfb1bb7, 0x7b28, 0x4c35, 0xbd, 0x9f, 0x7e, 0x89, 0xa1, 0x9e, 0x54, 0xe2	EFI_ DHCP6 PROTOCOL.Parse() - Parse() returns EFI_SUCCESS with the valid parameters.	Call Parse() with the valid parameters, The return status should be EFI_SUCCESS.

5.24.3.11.8	0xbb477381, 0x7731, 0x4259, 0x87, 0x01, 0xca, 0x1f, 0x71, 0xd6, 0xf9, 0x7e	The `OpCode` should be retrieved correctly.	The `OpCode` should be retrieved correctly.

21 Network Protocols TCP, IP and Configuration

21.1 EFI_TCP4_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_TCP4_PROTOCOL Section.

21.1.1 GetModeData()

Number	GUID	Assertion	Test Description
5.25.1.1.1	0xf7c924b2, 0xaaa6, 0x4729, 0xb1, 0xd0, 0x71, 0xf8, 0xed, 0xc8, 0x81, 0x8f	`EFI_TCP4_PROTOCOL.GetModeData()` – invokes `GetModeData()` with a *`Tcp4State`* value of `NULL`.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new Tcp4 instance. 3. Call `EFI_TCP4_PROTOCOL.GetModeData()` with a *`Tcp4State`* value of `NULL`. The return status should be `EFI_SUCCESS`. 4. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.2	0xd39219b6, 0xa262, 0x4797, 0xac, 0x44, 0x35, 0xe5, 0x46, 0xc0, 0xe9, 0xc8	`EFI_TCP4_PROTOCOL.GetModeData()` – invokes `GetModeData()` with a *`Tcp4ConfigData`* value of `NULL`.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new Tcp4 instance. 3. Call `EFI_TCP4_PROTOCOL.GetModeData()` with a *`Tcp4ConfigData`* value of `NULL`. The return status should be `EFI_SUCCESS`. 4. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.3	0x7be1ddb5, 0xf3bf, 0x4eb3, 0x87, 0x52, 0x9a, 0xf6, 0x91, 0x6c, 0x51, 0xc5	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with a *Ip4ModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with a *Ip4ModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.4	0x6255190b, 0x3eb5, 0x40e9, 0xbd, 0x24, 0x26, 0x85, 0xfc, 0x87, 0xab, 0x29	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with a *MnpConfigData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with a *MnpConfigData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.5	0x62f96356, 0x53d3, 0x4fdd, 0xb1, 0x36, 0x12, 0x53, 0xc2, 0xb0, 0x14, 0x8e	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with a *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3.Call **EFI_TCP4_PROTOCOL.GetModeData()** with a *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.6	0xf753264f, 0x22d0, 0x4e19, 0x81, 0x81, 0xf3, 0x4d, 0xd9, 0xf6, 0xdb, 0x59	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with both the *Tcp4State* and *Tcp4ConfigData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3.Call **EFI_TCP4_PROTOCOL.GetModeData()** with both the *Tcp4State* and *Tcp4ConfigData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.7	0x0848d02d, 0x3463, 0x4f06, 0xb1, 0x6e, 0xce, 0xd1, 0x32, 0x3b, 0x53, 0xd2	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with both the *Tcp4State* and *Ip4ModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3.Call **EFI_TCP4_PROTOCOL.GetModeData()** with both the *Tcp4State* and *Ip4ModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.8	0xa92b1577, 0x6d14, 0x4d77, 0x9f, 0x5b, 0x85, 0xba, 0x55, 0xf8, 0x1d, 0x52	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with both the *Tcp4State* and *MnpConfigData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3.Call **EFI_TCP4_PROTOCOL.GetModeData()** with both the *Tcp4State* and *MnpConfigData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.9	0x31388819, 0x2579, 0x414e, 0x89, 0x0f, 0xfe, 0xc9, 0xbe, 0x08, 0x8c, 0x37	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with both the *Tcp4State* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with both the *Tcp4State* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.10	0xec2502c3, 0xdf73, 0x4bff, 0xa4, 0xac, 0xaf, 0x5e, 0x77, 0x3d, 0xbf, 0xa1	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with both the *Tcp4ConfigData* and *Ip4ModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with both the *Tcp4ConfigData* and *Ip4ModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.11	0x32100ad2, 0xbc14, 0x426b, 0x86, 0xee, 0x0e, 0xc1, 0x8e, 0xb3, 0x11, 0xb2	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with both the *Tcp4ConfigData* and *MnpConfigData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with both the *Tcp4ConfigData* and *MnpConfigData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.12	0x3ae2f864, 0x8963, 0x48ca, 0xbc, 0xa5, 0x01, 0x0d, 0xdf, 0x13, 0x9e, 0xb1	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with both the *Tcp4ConfigData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with both the *Tcp4ConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.13	0xc72c71bf, 0x781f, 0x4a08, 0xac, 0xa1, 0xb0, 0x1f, 0xbc, 0x79, 0x91, 0x60	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with both the *Ip4ModeData* and *MnpConfigData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with both the *Ip4ModeData* and *MnpConfigData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.14	0x86fb248c, 0x3238, 0x411e, 0xa6, 0xa5, 0x41, 0x1c, 0x21, 0x42, 0x82, 0xc4	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with both the *Ip4ModeData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with both the *Ip4ModeData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.15	0xdddaf809, 0xa972, 0x4376, 0xb2, 0xdb, 0x1a, 0x35, 0x14, 0xcc, 0x88, 0x0a	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with both the *MnpConfigData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with both the *MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.16	0xf6873b19, 0xbdef, 0x4bac, 0x93, 0x4d, 0x55, 0xe0, 0x87, 0x06, 0x67, 0x2e	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with the *Tcp4State, Tcp4ConfigData* and *Ip4ModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4State, Tcp4ConfigData* and *Ip4ModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.17	0x8b5d7aa1, 0x9838, 0x4b5a, 0x88, 0x37, 0xa7, 0xd1, 0x93, 0x5f, 0x8e, 0x46	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with the *Tcp4State, Tcp4ConfigData* and *MnpConfigData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4State, Tcp4ConfigData* and *MnpConfigData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.18	0x064d8786,0x876c, 0x46a2, 0x84, 0xa7, 0x1a, 0x69, 0x8a, 0x59, 0x65, 0xb0	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with the *Tcp4State*, *Tcp4ConfigData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4State*, *Tcp4ConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.19	0xb98bb8a0, 0xf8bd, 0x405d, 0x99, 0x6c, 0x52, 0x47, 0x3c, 0x20, 0x43, 0x38	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with the *Tcp4State*, *Ip4ModeData* and *MnpConfigData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()GetModeData()** with the *Tcp4State*, *Ip4ModeData* and *MnpConfigData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.20	0x23fa07b0, 0xcd96, 0x490b, 0xa6, 0xf6, 0xe6, 0x5d, 0x8d, 0x89, 0x28, 0xc6	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with the *Tcp4State*, *Ip4ModeData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4State*, *Ip4ModeData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.21	0xbfa282e9, 0x6393, 0x428f, 0x8f, 0xe1, 0x6d, 0xf2, 0xca, 0xfc, 0x9b, 0x84	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with the *Tcp4State*, *MnpConfigData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4State*, *MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.22	0x245ea469, 0x0422, 0x45fa, 0x97, 0x4b, 0x0b, 0x45, 0xc2, 0xf8, 0x70, 0x27	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with the *Tcp4ConfigData*, *Ip4ModeData* and *MnpConfigData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4ConfigData*, *Ip4ModeData* and *MnpConfigData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.23	0x70445b77, 0x59ec, 0x4fd1, 0xba, 0x2b, 0x9a, 0xcd, 0x7e, 0x0f, 0x78, 0x83	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with the *Tcp4ConfigData*, *Ip4ModeData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4ConfigData*, *Ip4ModeData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.24	0xfa72381d, 0x5c30, 0x4dd1, 0xba, 0xf4, 0xff, 0xca, 0x30, 0x0a, 0x2f, 0x15	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with the *Tcp4ConfigData, MnpConfigData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4ConfigData, MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.25	0xad6d2b6f, 0x8e2f, 0x49ed, 0xa1, 0xd8, 0x3b, 0x33, 0x69, 0x04, 0x2c, 0x2e	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with the *Ip4ModeData, MnpConfigData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Ip4ModeData, MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.26	0x7d6ef330, 0x3522, 0x434d, 0x9f, 0xf7, 0x34, 0x84, 0xe4, 0x0d, 0x1f, 0xc5	**EFI_TCP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** with the *Tcp4ConfigData*,*Ip4ModeData, MnpConfigData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4ConfigData*,*Ip4ModeData, MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.27	0x1f83096c, 0x6342, 0x4f1a, 0xa1, 0x22, 0xe3, 0x1e, 0xd5, 0x63, 0x36, 0x53	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with the *Tcp4State*,*Ip4ModeData*,*MnpConfigData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4State*,*Ip4ModeData*, *MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.28	0xe7f67d55, 0x5bb8, 0x400c, 0x99, 0xfc, 0x53, 0x0e, 0x5d, 0xc0, 0x1f, 0x51	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with the *Tcp4State*, *Tcp4ConfigData*, *MnpConfigData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4State*, *Tcp4ConfigData*, *MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.29	0xa72e1aec, 0x5502, 0x434c, 0xb8, 0xed, 0x68, 0x0b, 0x54, 0xb2, 0xa8, 0x8e	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with the *Tcp4State*, *Tcp4ConfigData*, *Ip4ModeData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4State*, *Tcp4ConfigData*, *Ip4ModeData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.30	0x59e6caf6, 0x0db0, 0x45f9, 0x91, 0x50, 0xca, 0xdb, 0x1c, 0xae, 0x9b, 0xc2	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with the *Tcp4State*, *Tcp4ConfigData*, *Ip4ModeData* and *MnpConfigData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with the *Tcp4State*, *Tcp4ConfigData*, *Ip4ModeData* and *MnpConfigData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.31	0x3fd1ebb6, 0x3edd, 0x4a61, 0x98, 0x8e, 0xfc, 0x92, 0xbd, 0xef, 0x8d, 0xf0	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with all the optional parameters *Tcp4State*, *Tcp4ConfigData*, *Ip4ModeData*, *MnpConfigData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with all the optional parameters *Tcp4State*, *Tcp4ConfigData*, *Ip4ModeData*, *MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.32	0x53417686, 0xcf3b, 0x4dc5, 0x9d, 0x7b, 0x83, 0xad, 0x7c, 0x96, 0x3e, 0x0f	**EFI_TCP4_PROTOCOL .GetModeData()** – invokes **GetModeData()** with none of the optional parameters *Tcp4State*, *Tcp4ConfigData*, *Ip4ModeData*, *MnpConfigData* and *SnpModeData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with none of the optional parameters *Tcp4State*, *Tcp4ConfigData*, *Ip4ModeData*, *MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.33	0x05f9a5f1, 0x445d, 0x46d2, 0xb8, 0x82, 0xf0, 0xe2, 0x34, 0x72, 0xca, 0x48	**EFI_TCP4_PROTOCOL.GetModeData() –** invokes **GetModeData()** to correctly get the *Tcp4ConfigData*.TypeOfService.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection,then receive the packet. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with none of the optional parameters *Tcp4State*, *Tcp4ConfigData*, *Ip4ModeData*, *MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.34	0x529c2a7a, 0xf533, 0x4777, 0xa3, 0x7d, 0x09, 0x6f, 0x0c, 0x52, 0x99, 0xa7	**EFI_TCP4_PROTOCOL.GetModeData() –** invokes **GetModeData()** to correctly get the *Tcp4ConfigData* and TimeToLive.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection,then receive the packet. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with none of the optional parameters *Tcp4State*, *Tcp4ConfigData*, *Ip4ModeData*, *MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.1.35	0xe6bc773d, 0xf461, 0x4f0f, 0x97, 0xed, 0x78, 0x69, 0x7f, 0x0b, 0x81, 0xcb	**EFI_TCP4_PROTOCOL .GetModeData() –** invokes **GetModeData()** to correctly get the *Tcp4ConfigData* and AccessPoint.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection,then receive the packet. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with none of the optional parameters *Tcp4State*, *Tcp4ConfigData, Ip4ModeData, MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.1.36	0x42f51ebd, 0x24d2, 0x42af, 0xb9, 0xad, 0x7e, 0xb2, 0xfe, 0x2a, 0x18, 0x65	**EFI_TCP4_PROTOCOL .GetModeData() –** invokes **GetModeData()** to correctly get the *Tcp4ConfigData* and AccessPoint.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new Tcp4 instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection,then receive the packet. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** with none of the optional parameters *Tcp4State*, *Tcp4ConfigData, Ip4ModeData, MnpConfigData* and *SnpModeData* value of **NULL**. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.2 Configure()

Number	GUID	Assertion	Test Description
5.25.1.2.1	0x64729d75, 0x1007, 0x4b20, 0x9b, 0x78, 0x59, 0xc4, 0xc7, 0x02, 0xec, 0x9e	**EFI_TCP4_PROTOCOL.Configure()** – invokes **Configure()** when using a default address, and configuration has not finished yet.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** when using a default address, and configuration (through DHCP, BOOTP, RARP, etc.) has not finished yet. The return status should be **EFI_NO_MAPPING**. 3. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.2.2	0xe8cef00f, 0x0796, 0x4b1c, 0xbd, 0x09, 0x2c, 0x86, 0xdb, 0x4d, 0xba, 0x44	**EFI_TCP4_PROTOCOL.Configure()** – invokes **Configure()** with a *TcpConfigData->AccessPoint.StationAddress* value of an invalid unicast IPv4 address when *TcpConfigData->AccessPoint.UseDefaultAddress* is **FALSE**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **Tcp.Configure()** with a *TcpConfigData->AccessPoint.StationAddress* value of an invalid unicast IPv4 address when *TcpConfigData->AccessPoint.UseDefaultAddress* is **FALSE**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.2.3	0x6aaabbca, 0xb7d3, 0x49a1, 0x8f, 0x11, 0x4a, 0x82, 0x3f, 0x2e, 0xd9, 0x00	**EFI_TCP4_PROTOCOL.Configure()** – invokes **Configure()** with a *TcpConfigData->AccessPoint.SubnetMask* value of an invalid IPv4 address mask when *TcpConfigData->AccessPoint.UseDefaultAddress* is **FALSE**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **Tcp.Configure()** with a *TcpConfigData->AccessPoint.SubnetMask* value of an invalid IPv4 address mask when *TcpConfigData->AccessPoint.UseDefaultAddress* is **FALSE**. The subnet mask must be contiguous. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.2.4	0xa176de8a, 0xd68d, 0x4529, 0x97, 0xb5, 0xcf, 0x13, 0xa7, 0xe3, 0x33, 0xc0	**EFI_TCP4_PROTOCOL.Configure()** – invokes **Configure()** with a *TcpConfigData->AccessPoint. RemoteAddress* value of an invalid unicast IPv4 address.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** with a *TcpConfigData->AccessPoint. RemoteAddress* value of an invalid unicast IPv4 address. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.2.5	0xf3f1b054, 0xd497 ,0x4e1a, 0xa4, 0x67, 0x9c, 0x23, 0xab, 0xbb, 0x43, 0x08	**EFI_TCP4_PROTOCOL.Configure()** – invokes **Configure()** when a same access point has been configured in other TCP instance previously.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** when a same access point has been configured in other TCP instance previously. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.2.6	0x6fd9c85c, 0x7cc5, 0x480f, 0xa9, 0x14, 0x8f, 0xbd, 0x0d, 0x30, 0xba, 0x15	**EFI_TCP4_PROTOCOL. Configure()** – invokes **Configure()** with a *TcpConfigData->AccessPoint.RemoteAddress* value of 0 when *TcpConfigData->AccessPoint.ActiveFlag* is **TRUE**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **Tcp.Configure()** with a *TcpConfigData->AccessPoint.RemoteAddress* value of 0 when *TcpConfigData->AccessPoint.ActiveFlag* is **TRUE**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.2.7	0x0782f91f, 0x5553, 0x4854, 0x92, 0xbe, 0xb5, 0x25, 0x79, 0x0b, 0x42, 0x79	**EFI_TCP4_PROTOCOL.Configure()** – invokes **Configure()** with a *TcpConfigData->AccessPoint.RemotePort* value of 0 when *TcpConfigData->AccessPoint.ActiveFlag* is **TRUE**..	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** with *TcpConfigData->AccessPoint.RemotePort* is 0 when *TcpConfigData->AccessPoint.ActiveFlag* is **TRUE**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.2.8	0x21e9706f, 0xf449, 0x4c3c, 0x95, 0x6e, 0xf4, 0x28, 0xdd, 0x22, 0x5a, 0xb9	EFI_TCP4_PROTOCOL.Configure() – invokes Configure() with the TCP instance configured without calling Configure() with NULL to reset it.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Configure() to configure the new Tcp4 instance. 3. Call EFI_TCP4_PROTOCOL.Configure() to configure the Tcp4 instance again without calling Configure() with NULL to reset it. The return status should be EFI_ACCESS_DENIED. 4. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.
5.25.1.2.9	0xa1e6077c, 0x035e, 0x4684, 0x81, 0xe2, 0x99, 0xb2, 0x44, 0x4e, 0x0b, 0x9d	EFI_TCP4_PROTOCOL.Configure() – invokes Configure() when one or more of the control options are not supported in the implementation.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Configure() when one or more of the control options are not supported in the implementation. The return status should be EFI_UNSUPPORTED. 3. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.

21.1.3 Connect()

Number	GUID	Assertion	Test Description
5.25.1.4.1	0x0dc45007, 0xff6e, 0x41da, 0x81, 0x05, 0x55, 0x2d, 0x88, 0xe8, 0x09, 0x14	EFI_TCP4_PROTOCOL.Connect() – invokes Connect() when the instance has not been configured.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Connect() when the instance has not been configured. The return status should be EFI_NOT_STARTED. 3. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.
5.25.1.4.2	0xa00efef2, 0xd596, 0x4332, 0xa1, 0x9b, 0x38, 0x0a, 0xe0, 0xd7, 0x23, 0xe0	EFI_TCP4_PROTOCOL.Connect() – invokes Connect() when the instance is not configured as an active one.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Configure() to configure the new instance as not an active one. 3. Call EFI_TCP4_PROTOCOL.Connect() to open a connection when the instance is not configured as an active one. The return status should be EFI_ACCESS_DENIED. 4. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.4.3	0xe204e699, 0x7941, 0x4d65, 0x8b, 0x2e, 0xf2, 0xbe, 0xd3, 0x6c, 0xcf, 0x7e	**EFI_TCP4_PROTOCOL.Connect()** – invokes **Connect()** when the instance is not in *Tcp4State*Closed state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** configure the instance again when it is not in *Tcp4State***Closed** state. The return status should be **EFI_ACCESS_DENIED**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.4.4	0x3011f8f5, 0x6ccf, 0x46f4, 0xb9, 0x9a, 0x09, 0xd0, 0xf3, 0xde, 0x3a, 0x12	**EFI_TCP4_PROTOCOL.Connect() –** invokes **Connect()** with a *ConnectionToken* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** with a *ConnectionToken* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.4.5	0x513b33c4, 0x4df0, 0x449e, 0xb8, 0xf5, 0xd6, 0x4e, 0x30, 0x27, 0x0e, 0xa4	**EFI_TCP4_PROTOCOL.Connect()** – invokes **Connect()** with a *ConnectionToken->CompletionToken.Event* value of **NULL.**	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** with a *ConnectionToken->CompletionToken.Event* value of **NULL.** The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.4.6	0x672d8332, 0xa9a0, 0x4111, 0xa2,0x95, 0x10,0xfe, 0x88,0x17, 0x86,0x04	**EFI_TCP4_PROTOCOL.Connect()** – **Connect()** must return **EFI_CONNECTION_REFUSED** when the instance is in *SYN-RCVD* state & receive a *RST*	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** Receive *SYN* & Send a *SYN* to put TCP state machine in *SYN-RCVD* state. 4. Send a **RST** & check Connection Token state to be changed to **EFI_CONNECTION_REFUSED** 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.4 Accept()

Number	GUID	Assertion	Test Description
5.25.1.5.1	0x81d93128, 0xfcda, 0x49fa, 0x87, 0xea, 0xd4, 0x8e, 0x83, 0x1a, 0x6e, 0x8b	`EFI_TCP4_PROTOCOL.Accept()` – invokes `Accept()` when the instance has not been configured.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Accept()` when the instance has not been configured. The return status should be `EFI_NOT_STARTED`. 3. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.5.2	0x9f46e8f3, 0xc4e0, 0x4027, 0x88, 0x09, 0x6b, 0xc4, 0xc6, 0x5d, 0xca, 0xf5	`EFI_TCP4_PROTOCOL.Accept()` – invokes `Accept()` when the instance is not a passive one.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Accept()` when the instance is not a passive one. The return status should be `EFI_ACCESS_DENIED`. 4. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.5.3	0xd59b4f29, 0x874c, 0x4282, 0xac, 0x7d, 0x3f, 0xf6, 0x8d, 0x52, 0x54, 0xe8	`EFI_TCP4_PROTOCOL.Accept()` – invokes `Accept()` when the instance is not in *Tcp4State*Listen state.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance. 3. Call `EFI_TCP4_PROTOCOL.Accept()` to initiate an asynchronous accept request to wait for an incoming connection. 4.Call `EFI_TCP4_PROTOCOL.GetModeData()` to change the instance state to *Tcp4State*Established. 5. Call `EFI_TCP4_PROTOCOL.Accept()` when the instance is not in *Tcp4State*`Listen` state. The return status should be `EFI_ACCESS_DENIED`. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.5.4	0x85f6ab8a, 0x9374, 0x4afe, 0x85, 0x76, 0x5e, 0xa4, 0x44, 0x57, 0x87, 0x31	`EFI_TCP4_PROTOCOL.Accept() –` invokes `Accept()` when the same listen token has already existed in the listen token queue of this TCP instance.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance. 3. Call `EFI_TCP4_PROTOCOL.Accept()` to initiate an asynchronous accept request to wait for an incoming connection. 4. Call `EFI_TCP4_PROTOCOL.Accept()` again when the same listen token has already existed in the listening token queue of this TCP instance. The return status should be `EFI_ACCESS_DENIED`. 5. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.5.5	0x26f62b3c, 0xb67a, 0x4f2a, 0x86, 0x8f, 0x65, 0x30, 0xf6, 0x5e, 0xe3, 0x1b	**EFI_TCP4_PROTOCOL.Accept()** – invokes **Accept()** with a *ListenToken* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Accept()** with a *ListenTokenListenToken* value of **NULL**. The return status should be **EFI_INVALID_PARAMETEREFI_INVALID_PARAMETER**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.5.6	0x4fbd5006, 0x0d81, 0x40d0, 0xb8, 0xff, 0xca, 0x77, 0x03, 0x80, 0x34, 0xb6	**EFI_TCP4_PROTOCOL.Accept()** – invokes **Accept()** with a *ListentToken->CompletionToken.Event* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Accept()** with a *ListentToken->CompletionToken.Event* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.5.7	0x0df289ca, 0xfc53, 0x4fc2, 0x92, 0xb3, 0xb4, 0x3a, 0xcf, 0x3c, 0x50, 0x34	**EFI_TCP4_PROTOCOL.Accept() –** invokes **Accept()** to listen on the passive instance to accept an incoming connection request.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Accept()** to listen on the passive instance to accept an incoming connection request. The return status should be **EFI_SUCCESS**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.5.8	0x71f6d2e2, 0x9d2a, 0x435e, 0x83,0x0e, 0x63,0x9f, 0x1f,0xe7, 0x31,0x95	**EFI_TCP4_PROTOCOL.Accept() –** Call **Accept()** to listen on the passive instance to accept an incoming connection request. If received a *RST*, parent TCP State should Still be *LISTEN*.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Accept()** to listen on the passive instance to accept an incoming connection request. 4.Send a *RST* to Host and Call **GetModeData()** to get Parent state. The state should be *LISTEN* 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.5.9	0x0b1d8b5c, 0xc111, 0x4548, 0xac,0x9e, 0x3c,0xc2, 0x85,0xaa, 0x0d,0xab	**EFI_TCP4_PROTOCOL.Accept()** – Call **Accept()** to listen on the passive instance to accept an incoming connection request. Must return **EFI_SUCCESS** after a successful passive mode connection	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Accept()** to listen on the passive instance to accept an incoming connection request. 4.Connect & check return status should be **EFI_SUCCESS.** 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.5.10	0xbef6d443, 0xbece, 0x4315, 0x84,0x57, 0x90,0xe4, 0xb1,0xc4, 0x34,0x0a	**EFI_TCP4_PROTOCOL.Accept()** – Call **Accept()** to listen on the passive instance to accept an incoming connection request. New created connection state should be *ESTABLISED* after a successful passive mode connection	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Accept()** to listen on the passive instance to accept an incoming connection request. 4.Connect & Call **GetModeData()** check new created connection status should be *ESTABLISED*. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.5 Transmit()

Number	GUID	Assertion	Test Description
5.25.1.6.1	0xe268c41a, 0x3749, 0x4e6c, 0x95, 0xdc, 0x11, 0x6c, 0x4a, 0x57, 0x93, 0x40	`EFI_TCP4_PROTOCOL.Transmit()` – invokes `Transmit()` when the instance has not been configured.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Transmit()` when the instance has not been configured. The return status should be `EFI_NOT_STARTED`. 3. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.6.2	0xf05cb723, 0x7194, 0x45f9, 0xae, 0x3d, 0x52, 0x9b, 0xb3, 0x63, 0xde, 0x19	`EFI_TCP4_PROTOCOL.Transmit()` – invokes `Transmit()` to transmit a packet with a *Token* value of `NULL`.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to open an active connection. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet with a *Token* value of `NULL`. The return status should be `EFI_INVALID_PARAMETER`. 5. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.6.3	0xaaba9e1f, 0xdc0c, 0x4320, 0x8a, 0x01, 0x51, 0xc0, 0x07, 0x22, 0xfb, 0x73	**EFI_TCP4_PROTOCOL.Transmit()** – invokes **Transmit()** to transmit a packet with a *Token->CompletionToken.Event* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet with a *Token->CompletionToken.Event* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.6.4	0x96eb6c53, 0x68bc, 0x4a3b, 0xa4, 0x07, 0x96, 0xbc, 0x97, 0xac, 0x8e, 0x1e	**EFI_TCP4_PROTOCOL.Transmit()** – invokes **Transmit()Transmit()** to transmit a packet with a *Token->Packet.TxData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet with a *Token->Packet.TxData* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.6.5	0xc0bce6b7, 0xcd60, 0x484a, 0xb3, 0x37, 0xf5, 0xb4, 0xfe, 0x99, 0x30, 0xb2	**EFI_TCP4_PROTOCOL.Transmit()** – invokes **Transmit()** to transmit a packet with a *Token->Packet.FragmentCount* value of 0**.**	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet with a *Token->Packet.FragmentCount* value of 0**.** The return status should be **EFI_INVALID_PARAMETER**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.6.6	0xc00b7871, 0xa4ac, 0x4bfd, 0x81, 0xda, 0x78, 0x52, 0xc0, 0xc0, 0x54, 0x65	**EFI_TCP4_PROTOCOL.Transmit()** – invokes **Transmit()** to transmit a packet with a *Token->Packet.DataLength* value other than equal to the sum of fragment lengths.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet with a *Token->Packet.DataLength* value other than equal to the sum of fragment lengths. The return status should be **EFI_INVALID_PARAMETER.** 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.6.7	0x7e824bb2, 0xb6cd, 0x49b6, 0x9f, 0x1b, 0xe3, 0x60, 0x02, 0x7d, 0xd7, 0x5f	**EFI_TCP4_PROTOCOL.Transmit()** – invokes **Transmit()** when a transmit completion token with the same *Token->CompletionToken.Event* which was already in the transmission queue.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** when a transmit completion token with the same *Token->CompletionToken.Event* in step 4 which was already in the transmission queue. The return status should be **EFI_ACCESS_DENIED**. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.6.8	0x97d1f634, 0x39aa, 0x44a3, 0xb4, 0xc8, 0x22, 0xa4, 0x17, 0x2b, 0x9a, 0x12	**EFI_TCP4_PROTOCOL.Transmit()** – invokes **Transmit()** when the current instance is in *Tcp4State*Closed state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. OS send RST to let EUT enter *Tcp4State*Closed state. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** when the current instance is in *Tcp4State***Closed** state. The return status should be **EFI_ACCESS_DENIED**. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.6.9	0x42145b1a, 0xdd0c, 0x40f8, 0x8f, 0x9a, 0x4c, 0xfc, 0xb6, 0xde, 0x88, 0x2e	**EFI_TCP4_PROTOCOL.Transmit()** – invokes **Transmit()** when the current instance is a passive one and it is in *Tcp4State***Listen** state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Transmit()** when the current instance is a passive one and it is in *Tcp4State***Listen** state. The return status should be **EFI_ACCESS_DENIED**. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.6.10	0xb1618c99, 0xc9c4, 0x4b90, 0x86, 0x4a, 0x8f, 0xa3, 0x32, 0xfd, 0x13, 0xe6	**EFI_TCP4_PROTOCOL.Transmit()** – invokes **Transmit()** when user has called **Close()** to disconnect this connection.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Close()** the disconnect the connection opened in step 3. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet when the connection was disconnected in step 4. The return status should be **EFI_ACCESS_DENIED**. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.6.11	0xb5b0f9ab, 0x04f3, 0x4269, 0x96, 0xa6, 0x40, 0xf5, 0x48, 0xa0, 0x9b, 0x7e	**EFI_TCP4_PROTOCOL.Transmit()** – Tests that the [EUT] correctly handles FIN segment during data transmission.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet and call ReceiveTcpPacket to receive the packet. In addition, send a responding packet with FIN, ACK segment to end one side of the connection. 6. call ReceiveTcpPacket to receive the packet, and send the ack packet. 7. call ReceiveTcpPacket to receive the packet, and send the ack packet for the second time. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.6.12	0x19052fce, 0x5744, 0x470f, 0x8f, 0xc0, 0xc3, 0x84, 0xcc, 0x88, 0x57, 0x1d	**EFI_TCP4_PROTOCOL.Transmit()** –Checks the validity of [PSH] bit during data transimission, by sending 16 bytes data segment to [EUT], with [ENTS] default MSS = 536.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet and call ReceiveTcpPacket to receive the packet. In addition, send a responding packet. 5. Check the *Token.Status*. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.6.13	0x7740ac88, 0x4cf3, 0x4943, 0x9b, 0xf9, 0xec, 0xc4, 0x6a, 0x58, 0xcc, 0x90	**EFI_TCP4_PROTOCOL.Transmit()** –Checks the validity of [PSH] bit during data transimission, by sending 1024 bytes data segment to [EUT], with [ENTS] default MSS = 536.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet and call ReceiveTcpPacket to receive the packet. In addition, send a responding packet. 5. Check the *Token.Status*. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.6.14	0xc6e11d01, 0x485b, 0x4585, 0x9a, 0x2e, 0xcf, 0x43, 0xac, 0x94, 0x2e, 0x1a	**EFI_TCP4_PROTOCOL.Transmit()** –Transmits two fragments.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet and call ReceiveTcpPacket to receive the packet. In addition, send a responding packet. 5. Check the *Token.Status*. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.6.15	0xa5f63716, 0xd4a2, 0x44dc, 0x93, 0x2a, 0xd8, 0xdf, 0x33, 0xd2, 0xa1, 0x65	**`EFI_TCP4_PROTOCOL.Transmit()`** –Transmits more fragments.	1. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Tcp4 child. 2. Call **`EFI_TCP4_PROTOCOL.Configure()`** to configure the new instance. 3. Call **`EFI_TCP4_PROTOCOL.Connect()`** to open an active connection. 4. Call **`EFI_TCP4_PROTOCOL.Transmit()`** to transmit a packet and call ReceiveTcpPacket to receive the packet. In addition, send a responding packet. 5. Check the *`Token.Status`*. 6. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Tcp4 child and clean up the environment.

21.1.6 Receive()

Number	GUID	Assertion	Test Description
5.25.1.7.1	0xe28b3623, 0xc8ba, 0x431a, 0x91, 0xcd, 0xe2, 0xc5, 0x60, 0x36, 0xaa, 0x80	EFI_TCP4_PROTOCOL.Receive – invokes Receive() when the instance has not been configured.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Receive() when the instance has not been configured. 3. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.
5.25.1.7.2	0x484c93a6, 0x93ba, 0x429f, 0x9e, 0x63, 0x0a, 0x7d, 0x5c, 0x19, 0xf5, 0xc7	EFI_TCP4_PROTOCOL.Receive – invokes Receive() with a *Token* value of NULL.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Configure() to configure the new instance. 3. Call EFI_TCP4_PROTOCOL.Connect() to open an active connection. 4. Call EFI_TCP4_PROTOCOL.Receive() with a *Token* value of NULL. The return status should be EFI_INVALID_PARAMETER. 5. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.7.3	0xbe0ff6c1, 0x26a0, 0x4c3f, 0x88, 0xc7, 0xcc, 0xfc, 0x9f, 0xc8, 0xbe, 0x28	**EFI_TCP4_PROTOCOL.Receive** – invokes **Receive()** with a *Token->CompletionToken.Event* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** with a *Token->CompletionToken.Event* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.7.4	0xd0d81b11, 0x23dc, 0x41ac, 0x8c, 0xec, 0xdd, 0x3c, 0x0f, 0x9f, 0x25, 0xef	**EFI_TCP4_PROTOCOL.Receive** – invokes **Receive()** with a *Token->Packet.RxData* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** with a *Token->Packet.RxData* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.7.5	0x6d723765, 0x1345, 0x45ad, 0xb3, 0x57, 0xf0, 0xbc, 0xa1, 0x4c, 0x0c, 0x8f	**EFI_TCP4_PROTOCOL.Receive –** invokes **Receive()** with a *Token->Packet.RxData->DataLength* value of 0.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** with a *Token->Packet.RxData->DataLength* value of 0. The return status should be **EFI_INVALID_PARAMETER**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.7.6	0x1aed8f61, 0xf658, 0x4abb, 0xac, 0x90, 0x04, 0x74, 0x2c, 0x46, 0x87, 0x57	**EFI_TCP4_PROTOCOL.Receive –** invokes **Receive()** with a *Token->Packet.RxData->DataLength* is not the sum of all FragmentBuffer length in *FragmentTable*.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** with a *Token->Packet.RxData->DataLength* value other than the sum of all FragmentBuffer length in *FragmentTable*. The return status should be **EFI_INVALID_PARAMETER**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.7.7	0x2ac8bc18, 0x6c65, 0x4b0d, 0xaf, 0xf1, 0x4f, 0xb5, 0x2e, 0x63, 0xc8, 0x4f	**EFI_TCP4_PROTOCOL.Receive** – invokes **Receive()** when the receive completion token with the same *Token->CompletionToken.Event* was already in the receive queue.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet. 5. Call **EFI_TCP4_PROTOCOL.Receive()** again when the receive completion token with the same *Token->CompletionToken.Event* was already in the receive queue. The return status should be **EFI_ACCESS_DENIED**. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.7.8	0x77f0240a, 0x16a4, 0x471a, 0x95, 0x52, 0xf6, 0x58, 0xf9, 0xbb, 0x11, 0xb1	**EFI_TCP4_PROTOCOL.Receive** – invokes **Receive()** when the current instance is in *Tcp4State*Closed state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. OS send RST segment to let EUT enter *Tcp4State***Closed** state. 5. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet when the instance is in *Tcp4State***Closed** state. The return status should be **EFI_ACCESS_DENIED**. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.7.9	0x276a8e6d, 0xf79a, 0x4cc5, 0xba, 0xcb, 0x99, 0x48, 0x38, 0x59, 0xde, 0xfb	**EFI_TCP4_PROTOCOL.Receive –** invokes **Receive()** when the current instance is a passive one and it is in *Tcp4State*Listen state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Accept()** to accept a connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet when the instance is a passive one and it is in *Tcp4State***Listen** state. The return status should be **EFI_ACCESS_DENIED**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.7.10	0xdde96586, 0xd067, 0x4f04, 0xa0, 0xd9, 0xbd, 0x94, 0x0e, 0x30, 0x97, 0x90	**EFI_TCP4_PROTOCOL.Receive –** invokes **Receive()** when user has called **Close()** to disconnect this connection.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Close()** the disconnect the connection opened in step 3. 5. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet when the connection was disconnected in step 4. The return status should be **EFI_ACCESS_DENIED**. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.7.20	0xc527d95b, 0xbf72, 0x4c94, 0xa8, 0xcc, 0x60, 0x8c, 0x47, 0x04, 0x85, 0x07	**EFI_TCP4_PROTOCOL.Receive –** invokes **Receive()** when the communication peer has closed the connection and there is no any buffered data in the receive buffer of this instance.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Handles the three-way handshake. 5. Configure the OS side to initiate the connection closing. 6. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet. 7. Clean up the environment on EUT side.
5.25.1.7.21	0xc9109f21, 0xd490, 0x4382, 0xbb, 0x22, 0x12, 0xfd, 0x81, 0x67, 0x14, 0xec	**EFI_TCP4_PROTOCOL.Receive –** invokes **Receive()** fails when connection is reseted by the communication peer.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Handles the three-way handshake. 5. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet. 6. Configure the OS side to reset the connection. 7. Clean up the environment on EUT side.

Number	GUID	Assertion	Test Description
5.25.1.7.11	0x36f08e10, 0xbf24, 0x4a97, 0x83, 0x42, 0x99, 0x32, 0x33, 0xff, 0xbe, 0x18	**EFI_TCP4_PROTOCOL.Receive** – invokes **Receive()** to receive a packet.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet and then check the *Token.Status* to verify if the data has been transmitted successfully. The return status should be **EFI_SUCCESS**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.7.12	0xda1653b3, 0xcf85, 0x4152, 0x88, 0x30, 0xd4, 0xbf, 0x54, 0x17, 0x6a, 0x22	**EFI_TCP4_PROTOCOL.Receive** – invokes **Receive()** to receive a packet with two fragment data.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet with two fragment data,and then check the *Token.Status* to verify if the data has been transmitted successfully. The return status should be **EFI_SUCCESS**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.7.13	0xd40ff5f0, 0xcb1d, 0x41cf, 0x8e, 0xab, 0x3f, 0xce, 0xa8, 0x93, 0x3f, 0x4f	**EFI_TCP4_PROTOCOL.Receive** – invokes **Receive()** to receive a packet with ten fragment data.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet with ten fragment data,and then check the *Token.Status* to verify if the data has been transmitted successfully. The return status should be **EFI_SUCCESS**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.7.14	0xf1974d5d, 0x5860, 0x4519, 0x8b, 0x8f, 0x78, 0xce, 0x0a, 0xad, 0xbb, 0xec	**EFI_TCP4_PROTOCOL.Receive** – Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(no overlaps).	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet.The return status should be **EFI_SUCCESS**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.7.15	0xc9d79086, 0x5eb8, 0x4c76, 0xa4, 0xc4, 0xf1, 0xfe, 0x78, 0x6f, 0xc0, 0x31	**EFI_TCP4_PROTOCOL.Receive –** Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(the second head overlaps the first tail).	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet.The return status should be **EFI_SUCCESS**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.7.16	0x3c0cc77e, 0xfb9b, 0x4b24, 0x85, 0xd0, 0xaf, 0x3f, 0x39, 0xc8, 0xfd, 0xb7	**EFI_TCP4_PROTOCOL.Receive –** Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(the second segment is included in the middle of the first one).	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet.The return status should be **EFI_SUCCESS**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.7.17	0x5252cae8, 0xb23b, 0x456e, 0x97, 0xdf, 0x1c, 0x01, 0xdd, 0xc4, 0xcd, 0x05	**EFI_TCP4_PROTOCOL.Receive –** Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(the third segment is included in the head of the second one).	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet.The return status should be **EFI_SUCCESS**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.7.18	0x8a11bbca, 0xe267, 0x4221, 0xa5, 0x50, 0x33, 0x62, 0x33, 0x88, 0xeb, 0x06	**EFI_TCP4_PROTOCOL.Receive –** Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(the third segment is included in the middle of the second one).	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet.The return status should be **EFI_SUCCESS**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.7.19	0x794eff7b, 0xb88f, 0x4f67, 0x9d, 0xa1, 0xd5, 0x0e, 0xa6, 0xbc, 0x5c, 0x37	**EFI_TCP4_PROTOCOL.Receive –** Checks if EFI TCP4 could correctly handle the current segment overlaps with previous segment(the first and the second segment is joined by the third one).	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet.The return status should be **EFI_SUCCESS**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.7 Close()

Number	GUID	Assertion	Test Description
5.25.1.8.1	0xc92fad2d, 0x446d, 0x43d7, 0xaf, 0xbe, 0x81, 0xce, 0x03, 0xd4, 0xe8, 0x12	`EFI_TCP4_PROTOCOL.Close` – invokes `Close()` when the instance has not been configured.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Close()` to close a connection when the instance has not been configured. The return status should be `EFI_NOT_STARTED`. 3. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.8.2	0x82827716, 0xb622, 0x4527, 0xb8, 0x9e, 0xa5, 0x30, 0x59, 0xce, 0xc9, 0xec	`EFI_TCP4_PROTOCOL.Close` – invokes `Close()` when `Configure()` has been called with *`TcpConfigData`* set to `NULL` and this function has not returned.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` with *TcpConfigData* set to `NULL`. 3. Call `EFI_TCP4_PROTOCOL.Close()` when the `Configure()` function has not returned. The return status should be `EFI_ACCESS_DENIED`. 4. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.8.3	0x9f19e873, 0x71a5, 0x4350, 0xa6, 0xb5, 0xa9, 0x96, 0x8c, 0x64, 0xe6, 0xde	`EFI_TCP4_PROTOCOL.Close` – invokes `Close()` when the previous `Close()` call on this instance has not finished.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to open an active connection. 4. Call `EFI_TCP4_PROTOCOL.Close()` to disconnect the connection opened in step 3. 5. Call `EFI_TCP4_PROTOCOL.Close()` when the previous `Close()` call on this instance has not finished. The return status should be `EFI_ACCESS_DENIED`. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.8.4	0xa9472aa1, 0xfff1, 0x4130, 0x90, 0xc9, 0xf8, 0x87, 0x69, 0x8f, 0x8b, 0xc1	**EFI_TCP4_PROTOCOL.Close** – invokes **Close()** with a *CloseToken* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Close()** with a *CloseToken* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.8.5	0x09caa34e, 0xdf4f, 0x4dcf, 0xbe, 0x5b, 0x7b, 0xe3, 0xf3, 0x68, 0x90, 0xc0	**EFI_TCP4_PROTOCOL.Close** – invokes **Close()** with a *CloseToken->CompletionToken.Event* value of **NULL**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Close()** with a *CloseToken->CompletionToken.Event* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.8.6	0x3756329a, 0x21c3, 0x41c6, 0xa1, 0x03, 0x15, 0x9a, 0x57, 0x93, 0x8e, 0x9f	**EFI_TCP4_PROTOCOL.Close –** invokes **Close()** as function test. After user called **Configure()** with **NULL** without close stopping, the *CloseToken.CompletionToken.Status* should be **EFI_ABORTED**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Close()** to disconnect the connection opened in step 3. The return status should be **EFI_SUCCESS**. 5. Call **EFI_TCP4_PROTOCOL.Configure()** with **NULL** without close stopping, then verify the *CloseToken.CompletionToken.Status* to be **EFI_ABORTED**. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.8.7	0x499852f9, 0x49c2, 0x4168, 0x8f, 0x90, 0xab, 0x97, 0x0f, 0x06, 0x53, 0x0b	**EFI_TCP4_PROTOCOL.Close –** invokes **Close()** as function test. Abort the TCP connection on close instead of the standard TCP close process by setting the *AbortOnClose* to **TRUE**.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection. 4. Call **EFI_TCP4_PROTOCOL.Close()** to disconnect the connection opened in step 3 with *AbortOnClose* set to **TRUE**. The return status should be **EFI_SUCCESS**. Then verify *Token.Status* has been updated to **EFI_ABORTED**. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.8 CnntClosing

Number	GUID	Assertion	Test Description
5.25.1.13.1	0xc9fa5b59, 0x7a1c, 0x4b2b, 0x9b, 0xce, 0x6b, 0xad, 0x38, 0x12, 0x2b, 0x0d	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly handle the closing connection when it initiates the closing.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection, and then handles the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Close()** to do a graceful close working flow, then check the *Token.Status* to verify the connection has been closed. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.13.2	0x8ae1e58b, 0xcd65, 0x4fb0, 0xba, 0x12, 0x43, 0x95, 0xef, 0xab, 0x9c, 0xd1	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly handle the closing connection when [OS] initiates the closing.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection, and then handles the three-way handshake. 4. Configure the [OS] to initiate the connection closing. 5. Call **EFI_TCP4_PROTOCOL.Close()** to do a graceful close working flow, then check the *Token.Status* to verify the connection has been closed. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.13.3	0x8b1bcbd7, 0x3db6, 0x46ec, 0x8b, 0xf0, 0x84, 0xb4, 0xb9, 0x0f, 0xb8, 0x95	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly handle the simultaneous closing connection.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection, and then handles the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Close()** to do a graceful close working flow, then check the *Token.Status* to verify the connection has been closed. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.13.4	0xebc0e165, 0x3146, 0x4fa1, 0x9a, 0xd8, 0x6d, 0x56, 0xdf, 0xb0, 0x9f, 0xd6	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly handle the reception of normal data segments after having already received partner's FIN segment.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection, and then handles the three-way handshake. 4. Configure the [OS] to initiate the connection closing. Then configure the [OS] to send data segments to the [EUT]. 5. Call Tcp.**GetModeData()**, and there is a expectation that EUT should return to CLOSE state. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.13.5	0x9530e11a, 0x4d42, 0x4c45, 0x9e, 0xe9, 0x30, 0x82, 0xfc, 0xc9, 0x0f, 0x97	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that correctly handle the reception of unacceptable data segments after having already received partner's FIN segment.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection, and then handles the three-way handshake. 4. Configure the [OS] to initiate the connection closing. Then configure the [OS] to send data segments to the [EUT]. 5. Call **EFI_TCP4_PROTOCOL.Close()** to do a graceful close working flow, then check the *Token.Status* to verify the connection has been closed. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.13.6	0x8cb38a66, 0xfb72, 0x4dce, 0x94, 0x8b, 0x3e, 0x8f, 0xae, 0x66, 0x6f, 0x98	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly perform the retransmission of FIN segment during the connection closing process.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection, and then handles the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Close()** to do a graceful close working flow. EUT should timeout 3 times and follow the sequence: ,6,12 ...then check the *Token.Status* to verify the connection has been closed. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.13.7	0xc9ef7a67, 0xc2a7, 0x40b4, 0xa9, 0x31, 0xba, 0x7a, 0x83, 0x16, 0x53, 0x15	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] that can correctly handle the half-close of the communication peer. If your peer still wants to send data after sending out `FIN`, EUT should ignore the data and interact with the peer correctly.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to open an active connection, and then handles the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet. 5. OS get the transmitted data packet and respond with `FIN`, `ACK` segment to end one side of the connection. 6. Expand the receive window together with data in the segment, EUT should ignore the data and interact with the peer correctly. 7. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.13.8	0xc4e81c62, 0xe709, 0x4096, 0xbb, 0xfb, 0x59, 0x99, 0x07, 0xaf, 0x89, 0x82	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that correctly support partner's half-close.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection, and then handles the three-way handshake. 4. Configure the [OS] to initiate the connection closing. Then Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. 5. OS get the transmitted data packet and check the *Token.Status* to verify the data has been sent out. 6. Call **EFI_TCP4_PROTOCOL.Close()** to close the connection, then check the *Token.Status* to verify the connection has been closed. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.13.9	0x37b8e036, 0x3ff9, 0x4401, 0x81, 0x76, 0xa5, 0x70, 0xd9, 0x16, 0xa9, 0x4e	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that correctly wait a **2xMSL** timeout period while it has initiated the closing of a connection.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection, and then handles the three-way handshake. Check the *Token.Status* to verify the connection has been established 4. Call **EFI_TCP4_PROTOCOL.Close()** to do a graceful close working flow. 5. Call **EFI_TCP4_PROTOCOL.Connect()** to reopen the connection when [EUT] is still in **TIME-WAIT** state. The return status should be **EFI_ACCESS_DENIED**. 6. Check the *Token.Status* to verify the connection has been closed. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.13.10	0x2c9f0ffe, 0xf355, 0x4a2f, 0xb6, 0xa2, 0xbf, 0x84, 0x6c, 0xe8, 0x33, 0x2f	EFI_TCP4_PROTOCOL – Tests that the [EUT] that can correctly handle a valid SYN segment while it is in TIME-WAIT state.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Configure() to configured the new instance. 3. Call EFI_TCP4_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake. Check the *Token.Status* to verify the connection has been established 4. Call EFI_TCP4_PROTOCOL.Close() to do a graceful close working flow. 5. Send a SYN segment with a larger sequence number than the previous connection contained. If the SYN is not in the window, an ACK should be sent out. 6. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.
5.25.1.13.11	0xaaf0c2ad, 0x5433, 0x46cf, 0xa4, 0xd9, 0xc3, 0xea, 0x65, 0xe1, 0x38, 0xfc	EFI_TCP4_PROTOCOL – Tests that the [EUT] that can correctly handle the buffered receive data when application already performed active close. The buffered data should be removed and RST segment should be sent out.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Configure() to configured the new instance. 3. Call EFI_TCP4_PROTOCOL.Connect() to open an active connection, and then handles the three-way handshake. Configure the [OS] to send data segments to the [EUT]. 4. Call EFI_TCP4_PROTOCOL.Close() to close the connection. The [EUT] should send out a RST segment. 5. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.13.12	0x7996049d, 0xc63f, 0x4bb4, 0x96, 0xa2, 0xb1, 0x90, 0xe7, 0x35, 0x8c, 0x3c	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly handle the send buffered data when application has already performed active close.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open an active connection, and then handles the three-way handshake. 4. Create event and configuration for transmit and close interface invoking. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Then [OS] get the transmitted data packet. 6. Call **EFI_TCP4_PROTOCOL.Close()** to close the connection. Then configure the [OS] to interact data transmission with the [EUT]. The last segment should have the **FIN** flag set. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.13.13	0xa740c41c, 0xa9b1, 0x4194, 0x8a, 0xf5, 0x6c, 0x92, 0xd9, 0x20, 0xc7, 0x78	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly handle and receive the data segment in **<SYN>** and **<FIN, ACK>** segments, receive all the data (throw down a receive token) after data transmission finished.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the passive TCP instance, then handles the three-way handshake. Check the *Token.Status* to verify the connection has been established. 4. Configure OS to send data together with FIN flag set. Then Call **EFI_TCP4_PROTOCOL.Receive()** to receive the data sent with the **SYN** and **<FIN, ACK>** segment. Check the received segment data length. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.13.14	0xd012d6bb, 0x9dac, 0x4e3b, 0xa5, 0x54, 0xf6, 0xe9, 0xf5, 0x77, 0x22, 0xb4	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] that can correctly handle and receive the data segment in `<SYN>` and `<FIN, ACK>` segments, and receive all the data (throw down a receive token) before data transmission.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as a passive one 3. Call `EFI_TCP4_PROTOCOL.Accept()` for the passive TCP instance, then handles the three-way handshake. Check the *`Token.Status`* to verify the connection has been established. 4. Call `EFI_TCP4_PROTOCOL.Receive()` to receive the data sent with the `SYN` and `<FIN, ACK>` segment. Then configure OS to send data together with FIN flag set. Check the received segment data length. 5. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

21.1.9 CnntOpening

Number	GUID	Assertion	Test Description
5.25.1.14.1	0x156e08bb, 0x21c4, 0x48a0, 0xbe, 0xc0, 0x8d, 0x0c, 0x17, 0x7b, 0x90, 0xf2	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly receive and handle the SYN segment with data.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the passive TCP instance, then handles the three-way handshake. Check the *Token.Status* to verify the connection has been established. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive the data sent with the **SYN** segment. Then check the received segment data length. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.14.2	0xd7814ee7, 0x2cc3, 0x4cc6, 0xb4, 0x3c , 0x54, 0x7e, 0x1f, 0x73, 0xc3, 0x12	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly establish the TCP connection through active open.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the passive TCP instance, then handles the three-way handshake. Check the *Token.Status* to verify the connection has been established. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.14.3	0xeac7fe49, 0x5202, 0x457f, 0x9e, 0x77, 0x49, 0xe5, 0x77, 0xa1, 0x4b, 0x4e	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly establish the TCP connection through active open. This connection should not affect any previously established connection.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the active TCP instance, then handles the three-way handshake. Check the *Token.Status* to verify the connection has been established. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child for the second connection.. 5. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the second instance as an active one 6. Call **EFI_TCP4_PROTOCOL.Connect()** for the second active TCP instance, then handles the three-way handshake. Check the *Token.Status* to verify the connection has been established. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.14.4	0xc5678e42, 0x6d91, 0x41c1, 0x96, 0x2d, 0xb6, 0x7b, 0xaa, 0x72, 0xf8, 0x21	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly establish the TCP connection through passive open with unspecified address/ port pair. This connection should not affect any previously established connection.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the passive TCP instance, then handles the three-way handshake. Check the *Token.Status* to verify if the connection has been established. 4. Try to establish TCP connection with unspecified address/port pair. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.14.5	0x3131d110, 0x7545, 0x46c5, 0x91, 0xd1, 0x87, 0x01, 0xd3, 0x04, 0x7f, 0xcf	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly establish the TCP connection through passive open with specified address/port pair. This connection should not affect any previously established connection.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the passive TCP instance, then handles the three-way handshake. Check the *Token.Status* to verify if the connection has been established. 4. Try to establish TCP connection with unspecified address/port pair. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.14.6	0x165ad06c, 0xf630, 0x4516, 0x95, 0xba, 0x90, 0x3f, 0xd8, 0xa2, 0x4d, 0xe4	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly establish the TCP connection through simultaneous open. It performs the following interactions: A ------<SYN>------> B A <-----<SYN>------- B A --<SYN, ACK>--> B A <-----<ACK>------- B	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the active TCP instance, then handle the three-way handshake. Check the *Token.Status* to verify if the connection has been established. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.14.7	0x2328abeb, 0x2dca, 0x4960, 0xa0, 0x93, 0x42, 0x94, 0xc8, 0x8c, 0x3d, 0x51	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly establish the TCP connection through simultaneous open. This connection should not affect any previously established connection.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance, then handle the three-way handshake and check the *Token.Status* to verify the connection has been established. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child for the second connection. 5. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the second instance as an active one. 6. Call **EFI_TCP4_PROTOCOL.Connect()** for the second instance, then handle the three-way handshake and check the *Token.Status* to verify if the connection has been established. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.14.9	0xe39e864a, 0x347d, 0x4c08, 0xa7, 0xec, 0x0e, 0x55, 0x34, 0xe8, 0xa0, 0x20	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] that can correctly time out when waiting a TCP connection to be established in `SYN_SENT` state.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance, and during 60 seconds, EUT should timeout following the sequence: 3, 6, 12, 24…. 4. Check the *`Token.Status`* to verify the connection has been timeouted. 5. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.14.10	0x697d126d, 0xd496, 0x448b, 0x85, 0x08, 0x60, 0x6d, 0xc1, 0xc6, 0x3f, 0x65	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] that can correctly time out when waiting a TCP connection to be established in `SYN_SENT` state.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance, and during 60 seconds, EUT should timeout following the sequence: 3, 6, 12, 24….In addition, EUT should send out RST segment and return to `CLOSED` state. 4. Check the *`Token.Status`* to verify the connection has been timeouted. 5. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.14.12	0xb22365c7, 0x6daa, 0x48e9, 0xa3, 0x7a, 0x1d, 0xe5, 0x47, 0xf4, 0x04, 0x4e	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly establish the TCP connection through simultaneous open. It performs the following interactions: A ------<SYN>------> B A <-----<SYN>------- B A ----<SYN, ACK>---> B A <---<SYN, ACK>---- B	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance, then handle the three-way handshake. 4. Check the *Token.Status* to verify the connection has been established. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.14.13	0x8e4d9bac, 0x42b6, 0x408f, 0xa2, 0x44, 0xd3, 0xfe, 0x9b, 0xdc, 0x0c, 0xc7	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly establish the TCP connection through simultaneous open. It performs the following interactions: A ------<SYN>------> B A <-----<SYN>------- B A <---<SYN, ACK>---- B A ----<SYN, ACK>---> B	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance, then handle the three-way handshake. 4. Check the *Token.Status* to verify the connection has been established. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.14.14	0x72d8a37d, 0x312e, 0x44ee, 0x86, 0xcb, 0xb5, 0x58, 0x5c, 0x63, 0x6d, 0x65	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly receive and handle the <SYN, ACK> segment with data, throw down receive token after data transmission.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance, then handle the three-way handshake. 4. Check the *Token.Status* to verify the connection has been established. 5. Call **EFI_TCP4_PROTOCOL.Receive()** to receive the data sent with the SYN segment. Then check the segment data length. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.14.15	0xe0c87d8a, 0x81d4, 0x4634, 0xa2, 0x0a, 0xee, 0xba, 0xdc, 0x44, 0x96, 0xe6	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly receive and handle the <SYN, ACK> segment with data, throw down receive token before data transmission.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Receive()** to receive the data sent with the SYN segment. 4. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance, then handle the three-way handshake. 5. Check the *Token.Status* to verify the connection has been established. 6. Get the received segment datalength to check the correction. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.14.16	0x13f5c5e1, 0xd4dc, 0x437d, 0xac, 0xa2, 0x93, 0x1a, 0x8d, 0x85, 0xe0, 0xd3	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly handle the flag combination: ACK, FIN through active open.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance, then handle the three-way handshake. In addition, EUT should ignore this unexpected segment and retransmit the SYN segment. 4. Check the *Token.Status* to verify the connection has been established. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.14.17	0x656575ec, 0x018b, 0x475a, 0x80, 0xa0, 0xff, 0x32, 0xef, 0x50, 0x31, 0x74	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly handle the flag combination: FIN, ACK through passive open.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance, then handle the three-way handshake. In addition, the data sent together with the FIN,ACK segment should be processed. 4. Check the *Token.Status* to verify the connection has been established. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.14.18	0xcaba9876, 0xc926, 0x42b3, 0xaf, 0x99, 0xb5, 0x7d, 0x71, 0x83, 0x62, 0x20	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly handle the flag combination: SYN, FIN, ACK through passive open.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance, then handle the three-way handshake. 4. Handle the normal three-way handshake. Then check the *Token.Status* to verify the connection has been established. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.14.19	0xcd97a722, 0xc8fe, 0x4584, 0xb3, 0x9c, 0x65, 0x9b, 0xbb, 0x2c, 0x5a, 0x6f	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that can correctly refuse the attempted connections from broadcast and multicast address.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance, then handle the three-way handshake. 4. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.10 CongestionCtrl

Number	GUID	Assertion	Test Description
5.25.1.15.1	0xb0cdf9b2, 0x0cc0, 0x4e99, 0x96, 0x83, 0xde, 0xf3, 0x96, 0xc1, 0xc6, 0xa7	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that correctly perform the slow start at the beginning of the connection transmission.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. 5. OS get the transmitted data packet and interact with EUT to expand the cwnd. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.15.2	0x05d19fac, 0x66e6, 0x4f41, 0xba, 0x70, 0xff, 0x3e, 0x48, 0x7f, 0x4d, 0x4a	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that correctly perform the slow start and congestion avoidance algorithms when data segment timeout causes congestion.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. 5. OS gets the transmitted data segments of the fist stage, and check the token status of transmit interface, then begin the second stage data transmission. 6. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit data segments as the second stage. 7. Wait for data retransmission and send back the ACK to all the transmitted data segments. In addition, EUT should enter slow start. 8. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.15.4	0xc12b24da, 0xa3c5, 0x4820, 0x81, 0x98, 0x6e, 0x34, 0xad, 0x28, 0xfc, 0xaf	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] that correctly perform the slow start and congestion avoidance algorithms when **SYN** segment timeout causes congestion.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** to for the instance. 4. Handle the three-way handshake. Configure the [OS] to ignore the first **SYN** segment and wait for the *ConnectionTimeout* seconds. When received the second **SYN** segment, make the [OS] send back the **SYN**, **ACK** segment. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. 6. OS get the transmitted data packet and interact with EUT to expand the cwnd. In addition, check the token status of transmit interface. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.15.5	0xf5c35856, 0x3c84, 0x40ce, 0xba, 0xf4, 0x91, 0x57, 0x7e, 0xfa, 0x44, 0x98	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] that correctly performs the fast retransmit and fast recovery algorithms receiving 3 or above duplicated acknowledgements. When an ACK arrives that acknowledges new data, this ACK is Full acknowledgements.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to open a new connection for the new instance. Then handle the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet. 5. OS get the transmitted data segments of the first stage, and check the token status of transmit interface, then begin the second stage data transmission. 6. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet as the second stage. 6. The cwnd should be expanded to 11*SMSS after the 1st stage data transmission. The second stage of data transmission includes 8192 (16*MSS) bytes data. Configure the OS to generate consecutive duplicate ACKs. 7. Configure the OS to acknowledge the last data segment and EUT will end the fast recovery and enter the congestion avoidance again. 8. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.15.6	0x0df29ac1, 0x5b58, 0x49cc, 0x95, 0x31, 0xce, 0xce, 0xb4, 0x49, 0xb5, 0x3a	**`EFI_TCP4_PROTOCOL`** – Tests that the [EUT] that correctly generate duplicated acknowledgements when it received disordering segments.	1. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Tcp4 child. 2. Call **`EFI_TCP4_PROTOCOL.Configure()`** to configured the new instance as an active one. 3. Call **`EFI_TCP4_PROTOCOL.Connect()`** to open a new connection for the new instance. Then handle the three-way handshake. 4. Configure the OS to send consecutive data segments to the EUT, drop one segment in the middle and EUT should generate duplicated ACKs as the result of receiving every data segments. 5. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.15.7	0x3a4fb624, 0x8b05, 0x46ce, 0x97, 0xd7, 0x0f, 0xc9, 0x1e, 0x5d, 0x37, 0x6a	**`EFI_TCP4_PROTOCOL`** – Tests that the [EUT] that correctly performs the fast retransmit and fast recovery algorithms receiving 3 or above duplicated acknowledgements. After exiting the fast recovery, [EUT] should enter congestion avoidance.	1. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Tcp4 child. 2. Call **`EFI_TCP4_PROTOCOL.Configure()`** to configured the new instance as an active one. 3. Call **`EFI_TCP4_PROTOCOL.Connect()`** to open a new connection for the new instance. Then handle the three-way handshake. 4. Call **`EFI_TCP4_PROTOCOL.Transmit()`** to transmit data segments of the first stage. 5. OS get the transmitted data segments of the fist stage, check the token status of transmit interface. Then begin the second stage data transmission. 6. Call **`EFI_TCP4_PROTOCOL.Transmit()`** to transmit data segments of the second stage. 7. The cwnd should be expanded to 11*SMSS after the 1st stage data transmission. The second stage of data transmission includes 8192 (16*MSS) bytes data. Configure the OS to generate consecutive duplicate ACKs. 8. Call **`EFI_TCP4_PROTOCOL.Transmit()`** to transmit data segments of the third stage. The third stage of data transmission should perform congestion avoidance. 9. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.15.8	0xa4d6bd97, 0x6d30, 0x4fec, 0x8b, 0x50, 0xcf, 0xac, 0xb1, 0x7e, 0x9e, 0x0a	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] that correctly performs the NewReno modification to TCP's fast recovery algorithm. After the first fast recovery, when an ACK arrives that acknowledges new data, this ACK is partial acknowledgements.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to open a new connection for the new instance. Then handle the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit data segments of the first stage. 5. OS get the transmitted data segments of the fist stage, check the token status of transmit interface. Then begin the second stage data transmission. 6. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit data segments of the second stage. 7. The cwnd should be expanded to 11*SMSS after the 1st stage data transmission. The second stage of data transmission includes 8192 (16*MSS) bytes data. Configure the OS to generate consecutive duplicate ACKs. 8. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

21.1.11 NagleSWSA

Number	GUID	Assertion	Test Description
5.25.1.16.1	0xceef47a7, 0xf194, 0x4200, 0x9a, 0xbc, 0xe2, 0x9d, 0xfe, 0x80, 0xaa, 0x49	EFI_TCP4_PROTOCOL – Tests that the [EUT] correctly disables the Nagle Algorithm.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Configure() to configured the new instance as an active one, and disable the Nagle control option. 3. Call EFI_TCP4_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake. 4. Call EFI_TCP4_PROTOCOL.Transmit() to transmit a small segment. 5. Call EFI_TCP4_PROTOCOL.Transmit() to transmit another small segment. 6. OS gets the first transmitted data packet, and the 2nd segment should be sent out immediately. 7. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.16.2	0x3906f7fa, 0xbe7b, 0x435a, 0xb6, 0x78, 0x1d, 0x5b, 0xba, 0xe5, 0x51, 0x4a	**EFI_TCP4_PROTOCOL – ** Tests that the [EUT] correctly disables the Nagle Algorithm. When retransmission happens, the accumulated small segments should be sent out together.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one, and disable the Nagle control option. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a small segment. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit another small segment. 6. As Nagle is disabled, the two segments should be sent out immediately. In addition, they should be sent out separately during retransmission. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.16.3	0xa528b7a1, 0x23cb, 0x4601, 0xb2, 0x74, 0xd7, 0x0b, 0xcc, 0x17, 0x5e, 0x42	EFI_TCP4_PROTOCOL – Tests that the [EUT] correctly handles the small segments in accord with Nagle algorithm.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Configure() to configured the new instance as an active one, and enable the Nagle control option. 3. Call EFI_TCP4_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake. 4. Call EFI_TCP4_PROTOCOL.Transmit() to transmit three small segment. 5. OS get the first transmitted data segment and send back ACK segment. As Nagle is enabled, the last two segments should be sent out together. 6. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.
5.25.1.16.4	0x0d5581c0, 0x6903, 0x4387, 0xaf, 0xf7, 0xe3, 0x2c, 0xac, 0x17, 0xee, 0x33	EFI_TCP4_PROTOCOL – Tests that the [EUT] correctly handles the small segments in accord with Nagle algorithm. When retransmission happens, the accumulated small segments should be sent out together.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Configure() to configured the new instance as an active one, and enable the Nagle control option. 3. Call EFI_TCP4_PROTOCOL.Connect() to open a new connection for the new instance. Then handle the three-way handshake. 4. Call EFI_TCP4_PROTOCOL.Transmit() to transmit three small segment. 5. OS get the first transmitted data segment and as Nagle is enabled, the last two segments should be sent out together. 6. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.16.5	0xabf756ac, 0x54a7, 0x492c, 0xae, 0xa6, 0x6d, 0x46, 0xd7, 0x44, 0xb8, 0x72	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles the bulk data flow, the [EUT] should not respond with an acknowledgement segment for each of the received segments. In a stream of full-sized segments there should be an ACK for at least every second segment.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one, and disable the Nagle control option. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the passive instance. Then handle the three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open. 4. Configure the [OS] to send 10 full-sized data segments. There should be at least an ACK for every second segment. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.16.6	0x94c3ee05, 0x142e, 0x4f2e, 0x8a, 0x9a, 0x8f, 0x05, 0x25, 0xbb, 0xb4, 0x83	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles the bulk data flow, the [EUT] should not respond with an acknowledgement segment for each of the received segments. A TCP should implement a delayed ACK, but an ACK should not be excessively delayed.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one, and disable the Nagle control option. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the passive instance. Then handle the three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open. 4. EUT should delay ACK the data segment, but the delay MUST be less than 0.5 second. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.16.7	0x81d74381, 0xb0df, 0x4ef3, 0x8a, 0x1c, 0xdc, 0x7b, 0xe9, 0x60, 0xc6, 0x02	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles the bulk data flow, the [EUT] should not respond with an acknowledgement segment for each of the received segments. In a stream of single-byte segments there should be an ACK for at least every second segment.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one, and disable the Nagle control option. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the passive instance. Then handle the three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open. 4. Configure the [OS] to send 20 single-byte data segments. There should be at least an ACK for every second segment. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.16.8	0xd7c7813e, 0x4624, 0x4f11, 0xb3, 0x65, 0x45, 0x6e, 0x00, 0x30, 0x30, 0xe2	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly avoids the Silly Window Syndrome as the TCP receiver. The receiver should not advertise a larger window until the window can be increased at least one full-sized segment.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as a passive one, and disable the Nagle control option. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the passive instance. Then handle the three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open. 4. Configure the [OS] to send 4 data segment to fill the receive buffer. 5. Call Receive interface to get one full-sized data. 6. Get the Window expansion segment. Then send another 1024-bytes data to refill the EUT receive buffer. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.16.9	0xf853dee2, 0xa900, 0x417b, 0xb5, 0xce, 0x80, 0x86, 0x55, 0x17, 0xab, 0x57	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly avoids the Silly Window Syndrome as the TCP sender. The sender should not transmit unless everything can be sent out and no need to wait ACK.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one, and enable the Nagle control option. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the active instance. Then handle the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a small segment. 5. OS gets the EUT transmitted data segment. In addition, EUT should send out all the left data segments. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.16.10	0x93015811, 0x2c00, 0x4834, 0x83, 0x17, 0x7b, 0xbf, 0x7f, 0x1a, 0xcb, 0x52	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly avoids the Silly Window Syndrome as the TCP receiver. The sender should not transmit unless everything can be sent out and Nagle algorithm is disabled.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one, and disable the Nagle control option. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the active instance. Then handle the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a small segment. 5. OS gets the EUT transmitted data segment. In addition, configure the OS to acknowledge the second segment and advertise enough window to let EUT transmit all the left data segments. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.16.11	0xfa149507, 0x1607, 0x44da, 0xb2, 0xae, 0x5f, 0xd3, 0x51, 0x7d, 0x82, 0xba	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly avoids the Silly Window Syndrome as the TCP receiver. The sender should not transmit unless a full-sized segment can be sent.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one, and disable the Nagle control option. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the active instance. Then handle the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a small segment. 5. EUT should set persist timer, configure OS to increase the window size to exceed 512 bytes before the persist timer times out. In addition, repeat the steps before finishing the data transmission. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.16.12	0xceb5c9e5, 0xebce, 0x4486, 0xb5, 0xc5, 0x06, 0xa6, 0x0c, 0x36, 0x5e, 0xa6	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly avoids the Silly Window Syndrome as the TCP receiver. The sender should not transmit unless at least one-half of the Max Window that receive ever advertised.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one, and disable the Nagle control option. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the active instance. Then handle the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a small segment. 5. EUT should set persist timer, configure OS to increase the window size by 256 octets consecutively. Make sure the windows size exceed one-half of the Max Window that receive ever advertised before persist timer times out. 6. Increase the windows size step by step, when it accesses the left data size, EUT should send out the left buffered data at one time. 7. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

21.1.12 UrgHandling

Number	GUID	Assertion	Test Description
5.25.1.12.1	0x355d3648, 0x8375, 0x4b16, 0x94, 0xc4, 0x19, 0xe1, 0xbc, 0x87, 0xfc, 0x8b	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly uses the urgent pointer to denote the last urgent octet of urgent data.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to open a new connection for the new instance. Then handle the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit data segments. 5. Get the transmitted data segment and check the urgent pointer, it should point to the sequence number of the last octet. Then check the token status of transmit interface. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.12.2	0x03663fa9, 0x0a34, 0x43a5, 0x84, 0x5b, 0x2c, 0x36, 0x7f, 0x7e, 0xb6, 0xd8	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly uses the urgent pointer to denote the last urgent octet of urgent data. The urgent data exceeds the maximum number of urgent pointer.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit data segments with the length 65536. 5. Get the transmitted data segment and check the urgent pointer. 6. The urgent pointer will rollback but the EUT should maintain the correct value of the urgent pointer. After sending out the first data segment, EUT should send the second data segment with urgent pointer 65024(65536 – 512). 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.12.3	0xfce0e13a, 0x35df, 0x4713, 0xaf, 0xb8, 0x4d, 0x1e, 0xcc, 0xa5, 0x82, 0x9b	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly uses the urgent pointer to denote the last urgent octet of urgent data. The urgent pointer rollbacks for two times.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to open a new connection for the new instance. Then handle the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit data segments with the length 131401. 5. OS get the transmitted data packet and interact with EUT to expand the cwnd. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.12.4	0x75f47641, 0x2982, 0x4d51, 0x95, 0x3b, 0x4b, 0x65, 0x91, 0x73, 0x5e, 0x76	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly receives urgent data segments of updated and variable lengths. OS sends some urgent data between normal data transmission.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to open a new connection for the new instance. Then handle the three-way handshake. 4. Configure the OS to send normal data including urgent data segments. 5. Call `EFI_TCP4_PROTOCOL.Receive()` to receive the normal data and get the received segment data length to check the correction. 6. Call `EFI_TCP4_PROTOCOL.Receive()` to receive the first section of urgent data. Get the received segment data length to check the correction. 7. Send the remained urgent data and normal data. 8. Call `EFI_TCP4_PROTOCOL.Receive()` to receive the second section of urgent data and the remained normal data. Check the data length. 9. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.12.5	0xd0f54967, 0xaa9b, 0x4017, 0x87, 0x87, 0x24, 0xfb, 0x34, 0x9d, 0xe4, 0x51	**EFI_TCP4_PROTOCOL –** Tests that the [EUT] correctly receives urgent data segments of updated and variable lengths. OS sends some urgent data in the SYN segment.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Receive()** to receive the normal data and get the received segment data length to check the correction. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.12.6	0x4cbb57e5, 0xe348, 0x4340, 0x81, 0x9e, 0xed, 0x61, 0x5a, 0xc2, 0x1a, 0x35	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly receives urgent data segments of updated and variable lengths. The urgent pointer just points to the sequence of FIN flag.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configured the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to open a new connection for the new instance. Then handle the three-way handshake. 4. Configure the OS to send normal data including urgent data segments. 5. Call `EFI_TCP4_PROTOCOL.Receive()` to receive the data segments, and check the data length. 6. Call `EFI_TCP4_PROTOCOL.Receive()` to receive the urgent data segments, and check the data length. 7. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.12.7	0x6145a7f3, 0xbb3d, 0x48e8, 0xab, 0xdf, 0x90, 0xc9, 0x87, 0x82, 0xdc, 0x25	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly receives urgent data segments of updated and variable lengths. The urgent pointer exceeds the sequence of FIN flag.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Configure the OS to send normal data including urgent data segments, and make the urgent pointer exceed the sequence if FIN flag.. 5. Call **EFI_TCP4_PROTOCOL.Receive()** to receive the data segments, and check the data length. 6. Call **EFI_TCP4_PROTOCOL.Receive()** to receive the urgent data segments, and check the data length. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.12.8	0x73cf4c9a, 0x8c1d, 0x4b7f, 0x94, 0x7c, 0x7f, 0x74, 0x06, 0xf5, 0x10, 0x1d	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handle the urgent data transmission when communication peer's receive window is 0.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. 5. After OS got the transmitted data packet, Make the [OS] send an acknowledge segment with a 0 window. Then check whether EUT can still send out data segment or not. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.12.9	0x6019f57b, 0xd99f, 0x47b4, 0x94, 0x4a, 0x86, 0x80, 0x3e, 0x55, 0x63, 0x54	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handle the urgent data transmission when communication peer's receive window is 0.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. 5. After OS got the transmitted data packet, Make the [OS] send an acknowledge segment with a 0 window. 6. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit an urgent packet. Then check whether EUT can still send out data segment or not. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.13 RstHandling

Number	GUID	Assertion	Test Description
5.25.1.17.1	0x1dd96986, 0x44c7, 0x4981, 0xba, 0x01, 0x14, 0x73, 0xff, 0x82, 0xb2, 0xed	`EFI_TCP4_PROTOCOL` – Tests that the <EUT> correctly send out the reset segment while in <CLOSED> state.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to open a new connection for the new instance. Then handle the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Close()` to do a graceful close working flow. 5. Call `EFI_TCP4_PROTOCOL.GetModeData()` to validate RST generation in <CLOSED> state. 6. In <CLOSED> state, check OS send SYN, and EUT respond with RST. 7. In <CLOSED> state, check OS send FIN, and EUT respond with RST. 8. In <CLOSED> state, check OS send URG|ACK, EUT respond with RST. 9. In <CLOSED> state, check OS send RST|ACK, and EUT respond with Nothing. 10. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.2	0x554f2d12, 0xfa71, 0x48eb, 0x96, 0x02, 0xff, 0x5c, 0xfb, 0x8d, 0x45, 0xe6	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly send out the reset segment while in <ESTABLISHED> state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate RST generation in <ESTABLISHED> state. 5. Instruct OS send out un-acceptable ACK, and expect receive ACK which indicate the expected next sequence number. 6. Verify <EUT> send out ACK, and the recvd ACK.ack_id indicating correct seq_id. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.3	0x12dea7e9, 0x1773, 0x4adb, 0x97, 0x27, 0xe8, 0xc3, 0xcf, 0xfb, 0xb9, 0x7b	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly send out the reset segment while in <CLOSE-WAIT> state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Change the state from ESTABLISEHD to CLOSE_WAIT, and call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it. 5. Verify <EUT> send out ACK, and the recvd ACK.ack_id indicating correct seq_id. Then send RST to disconnect the session 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.4	0xebf00938, 0xb335, 0x4a33, 0xa2, 0x7b, 0x4d, 0x54, 0xf6, 0x42, 0x72, 0x99	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly send out the reset segment while in <LAST-ACK> state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Instruct EUT enter LAST_ACK state: OS --> EUT: FIN EUT --> OS: ACK EUT --> OS: FIN Call **EFI_TCP4_PROTOCOL.Close()** to do a graceful close working flow. 5. Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it has enter LAST_WAIT state. 6. Verify whether EUT correctly send out RST in LAST_ACK state. 7. Verify does connection remains in the same states after received any unacceptable segment. 8. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.5	0x21941c4e, 0xb4e3, 0x422b, 0x81, 0x58, 0xef, 0xcd, 0x28, 0xb0, 0xee, 0xef	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly send out the reset segment while in <FIN_WAIT_1> state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Instruct EUT from ESTABLISHED to LAST_ACK state: Call **EFI_TCP4_PROTOCOL.Close()** interface to do a graceful close working flow. Then call **EFI_TCP4_PROTOCOL.GetModeData()** to validate enter FIN_WAIT_1 state. 5. Verify whether EUT correctly send out RST in FIN_WAIT_1 state. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.6	0xee1c295d, 0x13e1, 0x4bc3, 0x94, 0x4b, 0xb5, 0x2e, 0xaf, 0x48, 0xb2, 0x5f	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly send out the reset segment while in <FIN_WAIT_2> state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Instruct EUT from ESTABLISHED to LAST_ACK state: Call **EFI_TCP4_PROTOCOL.Close()** interface to do a graceful close working flow, then OS --> EUT: ACK. Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate enter FIN_WAIT_1 state. 5. Verify whether EUT correctly send out RST in FIN_WAIT_1 state. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.7	0x4fac9b90, 0xf3c4, 0x4779, 0xab, 0x3f, 0x32, 0xe8, 0xd9, 0x9b, 0x8b, 0x09	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly send out the reset segment while in <CLOSING> state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Instruct EUT enter LAST_ACK state: (1) EUT --> OS: FIN Call **EFI_TCP4_PROTOCOL.Close()** to do a graceful close working flow. (2) OS --> EUT: FIN (3) EUT --> OS: ACK (4) Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it is in CLOSING state. 5. Verify whether EUT correctly send out RST in CLOSING state. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.8	0xfa9a7729, 0xc10b, 0x4233, 0xb8, 0xe9, 0xeb, 0x8a, 0xf6, 0x65, 0x85, 0x75	`EFI_TCP4_PROTOCOL` – Tests that the <EUT> correctly send out the reset segment while in <TIME_WAIT> state.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to open a new connection for the new instance. Then handle the three-way handshake. 4. Instruct EUT enter LAST_ACK state: (1) EUT --> OS: FIN Call `EFI_TCP4_PROTOCOL.Close()` to do a graceful close working flow. (2) EUT --> OS: FIN (3) OS --> EUT: FIN|ACK (4) Call `EFI_TCP4_PROTOCOL.GetModeData()` to validate it is in TIME_WAIT state. 5. Verify whether EUT correctly send out RST in TIME_WAIT state. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.9	0xd6646a78, 0x5508, 0x4643, 0x9d, 0x9b, 0x0c, 0xca, 0x22, 0x22, 0x0a, 0xc6	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly send out the empty Acknowledge segment after received data segment with unacceptable Acknowledge.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configured the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to open a new connection for the new instance. Then handle the three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it is in <ESTABLISHED> state. 5. Validate RST generation in <ESTABLISHED> state. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.17.10	0xc0b6a498, 0x1cbd, 0x4df0, 0x97, 0x71, 0xd1, 0x95, 0x14, 0xec, 0x74, 0xf2	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly handles the reception of a RST segment in LISTEN state - <EUT> should ignore the reset segment and remain in LISTEN state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it is in LISTEN state. 4. Instruct <OS> send a RST segment, and expect behavior: no response from EUT. 5. Instruct <OS> send a SYN segment, and receive SYN|ACK from Ack. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.11	0xe48e5518, 0xaf29, 0x4e2b, 0xb9, 0xba, 0xfe, 0xfc, 0x0a, 0x37, 0x19, 0x56	`EFI_TCP4_PROTOCOL` – Tests that the <EUT> correctly handles the reception of a RST segment in SYN_RCVD state - Previous state is LISTEN and it returns to LISTEN state	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as a passive one. 3. Instruct <OS> send a SYN segment, and expect behavior: receive SYN|ACK. Then receive the packet. 4. Instruct <OS> send a valid RST segment, Call `EFI_TCP4_PROTOCOL.GetModeData()` to validate it is in LISTEN state. 5. Re-initialize the connection, and let it enter SYN_RCVD state. 6. Instruct <OS> send a SYN segment, and expect behavior: receive SYN|ACK. Then receive the packet. 7. Instruct <OS> send a valid RST segment, Call `EFI_TCP4_PROTOCOL.GetModeData()` to validate it is in LISTEN state. 8. Call `EFI_TCP4_PROTOCOL.GetModeData()` to validate it is in LISTEN state. 9. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.13	0x386fc38f, 0x8f4d, 0x4c34, 0x85, 0x68, 0x62, 0x71, 0x51, 0x0c, 0x35, 0xf5	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly handles the reception of a RST segment in SYN_SENT state - return to CLOSED state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to initialize connection. 4. <EUT> --> <OS>: SYN, then call **EFI_TCP4_PROTOCOL.GetModeCall()** to validate it is in SYN_SENT state. 5. Instruct <OS> send a valid RST segment, and its sequence number is one-byte less than window boundary. Expect that on receiving a valid RST, the connection returned to CLOSED state. 6. OS --> EUT: SYNC, and expect receive RST, which indicates that EUT is CLOSED state. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.14	0xb886e8c2, 0xf6e7, 0x40e3, 0xbf, 0xc8, 0x78, 0xc3, 0x91, 0x91, 0x8d, 0xae	`EFI_TCP4_PROTOCOL` – Tests that the <EUT> correctly handles the reception of a RST segment in ESTABLISHED state - return to CLOSED state.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to initialize connection. 4. <EUT> --> <OS>: SYN <OS> --> <EUT>: SYN|ACK <EUT> --> <OS>: ACK Call `EFI_TCP4_PROTOCOL.GetModeData()` to validate it is in ESTABLISHED state. 5. Instruct <OS> send a valid RST segment, and its sequence number is one-byte less than window boundary. Expect that on receiving a valid RST, the connection returned to CLOSED state. 6. OS --> EUT: SYNC, and expect receive RST, which indicates that EUT is CLOSED state. 7. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.15	0x1a49bc31, 0xad75, 0x4165, 0xaf, 0xff, 0xae, 0xf0, 0x1d, 0x1a, 0x7b, 0x29	**EFI_TCP4_PROTOCOL –** Tests that the <EUT> correctly handles the reception of a RST segment in FIN_WAIT_1 state - return to CLOSED state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to initialize connection. 4. <EUT> --> <OS>: SYN <OS> --> <EUT>: SYN|ACK <EUT> --> <OS>: ACK Call **EFI_TCP4_PROTOCOL.Close()** to make EUT enter FIN_WAIT_1. <EUT> --> <OS>: FIN Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it is in FIN_WAIT_1 state. 5. Instruct <OS> send a valid RST segment, and its sequence number is at window boundary. Expect that on receiving a valid RST, the connection returned to CLOSED state. 6. OS --> EUT: SYNC, and expect receive RST, which indicates that EUT is CLOSED state. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.16	0xe88fa39a, 0xfbc5, 0x4366, 0x9c, 0x68, 0x48, 0x99, 0x78, 0xd4, 0x0e, 0x23	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly handles the reception of a RST segment in FIN_WAIT_2 state - return to CLOSED state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to initialize connection. 4. <EUT> --> <OS>: SYN <OS> --> <EUT>: SYN|ACK <EUT> --> <OS>: ACK Call **EFI_TCP4_PROTOCOL.Close()** to make EUT enter FIN_WAIT_1. <EUT> --> <OS>: FIN <OS> --> <EUT>: ACK Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it is in FIN_WAIT_2 state. 5. Instruct <OS> send a valid RST segment, and its sequence number is what is expected. Expect that on receiving a valid RST, the connection returned to CLOSED state. 6. OS --> EUT: SYNC, and expect receive RST, which indicates that EUT is CLOSED state. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.17	0x600a697d, 0x6250, 0x49a2, 0x97, 0xac, 0xa3, 0xc7, 0x28, 0x20, 0x3f, 0x9d	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly validate the rcvd RST segment while in SYN_SENT state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to initialize connection. 4. <EUT> --> <OS>: SYN, then call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it is in SYN_SENT state. 5. Instruct <OS> send a invalid RST segment, and RST.ack doesn't ack the SYN. Then call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it is still in SYNC_SENT state. 6. OS --> EUT: SYNC EUT --> OS: SYNC_ACK EUT --> OS: RST, and validate the RST.seq be equal to received ACK. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.17.18	0xa9631841, 0x2e5e, 0x49cb, 0xb9, 0xeb, 0x9a, 0xba, 0x04, 0xaf, 0xa3, 0x5f	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly validate the rcvd RST segment while in LISTEN state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it is in LISTEN state. 4. Instruct <OS> send a invalid RST segment, RST.Seq not in the window. In addition, expect that no packet send out from EUT. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.19	0x4226ee2f, 0xd8f2, 0x46e2, 0x8f, 0xaf, 0x1a, 0x00, 0x42, 0xf6, 0x7e, 0x29	EFI_TCP4_PROTOCOL – Tests that the <EUT> correctly validate the rcvd RST segment while in LISTEN state.	
5.25.1.17.20	0xdf8dc924, 0xa0a4, 0x4520, 0x9d, 0x07, 0x59, 0xae, 0x21, 0x8b, 0xb4, 0x53	EFI_TCP4_PROTOCOL – Tests that the <EUT> correctly validate the rcvd RST segment while in ESTABLISHED state.	1. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 2. Call EFI_TCP4_PROTOCOL.Configure() to configure the new instance as an active one. 3. Call EFI_TCP4_PROTOCOL.Connect() to initialize connection. Then Handle three-way handshake. Call EFI_TCP4_PROTOCOL.GetModeData() to validate it is in ESTABLISHED state. 4. Instruct <OS> send a invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection will still in ESTABLISHED state. 5. OS --> EUT: SYNC, and expect: EUT --> OS: SYNC_ACK 6. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.21	0x17f9536e, 0xa472, 0x4b33, 0x9e, 0x2c, 0x30, 0xb1, 0x8d, 0x82, 0x49, 0x44	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly validate the rcvd RST segment while in FIN_WAIT_1 state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to initialize connection. Then Handle three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Close()** to make EUT enter FIN_WAIT_1. Then call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it. 5. Instruct <OS> send a invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in FIN_WAIT_1 state. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.22	0xe99b76fc, 0x1f57, 0x4f68, 0x8b, 0x16, 0x4e, 0xf8, 0x1a, 0xa7, 0xc6, 0x01	`EFI_TCP4_PROTOCOL` – Tests that the <EUT> correctly validate the rcvd RST segment while in FIN_WAIT_2 state.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` to initialize connection. Then Handle three-way handshake. 4. `Call` `EFI_TCP4_PROTOCOL.Close()` to make EUT enter FIN_WAIT_1. Then OS --> EUT: ACK, and call `EFI_TCP4_PROTOCOL.GetModeData()` to validate it enter in FIN_WAIT_2 state. 5. Instruct <OS> send a invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in FIN_WAIT_2 state. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.23	0xc7f281cf, 0x5ff7, 0x475e, 0xab, 0x0e, 0x8e, 0x13, 0x76, 0xb4, 0x46, 0xa6	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly validate the rcvd RST segment while in CLOSE_WAIT state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to initialize connection. Then Handle three-way handshake. 4. <OS> --> <EUT>: FIN <EUT> --> <OS>: ACK Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate it is in CLOSE_WAIT state. 5. Instruct <OS> send a invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in CLOSE_WAIT state. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.24	0xeea6dd88, 0x1df4, 0x438e, 0xa5, 0x2b, 0xee, 0x9f, 0xc5, 0xb2, 0xd6, 0xf7	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly validate the rcvd RST segment while in CLOSEING state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to initialize connection. Then Handle three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Close()** to close the connection; <EUT> --> <OS>: FIN; <OS> --> <EUT>: FIN. Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate enter CLOSEING state. 5. Instruct <OS> send an invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in CLOSEING state. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.25	0xb316e0cc, 0x260e, 0x4d24, 0xa5, 0xee, 0xf4, 0xae, 0x34, 0x30, 0xa9, 0x52	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly validate the rcvd RST segment while in TIME_WAIT state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to initialize connection. Then Handle three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Close()** to close the connection; <EUT> --> <OS>: FIN; <OS> --> <EUT>: FIN. <EUT> --> <OS>: ACK; <OS> --> <EUT>: ACK; Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate enter TIME_WAIT state. 5. Instruct <OS> send an invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in TIME_WAIT state. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.17.26	0x9a8293c3, 0x3d43, 0x4cfd, 0xb3, 0x73, 0xb1, 0xca, 0x0d, 0xef, 0x91, 0x66	**EFI_TCP4_PROTOCOL** – Tests that the <EUT> correctly validate the rcvd RST segment while in LAST_LACK state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** to initialize connection. Then Handle three-way handshake. 4. <OS> --> <EUT>: FIN; <EUT> --> <OS>: ACK; Call **EFI_TCP4_PROTOCOL.Close()** to close the connection; <EUT> --> <OS>: FIN Call **EFI_TCP4_PROTOCOL.GetModeData()** to validate enter LAST_LACK state. 5. Instruct <OS> send an invalid RST segment, RST.ack doesn't ack the SYN. In addition, the connection is still in LAST_LACK state. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.14 WinFlowCtrl

Number	GUID	Assertion	Test Description
5.25.1.18.1	0xe107339e, 0xed3b, 0x44fa, 0xa9, 0x18, 0x83, 0xf0, 0x10, 0x0e, 0x70, 0x14	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly receives the segment that has the advertised receive window open right-edge and close left-edge.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance. Then handle three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet. 5. After OS got the transmitted data packet, configure the [OS] to send back ACK segment to acknowledge the first segment and keep the advertised window to be 1536 octets. 6. Configure the [OS] to finish the data interaction with [EUT]. 7. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.18.2	0x823c66d7, 0x2787, 0x400d, 0x8f, 0x62, 0x69, 0xdd, 0x3b, 0x21, 0x1f, 0x58	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly receives the segment that has the advertised receive window open right-edge and keep left-edge.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance. Then handle three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet with the length 3072. 5. After OS got the transmitted data packet, configure the [OS] to send back ACK segment to acknowledge the first segment and change the advertised window to be 1024 octets. 6. Acknowledge the SYN segment sent from the [EUT] and change the advertised window to be 1536 octets. 7. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.18.3	0x530d5e6d, 0x928e, 0x42c3, 0xa4, 0x6e, 0x74, 0x93, 0xc0, 0xac, 0xca, 0xbf	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly receives the segment that has the advertised receive window open right-edge and include the duplicated ACKs.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance. Then handle three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet with the length 5120. 5. After OS got the transmitted data packet, configure the [OS] to send back an ACK segment to acknowledge the SYN segment sent by the [EUT]. 6. Change the advertised window to be 2048 octets and capture the responded segments. 7. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.18.4	0x1a697687, 0x3deb, 0x4b7b, 0x89, 0x6f, 0x78, 0x35, 0x95, 0x1b, 0x7a, 0xe9	**`EFI_TCP4_PROTOCOL`** – Tests that the [EUT] correctly transmits the advertised window size of data.	1. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Tcp4 child. 2. Call **`EFI_TCP4_PROTOCOL.Configure()`** to configure the new instance as an active one. 3. Call **`EFI_TCP4_PROTOCOL.Connect()`** for the new instance. Then handle three-way handshake. 4. Call **`EFI_TCP4_PROTOCOL.Transmit()`** to transmit a packet with the length 1024. 5. After OS got the transmitted data packet, configure the [OS] to send back ACK segment to acknowledge the first segment and keep the advertised window to be 2048 octets. 6. Configure the [OS] to finish the data interaction with [EUT]. 7. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.18.5	0xbc12abb0, 0xf022, 0x4705, 0x9d, 0x12, 0x32, 0x78, 0xaa, 0x89, 0x80, 0xb8	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles the bulk data flow, the [EUT] should not respond with an acknowledgement segment for each of the received segments.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance. Then handle three-way handshake. 4. Make the [OS] send ten full-sized and consecutive segments and capture the responded segments. The [EUT] should not respond with an acknowledgement segment for each of the received segments. There should be an acknowledgement segment for at least every second segment. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.18.6	0x0541c800, 0x7639, 0x46f5, 0x90, 0x1a, 0x20, 0x7c, 0xc3, 0x11, 0x44, 0xc9	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly handles a link partner's shrinking window with right-edge shrinking and left-edge closing - test Right Edge Shrinks with Left Edge Closes.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance. Then handle three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet with the length 1024. 5. After OS got the transmitted data packet, configure the [OS] to send back an ACK segment to acknowledge the data segments and change the advertised window to be 1024 octets and capture the responded segments. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.18.7	0x613c599e, 0x26e8, 0x4d39, 0x96, 0xe1, 0x2d, 0x30, 0xd6, 0xbe, 0x20, 0xf7	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles a link partner's shrinking window with right-edge shrinking and left-edge closing - test Right Edge Shrinks with Left Edge Keeps.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance. Then handle three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet with the length 5120. 5. After OS got the transmitted data packet, configure the [OS] to send back an ACK segment to acknowledge the SYN sent by the [EUT] and change the advertised window to be 2048 octets and capture the responded segments. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.18.8	0xbbb555fc, 0x8a4d, 0x41eb, 0xaf, 0x1d, 0x8c, 0xc9, 0x87, 0xb4, 0x46, 0x45	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles a link partner's shrinking window with right-edge shrinking and left-edge closing - test Right Edge Shrinks with Duplicated ACK.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance. Then handle three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet with the length 5120. 5. After OS got the transmitted data packet, configure the [OS] to send back an ACK segment to acknowledge the SYN sent by the [EUT] and change the advertised window to be 2048 octets and capture the responded segments. In addition, window update segment including duplicated ACKs should be discarded 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.18.9	0x5b42c4d0, 0xaf0c, 0x4ae9, 0x9f, 0xfc, 0xb4, 0xf8, 0x3f, 0xcd, 0x4d, 0x73	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly handles a link partner's shrinking window when the data retransmission happens.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance. Then handle three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet with the length 5120. 5. When capturing the retransmitted A segment, configure the [OS] to send back ACK segments and capture the responded segments separately. The ACK is to acknowledge the A segment and change the advertised window to be 1536 octets. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.18.10	0xf3a8f990, 0x0f1f, 0x408f, 0xad, 0x66, 0x2c, 0x98, 0x1f, 0xc1, 0x65, 0x34	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly receives data segments while its partner's advertised window is 0.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance. Then handle three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet with the length 5120. 5. After OS got the transmitted data packet, make the [OS] send an acknowledge segment with a 0 window. Then validate EUT send out the ACK segment correctly. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.18.11	0xce6f5d62, 0x0c72, 0x412d, 0x9a, 0xf4, 0xc8, 0xcc, 0x96, 0x8d, 0xe3, 0x42	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly probes a partner's advertised 0 window.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance. Then handle three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet with the length 5120. 5. After OS got the transmitted data packet, make the [OS] send an acknowledge segment with a 0 window, and in current implementation, the 0 window probing segment contains no data. Then validate EUT send out the ACK segment correctly. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.18.12	0x0165a4f8, 0x5976, 0x4051, 0xa2, 0x73, 0x9e, 0xa1, 0x62, 0xe5, 0xc9, 0xac	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly probes a partner's advertised 0 window, when partner advertises non-0 window, EUT can send out left data segments correctly.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open. 4. Configure the [OS] to send tcp segment with different length payloads. Then validate EUT process and respond correctly. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.15 Options

Number	GUID	Assertion	Test Description
5.25.1.19.1	0x1f1c574b, 0xd5b8, 0x4111, 0x90, 0x14, 0xf6, 0x50, 0x04, 0x3c, 0x8a, 0x71	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly ignores the unsupported options as long as the option has a valid length field.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open. 4. Configure the [OS] to send different unsupported options' tcp segments. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.2	0x5be584cc, 0x39e0, 0x4bcf, 0xaf, 0x69, 0xda, 0x64, 0xff, 0xfa, 0x9a, 0x02	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles End-of-Options option.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open. 4. Configure the [OS] to send tcp segment with CombinedOptions containing End-of-Options option. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.3	0xbfc4a76f, 0x19ad, 0x4f34, 0x97, 0x51, 0x07, 0xd3, 0xd5, 0xe4, 0x92, 0x0a	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles End-of-Options option. There are more options behind the End-of-Options option. These options should be ignored.	1. Build combined options field as No-Option No-Option No-Option End-of-Options Option MSS10-Option (this option should be ignored). 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.4	0xbc3c725e, 0x8784, 0x4559, 0x81, 0x91, 0x60, 0x66, 0x93, 0xb0, 0x9a, 0xd1	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles No-Operation option, segment with the No-Operation option between multiple options but not coinciding with the word boundary.	1. Build combined options field as the No-Operation option between multiple options but not coinciding with the word boundary. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Then check OS get the transmitted data packet. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.5	0x957bd7ef, 0x6a40, 0x46e2, 0xbd, 0x62, 0x9b, 0xa2, 0x39, 0x35, 0xe2, 0x96	EFI_TCP4_PROTOCOL – Tests that the [EUT] correctly handles No-Operation option, segment with the No-Operation option between multiple options at the word boundary.	1. Build combined options field as the No-Operation option between multiple options at the word boundary. 2. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 3. Call EFI_TCP4_PROTOCOL.Configure() to configure the new instance as a passive one. 4. Call EFI_TCP4_PROTOCOL.Accept() for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open. 5. Call EFI_TCP4_PROTOCOL.Transmit() to transmit a packet. Then check OS get the transmitted data packet. 6. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.6	0xd4f6ab22, 0x5d0a, 0x4f9e, 0xa5, 0xce, 0x80, 0x94, 0xf2, 0x42, 0xc2, 0xd0	EFI_TCP4_PROTOCOL – Tests that the [EUT] correctly handles No-Operation option, segment with the No-Operation option between multiple options but not coinciding with the word boundary. one item of the same option is split in different words.	1. Build combined options field as the No-Operation option between multiple options at the word boundary, one item of the same option is split in different words. 2. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 3. Call EFI_TCP4_PROTOCOL.Configure() to configure the new instance as a passive one. 4. Call EFI_TCP4_PROTOCOL.Accept() for the new instance. Then handle three-way handshake. Make [EUT] enter ESTABLISHED state through passive connection open. 5. Call EFI_TCP4_PROTOCOL.Transmit() to transmit a packet. Then check OS get the transmitted data packet. 6. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.7	0xee9c7ea4, 0x3bec, 0x4de0, 0x84, 0x65, 0xcb, 0x18, 0x21, 0x4e, 0x3b, 0x01	**EFI_TCP4_PROTOCOL** – Tests that the functionality - Tests that the [EUT] correctly transmits MSS option in <SYN> segment.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance. 4. Handle three-way handshake and check EUT send out SYN segment with MSS correctly. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.8	0xd69abe03, 0xdbb5, 0x473f, 0x91, 0x59, 0xf3, 0x43, 0xe7, 0xf0, 0x04, 0xe8	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly receives MSS option in <SYN> segment, and then replies to transmit MSS option in <SYN, ACK> segment correctly.	1. Build TCP segment with MSS OPTION, here MSS = 256. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. 5. Handle three-way handshake and Check the *Token.Status* to verify the Accept connection has been completed. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.9	0x98e61624, 0x7c30, 0x4d11, 0x8b, 0xf0, 0x45, 0x4e, 0xd8, 0x0b, 0x21, 0xc0	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly adheres to the MSS of the connection. [EUT] will automatically divide up transmitting data segment if its size is larger than [OS] announced MSS value.	1. Build TCP segment with MSS OPTION, here MSS = 100. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 4. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance. Then handle three-way handshake. 5. [OS] send SYN & ACK segment with MSS option and receive ACK segment. 6. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token.Status* to verify the data has been transmitted successfully. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.10	0x50efbcf2, 0xabe6, 0x4cfa, 0x94, 0xc7, 0x78, 0x86, 0xe9, 0x38, 0xd8, 0x59	**EFI_TCP4_PROTOCOL** – Tests that when [EUT] received <SYN> segment without MSS option, [EUT] could take [OS]'s MSS as RFC default value 536.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Then handle three-way handshake. 4. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token.Status* to verify the data has been transmitted successfully. 5. [OS] sends data to [EUT]: Create a data segment to be transmitted, with size larger than RFC_TCP_DEF_MSS. 6. Call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet. Check the *Token.Status* to verify the data has been transmitted successfully. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.11	0xab7715ef, 0x8d1f, 0x4b68, 0xb8, 0x66, 0xb3, 0x8d, 0x84, 0x71, 0x98, 0x71	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly transmit and receive the MSS option in segments without the SYN flag set high.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. 4. Handle three-way handshake. Send segment with another MSS in non-SYN segment. The [EUT] should ignore the MSS option in non-SYN segments. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token.Status* to verify the data has been transmitted successfully. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.12	0xa0845af3, 0x382f, 0x4ab9, 0x8d, 0xe0, 0xe6, 0xc3, 0x0c, 0xcd, 0x95, 0xd0	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handle the reception of MSS option with invalid option value. Let MSS = 0. Value 0 should be ignored and replaced with 64 (**EFI_TCP_MIN_MSS**).	1. Build TCP MSS option, MSS = 0, invalid value. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 4. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance.Handle three-way handshake. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token.Status* to verify the data has been transmitted successfully. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.13	0xa7d40772, 0xc53a, 0x44f6, 0x98, 0x1e, 0xbf, 0x9f, 0xa6, 0xcf, 0x56, 0x5b	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handle the reception of MSS option with invalid option value. Let MSS > 1460. [EUT] should ignore MSS larger than 1460 and replace it with 1460.	1. Build TCP MSS option, MSS = 2048, invalid value, larger than 1460(Maximum MSS). 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 4. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance.Handle three-way handshake. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token.Status* to verify the data has been transmitted successfully. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.14	0x1b50447f, 0x868c, 0x4ea4, 0x93, 0xc0, 0xcb, 0x00, 0x73, 0x31, 0x52, 0xcf	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handle the reception of segments with unaligned MSS option. Format 1.	1. Create unaligned MSS option with format 1. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 4. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance. Handle three-way handshake. 5. [OS] send SYN & ACK segment with MSS option. Then call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token*.**Status** to verify the data has been transmitted successfully. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.15	0x3973bbb2, 0xe1c5, 0x40ea, 0x8e, 0x50, 0xdb, 0x53, 0x8e, 0xc1, 0x42, 0xa9	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly handle the reception of segments with unaligned MSS option. Format 2.	1. Create unaligned MSS option with format 2. 2. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 3. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 4. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance. Handle three-way handshake. 5. [OS] send SYN & ACK segment with MSS option. Then call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet. Check the *Token*`.Status` to verify the data has been transmitted successfully. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.16	0xb74382c6, 0x37dc, 0x4151, 0x9d, 0xe3, 0xd4, 0x98, 0x8e, 0x4c, 0xd8, 0xcd	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly handle the reception of segments with unaligned MSS option. Format 3.	1. Create unaligned MSS option with format 3. 2. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 3. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 4. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance. Handle three-way handshake. 5. [OS] send SYN & ACK segment with MSS option. Then call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet. Check the *Token*`.Status` to verify the data has been transmitted successfully. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.17	0x53cd1a49, 0xaa07, 0x4bf8, 0x95, 0x45, 0xa4, 0xd3, 0x83, 0x6c, 0x4f, 0xb4	**EFI_TCP4_PROTOCOL** – Tests that when [EUT] received <SYN> segment without MSS option, [EUT] could take [OS]'s MSS as RFC default value 536. With unaligned window scale option as format 2.	1. Create TCP option. Windows Scale: shift.cnt = 2. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Handle three-way handshake. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token.Status* to verify the data has been transmitted successfully. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.18	0x454d5884, 0xf7e1, 0x43a8, 0x97, 0xab, 0x48, 0xbb, 0xd2, 0x22, 0xa6, 0x5b	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly turns window scale option on.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Handle three-way handshake. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token.Status* to verify the data has been transmitted successfully. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.19	0xc8d0492a, 0x79e8, 0x411c, 0x91, 0x42, 0x08, 0x2e, 0x7a, 0x81, 0xbb, 0x86	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly ignores a Window scale option in a segment without SYN bit set.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. Then create another TCP option with another Windows Scale Value which will be sent in <ACK> Segment. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Handle three-way handshake. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token.Status* to verify the data has been transmitted successfully. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.20	0x691e1119, 0xe737, 0x4560, 0x96, 0x33, 0xb7, 0x57, 0xd6, 0x2e, 0x22, 0xde	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly interacts with the partner that doesn't support window scaling option.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance. Then handle three-way handshake. 4. OS send DATA & ACK segment, then call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet. In addition, check the *Token.Status* to verify the data has been transmitted successfully. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.21	0xca16dc5d, 0x5720, 0x45d0, 0xa2, 0xe4, 0x19, 0x98, 0xc2, 0xa8, 0x5f, 0x5c	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly handles the segment with window scaling shift count exceeding 14.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. Calculate [OS] MAX acceptable window. . In addition, set window scale with 16. 2. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 3. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as a passive one. 4. Call `EFI_TCP4_PROTOCOL.Accept()` for the new instance. Handle three-way handshake. 5. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet. Check the *`Token.Status`* to verify the data has been transmitted successfully. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.22	0xade14e0f, 0xa957, 0x4489, 0x83, 0xf8, 0xdb, 0x9f, 0x69, 0x1d, 0xfc, 0x18	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly handles the reception of segments with unaligned window scale option. Format 1.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. In addition, set window scale with 2. 2. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 3. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as a passive one. 4. Call `EFI_TCP4_PROTOCOL.Accept()` for the new instance. Handle three-way handshake. 5. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet. Check the *`Token.Status`* to verify the data has been transmitted successfully. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.23	0x90cc4928, 0xd470, 0x491d, 0xaf, 0xa8, 0x9d, 0x86, 0x07, 0xb7, 0xf3, 0x15	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles the reception of segments with unaligned window scale option. Format 2.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. In addition, set window scale with 2. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Handle three-way handshake. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token.Status* to verify the data has been transmitted successfully. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.24	0x5cd402e2, 0xe9d1, 0x40a7, 0x8a, 0xad, 0xe1, 0xc7, 0x89, 0x42, 0x52, 0x6b	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles the reception of segments with unaligned window scale option. Format 3.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. In addition, set window scale with 2. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Handle three-way handshake. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token.Status* to verify the data has been transmitted successfully. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.25	0xe47378c6, 0x77d8, 0x4f08, 0xbb, 0x52, 0xe7, 0x6b, 0x14, 0xd9, 0x28, 0xd6	**EFI_TCP4_PROTOCOL** – test when [OS]'s scaled window size larger than [OS]'s MSS, here, (256<<2) > 800, [EUT] could correctly send segment data with length small than MSS. With unaligned window scale option as format 2.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. In addition, set window scale with 2. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Handle three-way handshake. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Check the *Token*.**Status** to verify the data has been transmitted successfully. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.26	0x82aacaa9, 0xa48e, 0x47c2, 0xb8, 0xa8, 0x88, 0xd3, 0x18, 0xf1, 0xd4, 0xe1	**EFI_TCP4_PROTOCOL** – test TCP could disable timestamp option, when received <SYN> segment without timestamp while received data segment contain it.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. Timestamps: TSval = 0, TSecr = 0. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Handle three-way handshake. 5. OS send DATA & ACK segment, then call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet. 6. OS get the ACK segment and check the *Token*.**Status** to verify the data has been transmitted successfully. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.27	0xb0bf1171, 0x5e75, 0x42c4, 0x96, 0xed, 0x97, 0x21, 0xc6, 0x50, 0xe6, 0x87	**EFI_TCP4_PROTOCOL** – test TCP could disable timestamp option, when it receives <SYN, ACK> segment without timestamp option.	1. Build TCP Segment with MSS OPTION, MSS = 100. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 4. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance. Handle three-way handshake. 5. [OS] send SYN & ACK segment with MSS option and receive ACK segment. 6. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. Then check the *Token.Status* to verify the data has been transmitted successfully. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.28	0x6db78216, 0x1741, 0x4d22, 0x86, 0x2b, 0x1e, 0x37, 0x6f, 0x9f, 0xbe, 0xc9	**EFI_TCP4_PROTOCOL** – test TCP could correctly recognize and deal with the timestamp option when it is used in TCP option.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. Timestamps: TSval = 0, TSecr = 0. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Handle three-way handshake. 5. OS send DATA & ACK segment, then call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet. 6. OS get the ACK segment and check the *Token.Status* to verify the data has been transmitted successfully. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.29	0x688adc05, 0x942e, 0x4150, 0xa1, 0x6f, 0xec, 0xce, 0x9c, 0x3b, 0x66, 0x52	EFI_TCP4_PROTOCOL – Tests that the [EUT] correctly handles the reception of segments with unaligned Timestamp option. Format 1.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. Timestamps: TSval = 0, TSecr = 0. 2. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 3. Call EFI_TCP4_PROTOCOL.Configure() to configure the new instance as a passive one. 4. Call EFI_TCP4_PROTOCOL.Accept() for the new instance. Handle three-way handshake. 5. OS send DATA & ACK segment, then call EFI_TCP4_PROTOCOL.Receive() to receive a packet. 6. OS get the ACK segment and check the *Token.Status* to verify the data has been transmitted successfully. 7. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.
5.25.1.19.30	0x98e5cf1f, 0x72ce, 0x4be6, 0x99, 0x95, 0x05, 0x43, 0xcd, 0x6c, 0x82, 0x93	EFI_TCP4_PROTOCOL – Tests that the [EUT] correctly handles the reception of segments with unaligned Timestamp option. Format 2.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. Timestamps: TSval = 0, TSecr = 0. 2. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp4 child. 3. Call EFI_TCP4_PROTOCOL.Configure() to configure the new instance as a passive one. 4. Call EFI_TCP4_PROTOCOL.Accept() for the new instance. Handle three-way handshake. 5. OS send DATA & ACK segment, then call EFI_TCP4_PROTOCOL.Receive() to receive a packet. 6. OS get the ACK segment and check the *Token.Status* to verify the data has been transmitted successfully. 7. Call EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.19.31	0x2f71233b, 0xeeaf, 0x4dc5, 0xb3, 0xdd, 0x35, 0x9f, 0xd6, 0xa6, 0xa2, 0x42	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] correctly handles the reception of segments with unaligned Timestamp option. Format 3.	1. Create TCP option. MSS = L_MSS, Windows Scale: shift.cnt = 2. Timestamps: TSval = 0, TSecr = 0. 2. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Handle three-way handshake. 5. OS send DATA & ACK segment, then call **EFI_TCP4_PROTOCOL.Receive()** to receive a packet. 6. OS get the ACK segment and check the *Token.Status* to verify the data has been transmitted successfully. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.16 Others

Number	GUID	Assertion	Test Description
5.25.1.20.1	0xe78b5efa, 0xb455, 0x464e, 0xa2, 0x5f, 0xda, 0xf5, 0x3a, 0x14, 0x2c, 0x09	`EFI_TCP4_PROTOCOL` –Tests that the [EUT] can correctly handle SYN flood. [EUT] should NOT send out <RST> segment to reset incomplete connection queue when ConnectionTimeout (SYN time) haven't reached.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as a passive one. 3. Call `EFI_TCP4_PROTOCOL.Accept()` for the new instance. 4. Send <SYN> flood, and wait to SYN timeout (ConncetionTimeout), then [EUT] send out <RST> segment to reset the incomplete connection. 5. Handles the three-way handshake. OS gets the <SYN, ACK> segment and then sends <ACK> segment. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.20.2	0x0c2a1607, 0xdff9, 0x4794, 0xb8, 0xca, 0x04, 0x28, 0x6a, 0xdf, 0xa8, 0x46	`EFI_TCP4_PROTOCOL` –Tests that the [EUT] can correctly handle SYN flood. [EUT] accepts one or more connection request, thus making MaxSynBacklog NOT full. Accept following incoming <SYN> segment when MaxSynBacklog is NOT full.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as a passive one. 3. Call `EFI_TCP4_PROTOCOL.Accept()` for the new instance. 4. Send <SYN> flood. 5. Handles the three-way handshake. OS gets the <SYN, ACK> segment and then sends <ACK> segment. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.20.3	0xb8b111f9, 0xb3b7, 0x496b, 0x82, 0x5d, 0xaa, 0x9a, 0xd8, 0x59, 0x6c, 0x6e	**EFI_TCP4_PROTOCOL** –Tests that the [EUT] can correctly handle SYN flood. [EUT] should NOT send out <RST> segment to reset incomplete connection queue when ConnectionTimeout (SYN time) haven't reached. Discard following incoming <SYN> segment when MaxSynBacklog is full.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. 4. Send <SYN> flood, and send <SYN> segment to [EUT] when MaxSynBacklog is full. 5. Handles the three-way handshake. OS gets the <SYN, ACK> segment. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.20.4	0x111f5b8e, 0xf762, 0x4eaf, 0x93, 0xb9, 0xe0, 0x97, 0xcb, 0x5b, 0xcd, 0x3f	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] can correctly handle attack-Self consume attack.	1. Initialization of TCB related on OS side. Make the protocol address the same as [EUT], in order to attack. 2.Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. 5. Send <SYN> flood. 6. Handles the three-way handshake. OS gets the <SYN, ACK> segment and sends <ACK> segment 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.20.5	0x8d7dd35a, 0x05f1, 0x495d, 0x8e, 0xed, 0x7e, 0x54, 0x70, 0x20, 0xd7, 0x67	**EFI_TCP4_PROTOCOL** – Tests that the [EUT] can correctly handle attack-Self consume attack with SYN flood.	1. Initialization of TCB related on OS side. Make the protocol address the same as [EUT], in order to attack. 2.Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 3. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 4. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. 5. Handles the three-way handshake. OS gets the <SYN, ACK> segment and sends <ACK> segment 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.20.6	0xef277abd, 0xfe01, 0x4bbb, 0x91, 0x0d, 0xaa, 0xbb, 0x9f, 0x64, 0x68, 0xf4	**EFI_TCP4_PROTOCOL** – Tests that the functionality-Configure OS to send junky data after <FIN,ACK> segment, EUT should reset the connection.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as an active one. 3. Call **EFI_TCP4_PROTOCOL.Connect()** for the new instance. Then handles the three-way handshake. 5. OS sends <FIN, ACK> segment and receives <ACK> segment. 6. OS sends DATA & ACK segment and then receives <RST, ACK> segment. 7. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.20.7	0xa1c11437, 0xbe91, 0x4857, 0x9e, 0xbc, 0x99, 0xfc, 0x3a, 0x3f, 0xba, 0x98	**EFI_TCP4_PROTOCOL** – Tests that the functionality-In CLOSE_WAIT state, configure OS to send FIN to EUT. This FIN should not be duplicated of the last FIN segment.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Then handles the three-way handshake. 5. OS gets <SYN, ACK> segment and sends <ACK> segment. Then check the *Token.Status* to verify the **EFI_TCP4_PROTOCOL.Accept()** has completed. 6. OS sends <FIN, ACK> segment and receives <ACK> segment. 7. Calling **EFI_TCP4_PROTOCOL.GetModeData()**, now EUT is in CLOSE_WAIT state. 8. OS sends <FIN, ACK> segment and receives <ACK> segment. 9. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.20.8	0xeb18fb2d, 0x2306, 0x41bc, 0x9d, 0x68, 0x10, 0x87, 0x8f, 0xf3, 0xe5, 0xef	**EFI_TCP4_PROTOCOL** – Tests that the functionality-In LAST_ACK state, configure OS to send FIN to EUT. This FIN should not be duplicated of the last FIN segment.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Then handles the three-way handshake. 5. OS gets <SYN, ACK> segment and sends <ACK> segment. Then check the *Token.Status* to verify the **EFI_TCP4_PROTOCOL.Accept()** has completed. 6. OS sends <FIN, ACK> segment and receives <ACK> segment. 7. Calling **EFI_TCP4_PROTOCOL.GetModeData()**, now EUT is in CLOSE_WAIT state. 8. Call **EFI_TCP4_PROTOCOL.Close()** to do a graceful close working flow. Then call **EFI_TCP4_PROTOCOL.GetModeData()**, now EUT in LAST_ACK state. 9. OS sends <FIN, ACK> segment and receives <ACK> segment. 10. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.20.9	0x968f5b4d, 0x4801, 0x487f, 0x81, 0xc1, 0xa6, 0x16, 0x91, 0x44, 0x47, 0x72	`EFI_TCP4_PROTOCOL` – Tests that the functionality-In TIME_WAIT state, configure OS to send FIN to EUT. This FIN should not be duplicated of the last FIN segment.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as a passive one. 3. Call `EFI_TCP4_PROTOCOL.Accept()` for the new instance. Then handles the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Close()` to do a graceful close working flow. Then call `EFI_TCP4_PROTOCOL.GetModeData()`, now EUT in FIN_WAIT_1 state. 5. OS sends <FIN, ACK> segment and receives <ACK> segment. Calling `EFI_TCP4_PROTOCOL.GetModeData()`, and now EUT is in TIME_WAIT state. Then OS sends <FIN> segment. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.20.10	0x127d1f26, 0x9f39, 0x435c, 0x80, 0x34, 0x6b, 0x1c, 0xc9, 0x5e, 0x85, 0x3b	`EFI_TCP4_PROTOCOL` – Tests that the functionality-Configure EUT to send data in no-ESTABLISHED state.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as a passive one. 3. Call `EFI_TCP4_PROTOCOL.Accept()` for the new instance. Then handles the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Transmit()` to transmit a packet, without connection established. 5. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.1.20.11	0x2d2065ef, 0x7e6a, 0x419a, 0x84, 0x30, 0x2c, 0x1d, 0xbf, 0xf7, 0x0c, 0xac	**EFI_TCP4_PROTOCOL** – Tests that the functionality-Configure EUT to send data in CLOSE_WAIT state.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Then handles the three-way handshake. In addition, check the *Token.Status* to verify the **EFI_TCP4_PROTOCOL.Accept()** has completed. 4. OS sends <FIN, ACK> segment and receives <ACK> segment. Then call **EFI_TCP4_PROTOCOL.GetModeData()**, now EUT is in CLOSE_WAIT state. 5. Call **EFI_TCP4_PROTOCOL.Transmit()** to transmit a packet. 6. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.17 KeepAliveTimer

Number	GUID	Assertion	Test Description
5.25.1.21.1	0xece1fc13, 0x84f5, 0x413a, 0x90, 0xcb, 0x53, 0xfd, 0x45, 0x3a, 0x8d, 0x07	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly responds to the keep-alive segment which without garbage data.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as a passive one. 3. Call `EFI_TCP4_PROTOCOL.Accept()` for the new instance. Then handles the three-way handshake. 4. Check [EUT] correctly responds to the keep-alive segment which without one garbage data. 5. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.21.2	0x54e62a42, 0x25bb, 0x45c6, 0x90, 0x42, 0x93, 0x96, 0x8d, 0xab, 0xfc, 0x2c	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly responds to the keep-alive segment which with garbage data.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as a passive one. 3. Call `EFI_TCP4_PROTOCOL.Accept()` for the new instance. Then handles the three-way handshake. 4. Check [EUT] keeps connection when not all keep-alive probes were acknowledged. 5. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

21.1.18 RetransmissiomTimer

Number	GUID	Assertion	Test Description
5.25.1.22.1	0x64785c77, 0x4352, 0x4da5, 0xb0, 0xe8, 0x85, 0x0d, 0xdc, 0x5f, 0x32, 0x48	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly retransmit with the method of exponential back off.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance. Then handles the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Trasmit()` to make [EUT] send segment to [OS]. 5. Call `EFI_TCP4_PROTOCOL.Transmit()` to check [EUT] correctly retransmit. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.
5.25.1.22.2	0xf2474612, 0x61e6, 0x4bb9, 0x85, 0x7c, 0xb7, 0x00, 0x97, 0x05, 0x00, 0xdf	`EFI_TCP4_PROTOCOL` – Tests that the [EUT] correctly close connection when retransmission timer time out.	1. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Tcp4 child. 2. Call `EFI_TCP4_PROTOCOL.Configure()` to configure the new instance as an active one. 3. Call `EFI_TCP4_PROTOCOL.Connect()` for the new instance. Then handles the three-way handshake. 4. Call `EFI_TCP4_PROTOCOL.Trasmit()` to make [EUT] send segment to [OS]. 5. Check [EUT] correctly performs retransmission timer. 6. Call `EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Tcp4 child and clean up the environment.

21.1.19 HrdFormatACK

Number	GUID	Assertion	Test Description
5.25.1.23.1	0xb550f0a9, 0x302a, 0x445a, 0x9b, 0xbf, 0xdb, 0xd3, 0x93, 0x9a, 0xec, 0x79	**EFI_TCP4_PROTOCOL** –Tests that the [EUT] correctly generates the ACK numbers, and properly roll over the numbers.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Then handles the three-way handshake. Check the *Token.Status* to verify the connection has been established. 4. Send Segment with seq 4294967294 to see EUTS whether return rollover ack. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.
5.25.1.23.2	0xced29cf0, 0xbfa9, 0x4b92, 0xb9, 0xe9, 0xdc, 0x3e, 0xc9, 0xea, 0x6a, 0x53	**EFI_TCP4_PROTOCOL** –Tests that the [EUT] correctly generates the ACK numbers, and properly roll over the numbers.	1. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Tcp4 child. 2. Call **EFI_TCP4_PROTOCOL.Configure()** to configure the new instance as a passive one. 3. Call **EFI_TCP4_PROTOCOL.Accept()** for the new instance. Then handles the three-way handshake. Check the *Token*.**Status** to verify the connection has been established. 4. Send Segment with seq 4294967294 to see EUTS whether return rollover ack. 5. Call **EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Tcp4 child and clean up the environment.

21.1.20 HrdFormatCheckSum

Number	GUID	Assertion	Test Description
5.25.1.24.1	0xeb8958d6, 0x9fac, 0x4c35, 0xa1, 0x66, 0xf2, 0x35, 0x1f, 0x43, 0x61, 0xb7	**`EFI_TCP4_PROTOCOL`** –Test the [EUT]'s capability on generating a correct checksum field and discarding segments with invalid checksum.	1. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Tcp4 child. 2. Call **`EFI_TCP4_PROTOCOL.Configure()`** to configure the new instance as a passive one. 3. Call **`EFI_TCP4_PROTOCOL.Accept()`** for the new instance. Then handles the three-way handshake. Check the *`Token.Status`* to verify the connection has been established. 4. Send Segment with error CheckSum to see if EUTS discard this packet. 5. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Tcp4 child and clean up the environment.

21.1.21 PersistTimer

Number	GUID	Assertion	Test Description
5.25.1.25.1	0xb498bbfe, 0xd47e, 0x4c9e, 0xb9, 0x80, 0x8f, 0x83, 0xc7, 0x33, 0xc6, 0x26	**`EFI_TCP4_PROTOCOL`** – Tests that the [EUT] correctly performs persist timer with the method of exponential back off.	1. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Tcp4 child. 2. Call **`EFI_TCP4_PROTOCOL.Configure()`** to configure the new instance as an active one. 3. Call **`EFI_TCP4_PROTOCOL.Connect()`** for the new instance. Then handles the three-way handshake. 4. Call **`EFI_TCP4_PROTOCOL.Trasmit()`** to make [EUT] send segment to [OS]. 5. Call **`EFI_TCP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Tcp4 child and clean up the environment.

21.2 EFI_IP4_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_IP4_PROTOCOL Section.

21.2.1 GetModeData()

Number	GUID	Assertion	Test Description
5.25.2.1.1	0xac92ef07, 0xd325, 0x4e3a, 0xad, 0x81, 0x46, 0x46, 0x3c, 0xb4, 0x0f, 0xa8	EFI_IP4_PROTOCOL .GetModeData() - invokes GetModeData() to get all mode data when the Ip4 child has not been configured.	1. Call EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Ip4 child. 2. Call EFI_IP4_PROTOCOL.GetModeData() to get all mode data when the Ip4 child has not been configured. The return status should be EFI_SUCCESS. 3. Call EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment.
5.25.2.1.2	0x5abf337a, 0xfb74, 0x4812, 0x8c, 0xa3, 0x95, 0xb8, 0xbb, 0xed, 0x0b, 0xac	EFI_IP4_PROTOCOL .GetModeData() - invokes GetModeData() to get Ip4 mode data when the IP4 child has not been configured.	1. Call EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Ip4 child. 2. Call EFI_IP4_PROTOCOL.GetModeData() to get Ip4 mode data when the Ip4 child has not been configured. The return status should be EFI_SUCCESS. 3. Call EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment.
5.25.2.1.3	0x459937fd, 0x462d, 0x4b1f, 0x85, 0x78, 0x01, 0x78, 0xac, 0xcf, 0x2a, 0x2e	EFI_IP4_PROTOCOL .GetModeData() - invokes GetModeData() to get Mnp mode data when the IP4 child has not been configured.	1. Call EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Ip4 child. 2. Call EFI_IP4_PROTOCOL.GetModeData() to get Mnp mode data when the Ip4 child has not been configured. The return status should be EFI_SUCCESS. 3. Call EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.1.4	0x96463508, 0xc867, 0x410d, 0xab, 0x41, 0xc4, 0x3b, 0x54, 0x46, 0xe2, 0x53	**EFI_IP4_PROTOCOL .GetModeData()** - invokes **GetModeData()** to get Snp mode data when the IP4 child has not been configured.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.GetModeData()** to get Snp mode data when the Ip4 child has not been configured. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.1.5	0x1b1253d6, 0xfb71, 0x4672, 0x84, 0xfa, 0xb4, 0x0a, 0x20, 0xb1, 0xc0, 0xae	**EFI_IP4_PROTOCOL .GetModeData()** - invokes **GetModeData()** to get all mode data when the IP4 child has been configured.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.GetModeData()** to get all mode data when the Ip4 child has been configured. The return status should be **EFI_SUCCESS**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.1.6	0xa27e3c75, 0xf51a, 0x4c22, 0x8c, 0x64, 0xb4, 0x52, 0xb9, 0xc6, 0xd6, 0xc6	**EFI_IP4_PROTOCOL .GetModeData()** - invokes **GetModeData()** to get Ip4 mode data when the IP4 child has been configured.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.GetModeData()** to get Ip4 mode data when the Ip4 child has been configured. The return status should be **EFI_SUCCESS**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.1.7	0x0fa93b62, 0x3d3b, 0x40df, 0x8d, 0xea, 0x3f, 0x1e, 0x8e, 0xa2, 0x82, 0x1a	`EFI_IP4_PROTOCOL.GetModeData()` - invokes `GetModeData()` to get Mnp mode data when the IP4 child has been configured.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. 3. Call `EFI_IP4_PROTOCOL.GetModeData()` to get Mnp mode data when the Ip4 child has been configured. The return status should be `EFI_SUCCESS`. 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.
5.25.2.1.8	0xefce9133, 0x49e6, 0x426c, 0x92, 0x38, 0x2a, 0x09, 0xda, 0x74, 0x30, 0x2d	`EFI_IP4_PROTOCOL.GetModeData()` - invokes `GetModeData()` to get Snp mode data when the IP4 child has been configured.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. 3. Call `EFI_IP4_PROTOCOL.GetModeData()` to get Snp mode data when the Ip4 child has been configured. The return status should be `EFI_SUCCESS`. 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.1.9	0x6cbce077, 0x33b8, 0x4a73, 0x9e, 0x5a, 0x03, 0x41, 0xa9, 0xee, 0x44, 0xd4	**EFI_IP4_PROTOCOL.GetModeData()** - invokes **GetModeData()** to get all mode data and check the *IcmpTypeList* data item.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.GetModeData()** to get all mode data when the Ip4 child has been configured. The return status should be **EFI_SUCCESS**. Then check the *IcmpTypeCount* and *IcmpTypeList* data item. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.1.10	0x1fb8e582, 0x98c9, 0x461a, 0xbf, 0x26, 0xaf, 0x34, 0x6b, 0x1d, 0x23, 0xe0	**EFI_IP4_PROTOCOL.GetModeData()** - invokes **GetModeData()** to get all mode data and check the *RouteTable* data item.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.GetModeData()** to get all mode data when the Ip4 child has been configured. The return status should be **EFI_SUCCESS**. Then check the *RouteCount* and *RouteTable* data item. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.1.11	0x4f38bf49, 0x2be4, 0x489c, 0xac, 0xb9, 0x70, 0x3e, 0xb1, 0xe3, 0x5b, 0x3b	**EFI_IP4_PROTOCOL .GetModeData()** - invokes **GetModeData()** to get all mode data and check the *GroupTable* data item.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Groups()** to add a group address. 3. Call **EFI_IP4_PROTOCOL.GetModeData()** to get all mode data when the Ip4 child has been configured. The return status should be **EFI_SUCCESS**. Then check the *GroupCount* and *GroupTable* data item. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.1.12	0x3e8d5ff2, 0x5bec, 0x4e2d, 0xa6, 0x60, 0xe8, 0xfb, 0xe9, 0x8f, 0xb8, 0x49	**EFI_IP4_PROTOCOL .GetModeData()** - invokes **GetModeData()** to check the instance status when **Configure()** has been called with an *Ip4ModeData* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.GetModeData()** and then check the *IsStarted* and *IsConfigured* item in *Ip4ModeData*. 4. Call **EFI_IP4_PROTOCOL.Configure()** with an *Ip4ModeData* value of **NULL**. 5. Call **EFI_IP4_PROTOCOL.GetModeData()** . The return status should be **EFI_SUCCESS**. Then check the *IsStarted* and *IsConfigured* item in *Ip4ModeData* 6. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

21.2.2 Configure()

Number	GUID	Assertion	Test Description
5.25.2.2.1	0xf2e2bfe9, 0xe95d, 0x4c25, 0xa7, 0x0a, 0x59, 0x9c, 0xb7, 0x22, 0xcb, 0xde	**EFI_IP4_PROTOCOL .Configure()** - invokes **Configure()** with an *StationAddress* value of not an unicast IPv4 address.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance with an *StationAddress* value of not an unicast IPv4 address. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.2.2	0x1c90fd78, 0x789d, 0x4710, 0x9b, 0x12, 0x27, 0xea, 0x09, 0xee, 0x99, 0x8b	**EFI_IP4_PROTOCOL .Configure()** - invokes **Configure()** with an *SubnetMask* value of an invalid IPv4 subnet mask.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance with an *SubnetMask* value of an invalid IPv4 address. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.2.3	0x85e8e030, 0xf54a, 0x464c, 0x8e, 0xc7, 0xc8, 0xfb, 0x8f, 0x1a, 0x9b, 0xd1	`EFI_IP4_PROTOCOL.Configure()` - invokes `Configure()` to change the *StationAddress* when the instance has been configured before.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. The return status should be `EFI_SUCCESS`. 3. Call `EFI_IP4_PROTOCOL.Configure()` again when the *StationAddress* has been changed. The return status should be `EFI_ALREADY_STARTED`. 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.
5.25.2.2.4	0x62f11c24, 0xe8ff, 0x4687, 0x80, 0x3f, 0x40, 0x3f, 0x0f, 0x87, 0x0c, 0x8b	`EFI_IP4_PROTOCOL.Configure()` - invokes `Configure()` to change the *SubnetMask* when the instance has been configured before.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. The return status should be `EFI_SUCCESS`. 3. Call `EFI_IP4_PROTOCOL.Configure()` again when the *SubnetMask* has been changed. The return status should be `EFI_ALREADY_STARTED`. 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.2.5	0xdddcb20e, 0x00a4, 0x4001, 0x85, 0x08, 0x60, 0x77, 0x3c, 0xfa, 0xba, 0xb8	EFI_IP4_PROTOCOL.Configure() - invokes Configure() and call Transmit() and Receive() to check its function.	1. Call EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Ip4 child. 2. Call EFI_IP4_PROTOCOL.Configure() to configure the new instance. The return status should be EFI_SUCCESS. 3. Call EFI_IP4_PROTOCOL.Transmit() to transmit a packet and check it is successful. 4. Call EFI_IP4_PROTOCOL.Receive() to receive the packet and check it is successful. 5. Call EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment.
5.25.2.2.6	0xdf081df1, 0x845a, 0x4ffe, 0x9a, 0xa3, 0x78, 0xc3, 0x77, 0xa1, 0x35, 0xc0	EFI_IP4_PROTOCOL.Configure() - invokes Configure() and call Receive() to receive a packet.	1. Call EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Ip4 child. 2. Call EFI_IP4_PROTOCOL.Configure() to configure the new instance. The return status should be EFI_SUCCESS. 3. Call EFI_IP4_PROTOCOL.Receive() to receive the packet and check the packet field. 4. Call EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.2.7	0xedcd4582, 0x9349, 0x4f56, 0x9b, 0xac, 0x54, 0xe9, 0x2d, 0x6b, 0x27, 0xb4	**EFI_IP4_PROTOCOL .Configure()** - invokes **Configure()** and call **Receive()** to receive a packet from different *RemoteEther* and *RemoteIp*.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Receive()** to receive the packet from different *RemoteEther* and *RemoteIp*. Then check the packet field. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.2.8	0x90b93642, 0x81b3, 0x4d15, 0x9e, 0xbf, 0xdf, 0xc3, 0xaf, 0x70, 0xe1, 0xc6	**EFI_IP4_PROTOCOL .Configure()** - invokes **Configure()** and call **Transmit()** to transmit a packet.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit the packet and check it is successful. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.2.9	0x171c383a, 0x613b, 0x4d85, 0x9c, 0xd4, 0x85, 0x57, 0x59, 0x4f, 0xb5, 0x67	`EFI_IP4_PROTOCOL.Configure()` - invokes `Configure()` and call `Transmit()` and `Receive()` to check its function after call `Configure()` with an *`IpConfigData`* value of `NULL`.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. The return status should be `EFI_SUCCESS`. 3. Call `EFI_IP4_PROTOCOL.Configure()` with an *`IpConfigData`* value of `NULL`. The return status should be `EFI_SUCCESS`. Then call `EFI_IP4_PROTOCOL.Transmit()` and `EFI_IP4_PROTOCOL.Receive()`, the return status should be `EFI_NOT_STARTED`. 4. Call `EFI_IP4_PROTOCOL.Configure()` to configure the instance again. The return status should be `EFI_SUCCESS`. Then call `EFI_IP4_PROTOCOL.Transmit()` and `EFI_IP4_PROTOCOL.Receive()`, the return status should be `EFI_SUCCESS`. 5. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

21.2.3 Groups()

Number	GUID	Assertion	Test Description
5.25.2.3.1	0x360e7f0a, 0x635d, 0x4660, 0x95, 0x9c, 0x69, 0xa5, 0x39, 0x3c, 0x8d, 0x83	**EFI_IP4_PROTOCOL.Groups()** - invokes **Groups()** with a *JoinFlag* value of *TRUE* and a *GroupAddress* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Groups()** with a *JoinFlag* value of **TRUE** and a *GroupAddress* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.3.2	0x3ac80863, 0x67f2, 0x4554, 0x88, 0x72, 0xcd, 0x92, 0x98, 0xa1, 0xda, 0xac	**EFI_IP4_PROTOCOL.Groups()** - invokes **Groups()** with a *GroupAddress* value other than **NULL** and a **GroupAddress* value of an invalid multicast IPv4 address.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Groups()** with a *GroupAddress* value other than **NULL** and a **GroupAddress* value of an invalid multicast IPv4 address. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.3.3	0x9634a43a, 0x41bc, 0x49f9, 0x80, 0x1c, 0x0e, 0xc1, 0x8b, 0xe1, 0x5c, 0x04	`EFI_IP4_PROTOCOL.Groups()` - invokes `Groups()` to join a group address when it has already in the group table.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. The return status should be `EFI_SUCCESS`. 3. Call `EFI_IP4_PROTOCOL.Groups()` to join a group address into the group table. The return status should be `EFI_SUCCESS`. 4. Call `EFI_IP4_PROTOCOL.Groups()` to join the group address again when it has already joined in step 3. The return status should be `EFI_ALREADY_STARTED`. 5. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.3.4	0x4a2e6bd5, 0x2d4b, 0x4d81, 0xb5, 0x4b, 0x86, 0xc0, 0x03, 0x25, 0x9e, 0xf4	**EFI_IP4_PROTOCOL .Groups()** - invokes **Groups()** to leave a group address which is not in the group table.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Groups()** to join a group address into the group table. The return status should be **EFI_SUCCESS**. 4. Call **EFI_IP4_PROTOCOL.Groups()** to leave the group address joined in step 3. The return status should be **EFI_SUCCESS**. 5. Call **EFI_IP4_PROTOCOL.Groups()** to leave the group address again joined in step 3. The return status should be **EFI_NOT_FOUND**. 6. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.3.5	0x1cc6a89f, 0xf635, 0x4aa6, 0xb2, 0x18, 0xfa, 0xc4, 0x7f, 0x7b, 0x83, 0x7c	**EFI_IP4_PROTOCOL.Groups()** - invokes **Groups()** when the instance has not been started.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Configure()** again with an *IpConfigData* value of **NULL**. 4. Call **EFI_IP4_PROTOCOL.Groups()** with the a *JoinFlag* value of **TRUE** or **FALSE**. The return status should be **EFI_NOT_STARTED**. 5. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.3.6	0x6138d5ae, 0x78b8, 0x43fa, 0x9a, 0x8c, 0x03, 0xb1, 0x87, 0x6d, 0x93, 0x15	**EFI_IP4_PROTOCOL.Groups()** - invokes **Groups()** to join a group address and call **Receive()** to check that it is successful.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Groups()** to join a group address into the group table. The return status should be **EFI_SUCCESS**. 4. Call **EFI_IP4_PROTOCOL.Receice()** to receive a packet from the group IP and check that it is successful. 5. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.3.7	0x340a0020, 0x26ae, 0x4268, 0x87, 0x12, 0xe4, 0x58, 0x2d, 0x3e, 0x36, 0xe7	`EFI_IP4_PROTOCOL.Groups()` - invokes `Groups()` to join two group address and call `Receive()` after leaving a group address from the group table.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. The return status should be `EFI_SUCCESS`. 3. Call `EFI_IP4_PROTOCOL.Groups()` to join two group address into the group table. The return status should be `EFI_SUCCESS`. 4. Call `EFI_IP4_PROTOCOL.Groups()` to leave a group address from the group table. 5. Call `EFI_IP4_PROTOCOL.Receice()` to receive a packet from the group IP and check that it is successful. 6. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.3.8	0x3234871f, 0x9682, 0x4bbd, 0x85, 0x56, 0x4a, 0x17, 0xa9, 0x74, 0xdf, 0xb7	`EFI_IP4_PROTOCOL.Groups()` - invokes `Groups()` to leave all group address and call `Receive()` to check that it is successful.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. The return status should be `EFI_SUCCESS`. 3. Call `EFI_IP4_PROTOCOL.Groups()` to join two group address into the group table. The return status should be `EFI_SUCCESS`. 4. Call `EFI_IP4_PROTOCOL.Receice()` to receive a packet from the group IP and check it can not receive the packet. 5. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

21.2.4 Routes()

Number	GUID	Assertion	Test Description
5.25.2.4.1	0x9fa3288c, 0x1caa, 0x4174, 0xbc, 0x81, 0x84, 0x52, 0x16, 0x6f, 0x09, 0x58	**EFI_IP4_PROTOCOL. Routes()** - invokes **Routes()** with a *DeleteRoute* value of **FALSE** and a *SubnetAddress* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *DeleteRoute* value of FALSE and a *SubnetAddress* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.4.2	0x6ed77fe8, 0xb20a, 0x417c, 0xb7, 0x64, 0x69, 0x36, 0x70, 0x74, 0xdf, 0x49	**EFI_IP4_PROTOCOL. Routes()** - invokes **Routes()** with a *DeleteRoute* value of **FALSE** and a *SubnetMask* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *DeleteRoute* value of FALSE and a *SubnetMask* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.4.3	0x0ca07e01, 0xecf0, 0x4726, 0x8b, 0xb0, 0xb8, 0xd6, 0xde, 0xa2, 0x69, 0x77	**EFI_IP4_PROTOCOL.Routes()** - invokes **Routes()** with a *DeleteRoute* value of FALSE and a *GatewayAddress* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *DeleteRoute* value of **FALSE** and a *GatewayAddress* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.4.4	0xe7ba143d, 0xb80c, 0x411b, 0xa7, 0xf7, 0x60, 0xa2, 0xb5, 0x10, 0xc7, 0x3d	**EFI_IP4_PROTOCOL.Routes()** - invokes **Routes()** with a *DeleteRoute* value of TRUE and a *SubnetAddress* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *DeleteRoute* value of **TRUE** and a *SubnetAddress* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.4.5	0xf66dd341, 0xae38, 0x464e, 0x81, 0x22, 0x7f, 0xcb, 0xa4, 0x99, 0x1d, 0x31	**EFI_IP4_PROTOCOL.Routes()** - invokes **Routes()** with a *DeleteRoute* value of TRUE and a *SubnetMask* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *DeleteRoute* value of TRUE and a *SubnetMask* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.4.6	0x713db4d5, 0x4e17, 0x487a, 0x83, 0x62, 0xe1, 0x18, 0x8b, 0x9f, 0x5e, 0x61	**EFI_IP4_PROTOCOL.Routes()** - invokes **Routes()** with a *DeleteRoute* value of TRUE and a *GatewayAddress* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *DeleteRoute* value of **TRUE** and a *GatewayAddress* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.4.7	0xea35d39b, 0x7350, 0x427c, 0x8c, 0x04, 0x69, 0x0a, 0x75, 0x42, 0x75, 0x70	**EFI_IP4_PROTOCOL. Routes()** - invokes **Routes()** with a **SubnetMask* value of an invalid subnet mask.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a **SubnetMask* value of an invalid subnet mask. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.4.8	0xe02b9e49, 0x3889, 0x4183, 0xac, 0x91, 0xb7, 0x4a, 0x63, 0xb5, 0xcf, 0x8f	**EFI_IP4_PROTOCOL. Routes()** - invokes **Routes()** with a **GatewayAddress* value of an invalid unicast IPv4 address.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a **GatewayAddress* value of an invalid unicast IPv4 address. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.4.9	0x5a3132ea, 0x658e, 0x4bfb, 0xa3, 0xd2, 0x49, 0xeb, 0x6e, 0x88, 0xdf, 0xed	**EFI_IP4_PROTOCOL. Routes()** - invokes **Routes()** when the route has already been defined in the routing table (when *DeleteRoute* is **FALSE**).	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_PROTOCOL.Routes()** to add a route into the routing table. The return status should be **EFI_SUCCESS**. 4. Call **EFI_IP4_PROTOCOL.Routes()** to add the route again when it has already been defined in the routing table. The return status should be **EFI_ACCESS_DENIED**. 5. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.4.10	0x5f228ffc, 0xfc1c, 0x43f6, 0x99, 0x14, 0x26, 0xcd, 0xcb, 0xee, 0x24, 0x97	`EFI_IP4_PROTOCOL.Routes()` - invokes `Routes()` to delete a route which is not in the routing table.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. The return status should be `EFI_SUCCESS`. 3. Call `EFI_IP4_PROTOCOL.Routes()` to add a route into the routing table. The return status should be `EFI_SUCCESS`. 4. Call `EFI_IP4_PROTOCOL.Routes()` to delete the route added in step 3. The return status should be `EFI_SUCCESS`. 5. Call `EFI_IP4_PROTOCOL.Routes()` to delete the route again while it is not in the routing table. The return status should be `EFI_NOT_FOUND`. 6. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.
5.25.2.4.11	0x3c71e7d7, 0xe61e, 0x4973, 0x90, 0xff, 0x36, 0x5b, 0xe5, 0xa7, 0x92, 0xb4	`EFI_IP4_PROTOCOL.Routes()` - invokes `Routes()` to add a route when using the default address and configuration has not finished yet.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. 3. Call `EFI_IP4_PROTOCOL.Routes()` to add a route into the routing table when using the default address and configuration has not finished yet. The return status should be `EFI_NO_MAPPING.` 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.4.12	0xba7d5323, 0x36e4, 0x4b1a, 0x9e, 0x74, 0xdf, 0xe6, 0xd3, 0x30, 0xe5, 0xc5	`EFI_IP4_PROTOCOL.Routes()` - invokes `Routes()` delete a route when using the default address and configuration has not finished yet.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. 3. Call `EFI_IP4_PROTOCOL.Routes()` to delete a route into the routing table when using the default address and configuration has not finished yet. The return status should be `EFI_NO_MAPPING.` 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.
5.25.2.4.19	0xa51618f2, 0xe542, 0x4498, 0x82, 0xab, 0xc9, 0x9d, 0xc8, 0x61, 0x7f, 0xd0	`EFI_IP4_PROTOCOL.Routes()` - Invoke `Routes()` when the driver instance has not been started. The return status should be `EFI_NOT_STARTED`.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Routes()` to add a route into the routing table when the instance has not been started. The return status should be `EFI_NOT_STARTED.` 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.4.13	0xf3239a4b, 0x29c1, 0x461e, 0xbf, 0x54, 0x96, 0x5d, 0xd9, 0x2e, 0x69, 0xb5	**EFI_IP4_PROTOCOL. Routes()** - invokes **Routes()** with a *SubnetAddress* value of "0.0.0.0",a *SubnetMask* value of "0.0.0.0" and a *GatewayAddress* value of "172.16.210.162". Then call **Transmit()** to check it.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *SubnetAddress* value of "0.0.0.0",a *SubnetMask* value of "0.0.0.0" and a *GatewayAddress* value of "172.16.210.162". The return status should be **EFI_SUCCESS**. 4. Call Ip.**Transmit()** to check the packet. 5. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.4.14	0x7b17e47c, 0x0f7c, 0x4351, 0xa8, 0xfa, 0xf6, 0xf5, 0x9d, 0x03, 0x54, 0x93	**EFI_IP4_PROTOCOL. Routes()** - invokes **Routes()** with a *SubnetAddress* value of "172.16.210.0",a *SubnetMask* value of "255.255.255.0" and a *GatewayAddress* value of "0.0.0.0". Then call **Transmit()** to check it.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *SubnetAddress* value of "172.16.210.0",a *SubnetMask* value of "255.255.255.0" and a *GatewayAddress* value of "0.0.0.0". The return status should be **EFI_SUCCESS**. 4. Call **EFI_IP4_PROTOCOL.Transmit()** to check the packet. 5. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.4.15	0x52762945, 0x2148, 0x48c9, 0x82, 0xea, 0xac, 0x78, 0xf3, 0x7c, 0xb7, 0x23	**EFI_IP4_PROTOCOL.Routes()** - invokes **Routes()** with a *SubnetAddress* value of "172.16.220.0", a *SubnetMask* value of "255.255.255.0" and a *GatewayAddress* value of "172.16.210.162". Then call **Transmit()** to check it.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *SubnetAddress* value of "172.16.220.0", a *SubnetMask* value of "255.255.255.0" and a *GatewayAddress* value of "172.16.210.162". The return status should be **EFI_SUCCESS**. 4. Call **EFI_IP4_PROTOCOL.Transmit()** to check the packet. 5. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.4.16	0x91439045, 0x15f1, 0x4a25, 0x83, 0x0e, 0x4d, 0x0a, 0x2b, 0x2c, 0x13, 0x0a	**EFI_IP4_PROTOCOL.Routes()** - invokes **Routes()** to delete the route with a *SubnetAddress* value of "0.0.0.0", a *SubnetMask* value of "0.0.0.0" and a *GatewayAddress* value of "172.16.210.162". Then call **Transmit()** to check it.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *SubnetAddress* value of "0.0.0.0", a *SubnetMask* value of "0.0.0.0" and a *GatewayAddress* value of "172.16.210.162". The return status should be **EFI_SUCCESS**. 4. Call Ip.**Routes()** to delete the route added in step 3. 5. Call **EFI_IP4_PROTOCOL.Transmit()** to check the packet. 6. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.4.17	0x3f884c4d, 0xcfd5, 0x49b8, 0x8f, 0x08, 0xfb, 0xb7, 0xb7, 0x44, 0x1e, 0xed	**EFI_IP4_PROTOCOL. Routes()** - invokes **Routes()** to delete the route with a *SubnetAddress* value of "172.16.210.0", a *SubnetMask* value of "255.255.255.0" and a *GatewayAddress* value of "0.0.0.0". Then call **Transmit()** to check it.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *SubnetAddress* value of "172.16.210.0", a *SubnetMask* value of "255.255.255.0" and a *GatewayAddress* value of "0.0.0.0". The return status should be **EFI_SUCCESS**. 4. Call Ip.**Routes()** to delete the route added in step 3. 5. Call **EFI_IP4_PROTOCOL.Transmit()** to check the packet. 6. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.4.18	0x4745ddac, 0x9429, 0x4159, 0xbc, 0x13, 0x85, 0xf8, 0xd6, 0xe5, 0x23, 0x13	**EFI_IP4_PROTOCOL. Routes()** - invokes **Routes()** to delete the route with a *SubnetAddress* value of "172.16.220.0", a *SubnetMask* value of "255.255.255.0" and a *GatewayAddress* value of "172.16.210.162". Then call **Transmit()** to check it.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Routes()** with a *SubnetAddress* value of "172.16.220.0", a *SubnetMask* value of "255.255.255.0" and a *GatewayAddress* value of "172.16.210.162". The return status should be **EFI_SUCCESS**. 4. Call Ip.**Routes()** to delete the route added in step 3. 5. Call **EFI_IP4_PROTOCOL.Transmit()** to check the packet. 6. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

21.2.5 Transmit()

Number	GUID	Assertion	Test Description
5.25.2.5.1	0x47ba87f8, 0x188e, 0x4b41, 0x8d, 0x53, 0xa9, 0x08, 0x87, 0x73, 0x15, 0x6b	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with a *Token* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with a *Token* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.2	0x5701c82b, 0x64bf, 0x415e, 0x9f, 0x0f, 0x46, 0x23, 0x7b, 0x01, 0x91, 0xdf	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with a *Token.Event* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with a *Token.Event* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.3	0x44454955, 0x744c, 0x4648, 0xab, 0x05, 0x74, 0xac, 0x73, 0x0f, 0x9a, 0xa2	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with a *Token.Packet.TxData* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with a *Token.Packet.TxData* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.4	0xf8e8550e, 0x46ff, 0x4e49, 0x81, 0xe5, 0xf7, 0x06, 0x5a, 0xd4, 0x84, 0xf9	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with a *Token.Packet.OptionsLength* value other than 0 and a *Token*.Packet.OptionsBuffer value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with a *Token.Packet.OptionsLength* value other than 0 and a *Token.Packet.OptionsBuffer* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.5	0x9edbcb93, 0xa28b, 0x40ed, 0x90, 0xfa, 0xa1, 0x7d, 0x41, 0xed, 0x93, 0x7d	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with a *Token.Packet.FragmentCount* value of 0.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with a *Token*.Packet.FragmentCount value of 0. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.6	0x2ff682e3, 0x0b85, 0x4755, 0xaf, 0x58, 0x16, 0x57, 0x81, 0x23, 0x83, 0x2f	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with one or more of the *Token.Packet.TxData.FragmentTable[].FragmentLength* fields is 0.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with one or more of the *Token.Packet.TxData.FragmentTable[].FragmentLength* fields is 0. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.7	0x199e798a, 0x2f1a, 0x49ac, 0x81, 0x05, 0x91, 0xef, 0xc1, 0x24, 0x5b, 0xae	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with one or more of the *Token.Packet.TxData.FragmentTable[].FragmentBuffer* fields is **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with one or more of the *Token.Packet.TxData.FragmentTable[].FragmentBuffer* fields is **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.8	0x9bb3fb85, 0xbdfd, 0x4b0f, 0x95, 0x4c, 0x6a, 0x21, 0xbb, 0xff, 0x93, 0x7f	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with a *Token.Packet.TxData.TotalDataLength* value of 0.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with a *Token.Packet.TxData.TotalDataLength* value of 0. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.9	0xff0221ac, 0x7a1c, 0x40e7, 0xbf, 0xea, 0xb2, 0xde, 0x89, 0xb2, 0xbf, 0x76	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with a *Token.Packet.TxData.TotalDataLength* not equal to the sum of fragment lengths.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with a *Token.Packet.TxData.TotalDataLength* not equal to the sum of fragment lengths. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.10	0xa22a64e0, 0xd98c, 0x49af, 0x98, 0xe1, 0x0d, 0x30, 0x93, 0x29, 0x7d, 0x34	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with a *Token.Packet.TxData.OverrideData.GatewayAddress* in the override data structure value of an invalid unicast IPv4 address if *OverrideData* is not **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with a *Token.Packet.TxData.OverrideData.GatewayAddress* in the override data structure value of an invalid unicast IPv4 address if *OverrideData* is not **NULL**. (Set *SourceAddress* as "172.16.210.101" and *GatewayAddress* "172.16.210.255"). The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.11	0x2b27d386, 0xab2a, 0x4882, 0xa7, 0xf8, 0x71, 0xc0, 0xb6, 0x9c, 0xf9, 0x88	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with a *Token.Packet.TxData.OverrideData.GatewayAddress* in the override data structure value of an invalid unicast IPv4 address if *OverrideData* is not **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with a *Token.Packet.TxData.OverrideData.GatewayAddress* in the override data structure value of an invalid unicast IPv4 address if *OverrideData* is not **NULL**. (Set *SourceAddress* as "172.16.210.101" and *GatewayAddress* "172.16.210.254"). The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.12	0x0251b68d, 0x32fe, 0x4b0e, 0xad, 0xe9, 0xc8, 0x45, 0x71, 0xd4, 0xfe, 0xec	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with a *Token.Packet.TxData.OverrideData.GatewayAddress* in the override data structure value of an invalid unicast IPv4 address if *OverrideData* is not **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with a *Token.Packet.TxData.OverrideData.GatewayAddress* in the override data structure value of an invalid unicast IPv4 address if *OverrideData* is not **NULL**. (Set *SourceAddress* as "172.16.210.101" and *GatewayAddress* "240.0.0.2"). The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.13	0x3e687a19, 0x7b23, 0x45b7, 0x8f, 0x81, 0x0b, 0x1c, 0x28, 0xd5, 0x2a, 0x26	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** with a *Token.Packet.TxData.OverrideData.GatewayAddress* in the override data structure value of an invalid unicast IPv4 address if *OverrideData* is not **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** with a *Token.Packet.TxData.OverrideData.GatewayAddress* in the override data structure value of an invalid unicast IPv4 address if *OverrideData* is not **NULL**. (Set *SourceAddress* as "172.16.210.101" and *GatewayAddress* "255.255.255.255"). The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.14	0x00e45a87, 0xa739, 0x43af, 0xa7, 0x9f, 0x8d, 0xc7, 0xd3, 0x14, 0xab, 0x20	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** when the IP header in *FragmentTable* is not a well-formed header when *RawData* is **TRUE**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** when the IP header in *FragmentTable* is not a well-formed header when *RawData* is **TRUE**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.15	0x4fc5e7c5, 0xdb04, 0x4d15, 0x94, 0xa4, 0x2d, 0xba, 0xac, 0x60, 0xbd, 0xbc	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** when *Token.Packet.TxData.TotalDataLength* is not equal to the sum of fragment lengths.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** when *Token.Packet.TxData.TotalDataLength* is not equal to the sum of fragment lengths.(set *Token.Packet.TxData.TotalDataLength* as 1). The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.16	0x5264d068, 0xe5a1, 0x41eb, 0x9d, 0x1e, 0xf8, 0xff, 0x20, 0x37, 0x77, 0x3a	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** when the length of the IPv4 header + option length + total data length is greater than the maximum packet size and *DoNotFragment* is **TRUE**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** when the length of the IPv4 header + option length + total data length is greater than the maximum packet size and *DoNotFragment* is **TRUE**. The return status should be **EFI_BAD_BUFFER_SIZE**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.17	0x383b9eb0, 0xb83a, 0x447d, 0x85, 0xcc, 0xd5, 0x2d, 0x49, 0xe5, 0x34, 0x8d	`EFI_IP4_PROTOCOL.Transmit()` - invokes `Transmit()` when the length of the IPv4 header + option length + total data length is greater than MTU.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. 3. Call `EFI_IP4_PROTOCOL.Transmit()` when the length of the IPv4 header + option length + total data length is greater than MTU. The return status should be `EFI_BAD_BUFFER_SIZE`. 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.
5.25.2.5.18	0x0ca2174b, 0x3731, 0x469f, 0x98, 0x2f, 0xb3, 0x45, 0xd8, 0xad, 0x7b, 0x4a	`EFI_IP4_PROTOCOL.Transmit()` - invokes `Transmit()` when the transmit completion token with the same *`Token.Event`* was already in the transmit queue.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. 3. Call `EFI_IP4_PROTOCOL.Transmit()` to transmit a packet. 4. Call `EFI_IP4_PROTOCOL.Transmit()` with the same *`Token`* in step 2. The return status should be `EFI_ACCESS_DENIED`. 5. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.31	0x45b5cb36, 0xf07a, 0x493c, 0xac, 0xee, 0x49, 0x91, 0x66, 0x6f, 0x0f, 0x00	`EFI_IP4_PROTOCOL.Transmit()` - invoke `Transmit()` when the length of the IPv4 header + option length + total data length is greater than MTU.The return status shoule be `EFI_BAD_BUFFER_SIZE`.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. 4. Call `EFI_IP4_PROTOCOL.Transmit()` when the length of the IPv4 header + option length + total data length is greater than MTU. The return status shoule be `EFI_BAD_BUFFER_SIZE`. 5. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.
5.25.2.5.19	0x394621bf, 0xe45c, 0x4dc7, 0x8c, 0x59, 0xa4, 0xb6, 0x25, 0xb0, 0x72, 0x4f	`EFI_IP4_PROTOCOL.Transmit()` - invokes `Transmit()` when there is no route found to destination address.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. Configure the *`IpConfigData.StationAddress`* not same as *`TxData.DestinationAddress`*. 3. Call `EFI_IP4_PROTOCOL.Transmit()` when there is no route found to destination address. The return status should be `EFI_NOT_FOUND`. 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.20	0xb0e8dd55, 0x8e92, 0x4d9c, 0xba, 0x2d, 0x95, 0xcf, 0x35, 0x75, 0x71, 0x0b	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** when the instance has not been started.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. Then call **EFI_IP4_PROTOCOL.Configure()** again with a *IpConfigData* value of **NULL.** 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit a packet. The return status should be **EFI_NOT_STARTED**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.32	0x3f38c35e, 0x92b8, 0x4e20, 0xaa, 0x23, 0x4b, 0xd9, 0xf6, 0xb3, 0x57, 0x7a	**EFI_IP4_PROTOCOL.Transmit()** - invoke **Transmit()** when the instance has not been started.The return status should be **EFI_NOT_STARTED**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** when the instance has not been started.The return status should be **EFI_NOT_STARTED**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.21	0xac9ddcc1, 0xa095, 0x474b, 0x84, 0x06, 0x10, 0x37, 0xa4, 0x77, 0xe2, 0x24	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** to transmit an unicast packet.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit an unicast packet. The return status should be **EFI_SUCCESS**. Then check the packet field. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.22	0x3abee622, 0x0543, 0x46c6, 0xad, 0xfa, 0x97, 0x3a, 0x89, 0x6c, 0xbb, 0xdc	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** to transmit a multicast packet.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit a multicast packet. The return status should be **EFI_SUCCESS**. Then check the packet field. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.23	0xcc0ad3d9, 0xf1cd, 0x47e3, 0x81, 0x1d, 0xcb, 0x7a, 0x4e, 0x33, 0xd0, 0xfe	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** to transmit a broadcast packet.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit a broadcast packet. The return status should be **EFI_SUCCESS**. Then check the packet field. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.24	0x0979fc12, 0x53a1, 0x4cfb, 0x8c, 0xd7, 0xdf, 0xef, 0xb2, 0xc3, 0x76, 0x94	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** to transmit a packet using *OverrideData*.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. Set *IpConfigData .StationAddress* "172.16.210.102" and *IpConfigData .SubnetMask* "255.255.255.0". 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit a packet when set *OverrideData.SourceAddress* "172.16.210.101" and *OverrideData.GatewayAddress* "0.0.0.0". The return status should be **EFI_SUCCESS**. Then check the packet field. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.25	0x3b0ae017, 0xcb82, 0x4f94, 0xb3, 0x17, 0xf7, 0x1d, 0x25, 0xe0, 0x33, 0xed	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** to transmit a packet with *TxData.OptionsLength* set as 4.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit a packet with *TxData.OptionsLength* set as 4. The return status should be **EFI_SUCCESS**. Then check the packet field. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.26	0x2e24f6c8, 0x9fbf, 0x4fc3, 0xbb, 0x92, 0x1d, 0xd6, 0xab, 0xfa, 0xbd, 0x6f	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** to transmit a packet with *TxData.OptionsLength* set as 40 and initialize **TxData.OptionsBuffer**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit a packet with *TxData.OptionsLength* set as 40 and initialize *TxData.OptionsBuffer*. The return status should be **EFI_SUCCESS**. Then check the packet field. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.27	0x1da54ed7, 0x24d1, 0x4a19, 0xad, 0x19, 0x43, 0x89, 0x40, 0xd2, 0x73, 0xd2	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** to transmit a packet with *TxData.FragmentCount* set as 4 and *IpConfigData .DoNotFragment* set as **TRUE**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit a packet with *TxData.FragmentCount* set as 4 and *IpConfigData .DoNotFragment* set as **TRUE**. The return status should be **EFI_SUCCESS**. Then check the packet field. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.28	0xbd451149, 0xc815, 0x4454, 0xb5, 0xf1, 0x8e, 0x14, 0x47, 0x6f, 0x91, 0x17	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** to transmit a packet with *TxData.FragmentCount* set as 4 and *IpConfigData .DoNotFragment* set as **FALSE**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit 45 packets with *TxData.FragmentCount* set as 4 and *IpConfigData .DoNotFragment* set as **FALSE**. The return status should be **EFI_SUCCESS**. Then check the captured packets number. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.5.29	0x298bc2eb, 0xa07b, 0x4e66, 0xba, 0xef, 0x2d, 0x03, 0x11, 0x72, 0xd4, 0xcb	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** to transmit a packet with *TxData.DestinationAddress* set as "172.16.210.255" and *FragmentTable.FragmentBuffer* filled with char data.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit a packet with *TxData.DestinationAddress* set as "172.16.210.255" and *FragmentTable.FragmentBuffer* filled with char data. The return status should be **EFI_SUCCESS**. Then check packet field. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.5.30	0x538a9496, 0x49a0, 0x4fe9, 0xa9, 0xe3, 0x0b, 0x20, 0x3f, 0xef, 0x03, 0xbb	**EFI_IP4_PROTOCOL.Transmit()** - invokes **Transmit()** to transmit a packet with *FragmentTable.FragmentBuffer* filled with UNIT8 data and *FragmentTable.FragmentBuffer* initialized.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Transmit()** to transmit a packet when *FragmentTable.FragmentBuffer* filled with UNIT8 data and *FragmentTable.FragmentBuffer* initialized. The return status should be **EFI_SUCCESS**. Then check packet field. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

21.2.6 Receive()

Number	GUID	Assertion	Test Description
5.25.2.6.1	0x31ee7913, 0x8cdf, 0x47dd , 0xa7, 0x29, 0xc9, 0x70, 0x51, 0xfc, 0x25, 0xfe	**EFI_IP4_PROTOCOL.Receive()** - invokes **Receive()** with a *Token* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Receive()** to receive a packet with a *Token* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.6.2	0x2ca314a9, 0x1afe, 0x40a3, 0xa4, 0x91, 0xc3, 0xe7, 0x2b, 0x02, 0x33, 0x7d	**EFI_IP4_PROTOCOL.Receive()** - invokes **Receive()** with a *Token.Event* value of **NULL**.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Receive()** to receive a packet with a *Token.Event* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.6.3	0x4bb1005a, 0x5268, 0x4abf, 0x81, 0x34, 0x6d, 0x37, 0x0c, 0xde, 0x8e, 0x01	**EFI_IP4_PROTOCOL.Receive()** - invokes **Receive()** with the token that has already been placed in the receive queue.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Receive()** to receive a packet. 4. Call **EFI_IP4_PROTOCOL.Receive()** to receive a packet with the same *Token.Event* used in step 3. The return status should be **EFI_ACCESS_DENIED**. 5. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.6.4	0xb9a3d3cd, 0xe982, 0x4268, 0xa7, 0x2a, 0xc3, 0xe5, 0xe8, 0xb6, 0xac, 0xa0	**EFI_IP4_PROTOCOL.Receive()** - invokes **Receive()** when the instance has not been started.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. Then call **EFI_IP4_PROTOCOL.Configure()** again with *Token* is **NULL.** 3. Call **EFI_IP4_PROTOCOL.Receive()** to receive a packet. The return status should be **EFI_NOT_STARTED**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.6.5	0xf9658b87, 0x2377, 0x4fa2, 0xbe, 0x2a, 0x9c, 0x8d, 0x4b, 0x7e, 0xec, 0xe1	**EFI_IP4_PROTOCOL.Receive()** - invokes **Receive()** when an ICMP error packet was received.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Create (from IP head) and send an ICMP error packet, and Call **EFI_IP4_PROTOCOL.Receive()** to receive the packet. The return status should be **EFI_SUCCESS**. Then check the packet field. 6. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.6.6	0x134d695e, 0x6ea0, 0x46df, 0x8d, 0xbb, 0x62, 0x63, 0xf7, 0x1b, 0x29, 0x1a	**EFI_IP4_PROTOCOL.Receive()** - invokes **Receive()** when an ICMP error packet was received.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Create (from IP payload) and send an ICMP error packet, and Call **EFI_IP4_PROTOCOL.Receive()** to receive the packet. The return status should be **EFI_SUCCESS**. Then check the packet field. 6. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.6.7	0x4aed29df, 0x95c0, 0x42b0, 0xaa, 0x65, 0xff, 0x72, 0xf1, 0x6d, 0x22, 0x4a	**EFI_IP4_PROTOCOL.Receive()** - invokes **Receive()** to receive an ip packet.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Create an ip packet and call **EFI_IP4_PROTOCOL.Receive()** to receive the packet. The return status should be **EFI_SUCCESS**. Then check the packet field. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.6.8	0x47cb6918, 0xd454, 0x42f5, 0xa2, 0xab, 0x8e, 0xa5, 0x47, 0x3c, 0x6a, 0xab	**EFI_IP4_PROTOCOL.Receive()** - invokes **Receive()** to receive an ethernet packet.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Create an ethernet packet and call **EFI_IP4_PROTOCOL.Receive()** to receive the packet. The return status should be **EFI_SUCCESS**. Then check the field of the packet. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.6.9	0xb2a56bae, 0x716d, 0x48b1, 0x9e, 0xc0, 0xd6, 0xbe, 0xed, 0xb2, 0x0e, 0xe2	`EFI_IP4_PROTOCOL.Receive()` - invokes `Receive()` to receive 4 ip packets.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. 3. Create 4 ip packets and call `EFI_IP4_PROTOCOL.Receive()` to receive the packets. The return status should be `EFI_SUCCESS`. Then check the packets field and count. 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.
5.25.2.6.10	0x452c7b90, 0xc99f, 0x4106, 0xbe, 0xce, 0x2d, 0xcd, 0x53, 0x50, 0x73, 0xd4	`EFI_IP4_PROTOCOL.Receive()` - invokes `Receive()` to receive 45 ip packets.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. 3. Create 45 ip packets and call `EFI_IP4_PROTOCOL.Receive()` to receive the packets. The return status should be `EFI_SUCCESS`. Then check the packet field and count. 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.6.11	0x5f497c40, 0xa1d3, 0x4223, 0xbc, 0x33, 0x4c, 0x8d, 0x96, 0x7d, 0xfc, 0xf7	**`EFI_IP4_PROTOCOL.Receive()`** - invokes **`Receive()`** to receive a broadcast ip packet.	1. Call **`EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Ip4 child. 2. Call **`EFI_IP4_PROTOCOL.Configure()`** to configure the new instance. Set *`IpConfigData.AcceptBroadcast`* is **`TRUE`**. 3. Create an ip packet and set RemoteEther FF:FF:FF:FF:FF:FF. call **`EFI_IP4_PROTOCOL.Receive()`** to receive the broadcast packet. The return status should be **`EFI_SUCCESS`**. Then check the packet field. 4. Call **`EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Ip4 child and clean up the environment.
5.25.2.6.12	0x4be19438, 0xc5d8, 0x4af4, 0xaf, 0x0f, 0x8e, 0xc7, 0x49, 0x67, 0x2b, 0x40	**`EFI_IP4_PROTOCOL.Receive()`** - invokes **`Receive()`** to receive an unformatted packet by set *RawData* with **`TRUE`**.	1. Call **`EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Ip4 child. 2. Call **`EFI_IP4_PROTOCOL.Configure()`** to configure the new instance. Set *`RawData`* with **`TRUE`**. 3. Create an ip packet and call **`EFI_IP4_PROTOCOL.Receive()`** to receive an unformatted packet. The return status should be **`EFI_SUCCESS`**. 4. Call **`EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Ip4 child and clean up the environment.

21.2.7 Cancel()

Number	GUID	Assertion	Test Description
5.25.2.7.1	0x95d1ac2d, 0x4aaf, 0x4004, 0xb6, 0xa0, 0x8e, 0xec, 0x13, 0xd8, 0x31, 0xcc	`EFI_IP4_PROTOCOL.Cancel()` - invokes `Cancel()` when the asynchronous I/O request was not found in the transmit or receive queue.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. 3. Call `EFI_IP4_PROTOCOL.Cancel()` to abort an asynchronous transmit or receive request. The return status should be `EFI_NOT_FOUND`. 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.
5.25.2.7.2	0xb41eab67, 0xc87c, 0x46a8, 0xae, 0x9d, 0x2c, 0xec, 0x34, 0xf7, 0x6d, 0x38	`EFI_IP4_PROTOCOL.Cancel()` - invokes `Cancel()` with a *Token* value of `NULL` when the instance has not been started.	1. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new Ip4 child. 2. Call `EFI_IP4_PROTOCOL.Configure()` to configure the new instance. Then call `EFI_IP4_PROTOCOL.Configure()` again with *Token* `NULL`. 3. Call `EFI_IP4_PROTOCOL.Cancel()` to abort an asynchronous transmit or receive request with a *Token* value of `NULL`. The return status should be `EFI_NOT_STARTED`. 4. Call `EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.7.3	0x22fa385b, 0xc124, 0x41cd, 0xa6, 0xd9, 0x74, 0xf7, 0xc7, 0x78, 0x10, 0x88	**EFI_IP4_PROTOCOL.Cancel()** - invokes **Cancel()** when the instance has not been started.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. Then call **EFI_IP4_PROTOCOL.Configure()** again with *Token* **NULL**. 3. Call **EFI_IP4_PROTOCOL.Cancel()** to abort an asynchronous transmit or receive request. The return status should be **EFI_NOT_STARTED**. 4. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.
5.25.2.7.4	0xd5bd141b, 0x5ade, 0x4831, 0xaf, 0x3c, 0x15, 0x46, 0xcd, 0xf4, 0xbc, 0x41	**EFI_IP4_PROTOCOL.Cancel()** - invokes **Cancel()** to abort a receive request.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Receive()** to receive a packet. The return status should be **EFI_SUCCESS**. 4. Call **EFI_IP4_PROTOCOL.Cancel()** to abort the asynchronous receive request. The return status should be **EFI_SUCCESS**. Then check the status. 5. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

Number	GUID	Assertion	Test Description
5.25.2.7.5	0xf689d953, 0x1270, 0x448e, 0x93, 0xb1, 0xc0, 0xa5, 0x19, 0x1d, 0x6e, 0x10	**EFI_IP4_PROTOCOL.Cancel()** - invokes **Cancel()** with a *Token* value of **NULL** to abort all receive requests.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_PROTOCOL.Configure()** to configure the new instance. 3. Call **EFI_IP4_PROTOCOL.Receive()** twice to put two receive requests. The return status should be **EFI_SUCCESS**. 4. Call **EFI_IP4_PROTOCOL.Cancel()** with a *Token* value of **NULL** to abort all asynchronous receive requests. The return status should be **EFI_SUCCESS**. Then check the status. 5. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created Ip4 child and clean up the environment.

21.2.8 Poll()

Number	GUID	Assertion	Test Description
5.25.2.8.1	0x1c22cb9a, 0x14c5, 0x41a9, 0xa2, 0x00, 0x9e, 0x89, 0x90, 0xc4, 0x1b, 0xb4	**`EFI_IP4_PROTOCOL.Poll()`** - invokes **Poll()** when the instance has not been started.	1. Call **`EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Ip4 child. 2. Call **`EFI_IP4_PROTOCOL.Configure()`** to configure the new instance**.** Then call **`EFI_IP4_PROTOCOL.Configure()`** again with *`IpConfigData`* **`NULL`.** 3. Call **`EFI_IP4_PROTOCOL.Poll()`** for incoming data packets and processes outgoing data packets. The return status should be **`EFI_NOT_STARTED`**. 4. Call **`EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Ip4 child and clean up the environment.

21.2.9 CreateChild()

Number	GUID	Assertion	Test Description
5.25.2.9.1	0xafda2aee, 0x1e1d, 0x4212, 0x82, 0x0a, 0x49, 0x69, 0x96, 0x8c, 0x26, 0xea	**`EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** - invokes CreateChild() with a *`ChildHandle`* value of **`NULL`**.	1. Call **`EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new Ip4 child with a *`ChildHandle`* value of **`NULL`**. the return status should be **`EFI_INVALID_PARAMETER`**. 2. Call **`EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created Ip4 child and clean up the environment.
5.25.2.9.2	0x110c0779, 0x61f0, 0x46a5, 0x94, 0xd8, 0xe5, 0xf9, 0xfc, 0x24, 0xea, 0xba	**`EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** – invokes **`CreateChild()`** to create several Ip4 childs.	Call **`EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create childs three times and then destroy them.

21.2.10 DestroyChild()

Number	GUID	Assertion	Test Description
5.25.2.10.1	0x7b89cc20, 0x3546, 0x4d7d, 0xae, 0x4b, 0xd7, 0xa6, 0xac, 0x94, 0xe9, 0x6b	**EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** - invokes **DestroyChild()** when the *ChildHandle* does not support the protocol that is removed.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** with the parameter *ChildHandle* that was created just now. the return status should be **EFI_SUCCESS**. 3. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** with the parameter *ChildHandle* that was created just now. the return status should be **EFI_UNSUPPORTED**.
5.25.2.10.2	0x5e6fe618, 0x13a3, 0x4107, 0x8e, 0x1e, 0x35, 0xa8, 0x57, 0x84, 0x47, 0x12	**EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** - invokes DestroyChild() to destroy a **NULL** child.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new Ip4 child. 2. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** with the parameter *ChildHandle* is **NULL**.The return status should be **EFI_INVALID_PARAMETER**.
5.25.2.10.3	0x08e3cc7b, 0x4441, 0x4bf3, 0xac, 0x61, 0xec, 0x2e, 0x63, 0x82, 0xb8, 0x17	**EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** - invokes DestroyChild() to destroy the inexistent child.	1. Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the inexistent child. The return status should be **EFI_INVALID_PARAMETER.**
5.25.2.10.4	0x1400e3f9, 0x9681, 0x4da0, 0xbc, 0x18, 0xde, 0xce, 0xa8, 0x2f, 0x65, 0xf4	**EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** - to test the function of **DestroyChild().**	Call **EFI_IP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly three created Ip4 childs.

21.3 EFI_IP4_CONFIG_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_IP4_CONFIG_PROTOCOL Section.

21.3.1 Start()

Number	GUID	Assertion	Test Description
5.25.3.1.1	0x5e97a936, 0xe3df, 0x4755, 0xa8, 0x33, 0x42, 0x4c, 0xd0, 0xd3, 0x38, 0xda	**EFI_IP4_CONFIG_PROTOCOL.Start()** - invokes *Start()* when the parameter *DoneEvent* is **NULL**.	1. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*. 2. Call **EFI_IP4_CONFIG_PROTOCOL.Start()** to start the configuration process with a *DoneEvent* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*. 4. clean up the environment.
5.25.3.1.2	0xe527172c, 0x26d9, 0x440a, 0x85, 0x4c, 0x15, 0x49, 0xfc, 0x6d, 0x5e, 0x49	**EFI_IP4_CONFIG_PROTOCOL.Start()** - invokes **Start()** when the parameter *ReconfigEvent* is **NULL**.	1. Call **BS.CreateEvent()** to create a new Event for the parameter **DoneEvent**. 2. Call **EFI_IP4_CONFIG_PROTOCOL.Start()** to start the configuration process with a *ReconfigEvent* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 3. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*. 4. clean up the environment.

Number	GUID	Assertion	Test Description
5.25.3.1.3	0xcd185521, 0xd395, 0x4be4, 0xbf, 0x0e, 0x21, 0x42, 0xc7, 0xb5, 0x1c, 0x78	**EFI_IP4_CONFIG_PROTOCOL.Start()** - invokes **Start()** when the configuration policy for the EFI IPv4 Protocol driver has already started.	1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*. 2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*. 3. Call **EFI_IP4_CONFIG_PROTOCOL.Start()** to start the configuration process. The return status should be **EFI_SUCCESS**. 4. Call **EFI_IP4_CONFIG_PROTOCOL.Start()** to start the configuration process again. The return status should be **EFI_ALREADY_STARTED**. 5. Call **EFI_IP4_CONFIG_PROTOCOL.Stop()** to stop the configuration process. 6. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*. 7. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*. 8. clean up the environment.
5.25.3.1.4	0x686babd0, 0x3be4, 0x4be1, 0x9a, 0xed, 0x38, 0x29, 0x83, 0x6a, 0xfc, 0x04	**EFI_IP4_CONFIG_PROTOCOL.Start()** - invokes **Start()** when the parameters *DoneEvent* and *ReconfigEvent* are not **NULL**.	1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*. 2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*. 3. Call **EFI_IP4_CONFIG_PROTOCOL.Start()** to start the configuration process. The return status should be **EFI_SUCCESS**. 4. Call **EFI_IP4_CONFIG_PROTOCOL.Stop()** to stop the configuration process. 5. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*. 6. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*. 7. clean up the environment.

21.3.2 Stop()

Number	GUID	Assertion	Test Description
5.25.3.2.1	0xc5c3a59b, 0x4963, 0x43d5, 0x87, 0xfb, 0xc3, 0x53, 0x4c, 0x94, 0x5b, 0x38	**`EFI_IP4_CONFIG_PROTOCOL.Stop()`** - invokes **`Stop()`** when the configuration process has not been started.	1. Call **`BS.CreateEvent()`** to create a new Event for the parameter *`DoneEvent`*. 2. Call **`BS.CreateEvent()`** to create a new Event for the parameter *`ReconfigEvent`*. 3. Call **`EFI_IP4_CONFIG_PROTOCOL.Start()`** to start the configuration process. 4. Call **`EFI_IP4_CONFIG_PROTOCOL.Stop()`** to stop the configuration process. The return status should be **`EFI_SUCCESS`**. 5. Call **`EFI_IP4_CONFIG_PROTOCOL.Stop()`** to stop the configuration process again. The return status should be **`EFI_NOT_STARTED`**. 6. Call **`BS.CloseEvent()`** to close the Event for the parameter *`DoneEvent`*. 7. Call **`BS.CloseEvent()`** to close the Event for the parameter *`ReconfigEvent`*. 8. clean up the environment.

Number	GUID	Assertion	Test Description
5.25.3.2.2	0x68d111a9, 0x35c6, 0x4e54, 0xaf, 0xae, 0x93, 0xc8, 0xe2, 0x95, 0xad, 0x3b	**`EFI_IP4_CONFIG_PROTOCOL.Stop()`** - invokes **`Stop()`** to verify the configuration process.	1. Call **`BS.CreateEvent()`** to create a new Event for the parameter *`DoneEvent`*. 2. Call **`BS.CreateEvent()`** to create a new Event for the parameter *`ReconfigEvent`*. 3. Call **`EFI_IP4_CONFIG_PROTOCOL.Start()`** to start the configuration process. 4. Call **`EFI_IP4_CONFIG_PROTOCOL.Stop()`** to stop the configuration process. The return status should be **`EFI_SUCCESS`**. 5. Call **`BS.CloseEvent()`** to close the Event for the parameter *`DoneEvent`*. 6. Call **`BS.CloseEvent()`** to close the Event for the parameter *`ReconfigEvent`*. 7. clean up the environment.

21.3.3 GetData()

Number	GUID	Assertion	Test Description
5.25.3.3.1	0xd21e8801, 0x7a1b, 0x4258, 0x84, 0xbe, 0x47, 0x68, 0xc0, 0x25, 0xe7, 0x1b	**EFI_IP4_CONFIG_PROTOCOL.GetData()** - invokes **GetData()** when the configuration policy for the EFI IPv4 Protocol driver is not running.	1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*. 2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*. 3. Call **EFI_IP4_CONFIG_PROTOCOL.Start()** to start the configuration process. 4. Call **EFI_IP4_CONFIG_PROTOCOL.Stop()** to stop the configuration process. 5. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*. 6. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*. 7. Call **EFI_IP4_CONFIG_PROTOCOL.GetData()** to get configuration data when the driver is not running. The return status should be **EFI_NOT_STARTED**. 8. clean up the environment.

Number	GUID	Assertion	Test Description
5.25.3.3.2	0xb1b6d64a, 0xc963, 0x4d93, 0xaa, 0x56, 0xcd, 0xff, 0x2e, 0x09, 0x6a, 0x84	**`EFI_IP4_CONFIG_PROTOCOL.GetData()`** – invokes **`GetData()`** when EFI Ipv4 Protocol driver configuration is still running.	1. Call **`BS.CreateEvent()`** to create a new Event for the parameter *`DoneEvent`*. 2. Call **`BS.CreateEvent()`** to create a new Event for the parameter *`ReconfigEvent`*. 3. Call **`EFI_IP4_CONFIG_PROTOCOL.Start()`** to start the configuration process. 4. Call **`EFI_IP4_CONFIG_PROTOCOL.GetData()`** to get configuration data when the driver is still running. The return status should be **`EFI_NOT_READY`**. 5. Call **`EFI_IP4_CONFIG_PROTOCOL.Stop()`** to stop the configuration process. 6. Call **`BS.CloseEvent()`** to close the Event for the parameter *`DoneEvent`*. 7. Call **`BS.CloseEvent()`** to close the Event for the parameter *`ReconfigEvent`*. 8. clean up the environment.

Number	GUID	Assertion	Test Description
5.25.3.3.3	0x819d1861, 0xf092, 0x4c33, 0xbe, 0xf9, 0x8f, 0xf8, 0x8f, 0x05, 0xb2, 0xb3	**EFI_IP4_CONFIG_PROTOCOL.GetData()** – invokes **GetData()** when the parameter *IpConfigData Size* is smaller than the configuration data buffer.	1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*. 2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*. 3. Call **EFI_IP4_CONFIG_PROTOCOL.Start()** to start the configuration process. 4. Call **EFI_IP4_CONFIG_PROTOCOL.GetData()** to get configuration data with an *IpConfigData Size* value of 0. The return status should be **EFI_BUFFER_TOO_SMALL**. 5. Call **EFI_IP4_CONFIG_PROTOCOL.Stop()** to stop the configuration process. 6. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*. 7. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*. 8. clean up the environment.

Number	GUID	Assertion	Test Description
5.25.3.3.4	0x1257612e, 0xe00c, 0x43d1, 0x97, 0xef, 0xfb, 0x60, 0x00, 0x30, 0x03, 0x1e	**EFI_IP4_CONFIG_PROTOCOL.GetData()** – invokes **GetData()** when the parameter *IpConfigData* is **NULL**.	1. Call **BS.CreateEvent()** to create a new Event for the parameter *DoneEvent*. 2. Call **BS.CreateEvent()** to create a new Event for the parameter *ReconfigEvent*. 3. Call **EFI_IP4_CONFIG_PROTOCOL.Start()** to start the configuration process. 4. Call **EFI_IP4_CONFIG_PROTOCOL.GetData()** to get configuration data with an *IpConfigData* value of **NULL**. The return status should be **EFI_BUFFER_TOO_SMALL**. 5. Call **EFI_IP4_CONFIG_PROTOCOL.Stop()** to stop the configuration process. 6. Call **BS.CloseEvent()** to close the Event for the parameter *DoneEvent*. 7. Call **BS.CloseEvent()** to close the Event for the parameter *ReconfigEvent*. 8. clean up the environment.

Number	GUID	Assertion	Test Description
5.25.3.3.5	0x30710a44, 0x79e9, 0x45fc, 0x97, 0x4e, 0x3f, 0x48, 0x36, 0xbe, 0x33, 0xc8	`EFI_IP4_CONFIG_PROTOCOL.GetData()` – Test the function of `GetData()`.	1. Call `EFI_IP4_CONFIG_PROTOCOL.Stop()` to make sure configuration policy for the EFI IPv4 protocol driver is not running. 2. Call `BS.CreateEvent()` to create a new Event for the parameter *`DoneEvent`*. 3. Call `BS.CreateEvent()` to create a new Event for the parameter *`ReconfigEvent`*. 4. Call `EFI_IP4_CONFIG_PROTOCOL.Start()` to start the configuration process. 5. Send DHCPOFFER packet to agent. 6. Capture and validate DHCPREQUEST packet. 7. Send DHCPACK packet to agent 8. Call `EFI_IP4_CONFIG_PROTOCOL.GetData()` to get configuration data. 9. Call `EFI_IP4_CONFIG_PROTOCOL.Stop()` to stop the configuration process. 10. Call `BS.CloseEvent()` to close the Event for the parameter *`DoneEvent`*. 11. Call `BS.CloseEvent()` to close the Event for the parameter *`ReconfigEvent`*. 12. clean up the environment.

21.4 EFI_TCP6_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_TCP6_PROTOCOL Section.

21.4.1 CreateChild()/DestroyChild()

Number	GUID	Assertion	Test Description
5.25.4.1.1	0xfca64cbc, 0xd99e, 0x42f0, 0x91, 0x23, 0x07, 0x76, 0xd7, 0x71,0x82, 0x9f	EFI_TCP6_PROTOCOL.CreateChild() - CreateChild() returns EFI_INVALID_PARAMETER when ChildHandle is NULL.	Call CreateChild() when ChildHandle is NULL, the return status should be EFI_INVALID_PARAMETER.
5.25.4.1.2	0x991825b0, 0xd208, 0x429b, 0x98, 0xc9, 0x40, 0x46, 0xe5, 0x40, 0x00, 0x15	EFI_TCP6_PROTOCOL.DestroyChild() - DestroyChild() returns EFI_INVALID_PARAMETER with ChildHandle being NULL.	Call DestroyChild() when ChildHandle is NULL, the return status should be EFI_INVALID_PARAMETER.
5.25.4.1.3	0x7bfd1b83, 0x519b, 0x4bb4, 0x9a, 0x44, 0x12, 0x4a, 0xdc, 0x43, 0xdc, 0x56	EFI_TCP6_PROTOCOL.CreateChild() - CreateChild() returns EFI_SUCCESS with valid parameters.	5.25.4.1.3 to 5.25.4.1.6 belong to one case. 1. Call CreateChild() with valid parameters to create child1, The return status should be EFI_SUCCESS.
5.25.4.1.4	0x2d22615b, 0x8e8b, 0x44d2, 0x95, 0x25, 0xcc, 0x5c, 0x7e, 0x8c, 0x84, 0x54	EFI_TCP6_PROTOCOL.CreateChild() - CreateChild() returns EFI_SUCCESS with valid parameters.	2. Call CreateChild() with valid parameters to create child2, The return status should be EFI_SUCCESS.
5.25.4.1.5	0xd681c6b2, 0xa4d4, 0x4725, 0xab, 0xe5, 0xea, 0x5b, 0x03, 0x80, 0x76, 0xbf	EFI_TCP6_PROTOCOL.DestroyChild() - DestroyChild() returns EFI_SUCCESS with valid parameters.	3. Call DestroyChild() with valid parameters to destroy child1, The return status should be EFI_SUCCESS.
5.25.4.1.6	0x363eac60, 0x183a, 0x4b57, 0xae, 0x9e, 0x91, 0xcc, 0xf1, 0x95, 0x39, 0xfd	EFI_TCP6_PROTOCOL.DestroyChild() - DestroyChild() returns EFI_SUCCESS with valid parameters.	4. Call DestroyChild() with valid parameters to destroy child2, The return status should be EFI_SUCCESS.

21.4.2 GetModeData()

Number	GUID	Assertion	Test Description
5.25.4.2.1	0xd957c9de, 0x716a, 0x4f6e, 0xbe, 0x7c, 0x66, 0xc6, 0xe5, 0xa0, 0x2e, 0x09	EFI_TCP6_PROTOCOL.GetModeData() - GetModeData() returns EFI NOT STARTED when the instance is not configured.	Call GetModeData() with valid parameters before the TCP instance is configured., the return status should be EFI_NOT_STARTED.
5.25.4.2.2	0x88a3650b, 0x3aa5, 0x4417, 0x97, 0x71, 0xef, 0xa4, 0xf6, 0xe5, 0x9a, 0x79	EFI_TCP6_PROTOCOL.GetModeData() - GetModeData() returns EFI_SUCCESS with valid parameters.	5.25.4.2.2 to 5.25.4.2.8 belong to one case. 1. Call GetModeData() with all no NULL input parameters, the return status should be EFI_SUCCESS and the configured data should be correct.
5.25.4.2.3	0x798259ad, 0xbc64, 0x4989, 0x9d, 0x8b, 0x82, 0x48, 0x01, 0x1a, 0x03, 0x06	EFI_TCP6_PROTOCOL.GetModeData() - GetModeData() returns EFI_SUCCESS with valid parameters.	2. Call GetModeData() with all NULL input parameters, the return status should be EFI_SUCCESS.
5.25.4.2.4	0xccb9b645, 0xf133, 0x4a2c, 0xbc, 0x72, 0xc1, 0xf1, 0xc8, 0x15, 0x05, 0xe5	EFI_TCP6_PROTOCOL.GetModeData() - GetModeData() returns EFI_SUCCESS with valid parameters.	3. Call GetModeData() when TcpConnectionState is NULL, the return status should be EFI_SUCCESS.
5.25.4.2.5	0xa9389312, 0x0007, 0x48ec, 0xab, 0x83, 0x26, 0x81, 0x1d, 0x0f, 0xa7, 0x97	EFI_TCP6_PROTOCOL.GetModeData() - GetModeData() returns EFI_SUCCESS with valid parameters.	4. Call GetModeData() when TcpConfigData is NULL, the return status should be EFI_SUCCESS.
5.25.4.2.6	0x8aa7bf92, 0xf01f, 0x4de8, 0x80, 0xab, 0x78, 0x9f, 0x4d, 0xaa, 0x16, 0x49	EFI_TCP6_PROTOCOL.GetModeData() - GetModeData() returns EFI_SUCCESS with valid parameters.	5. Call GetModeData() when Ip6ModeData is NULL, the return status should be EFI_SUCCESS.
5.25.4.2.7	0x92fcc066, 0xf41d, 0x4aad, 0xa6, 0x02, 0xf8, 0x4e, 0xde, 0x26, 0x15, 0x6d	EFI_TCP6_PROTOCOL.GetModeData() - GetModeData() returns EFI_SUCCESS with valid parameters.	6. Call GetModeData() when MnpConfigData is NULL, the return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.25.4.2.8	0xb30b7510, 0x3055, 0x427d, 0x85, 0x4a, 0x79, 0xcd, 0xb1, 0xbb, 0xd2, 0x01	`EFI_TCP6_PROTOCOL.GetModeData() - GetModeData()` returns `EFI_SUCCESS` with valid parameters.	7. Call `GetModeData()` when `SnpModeData` is `NULL`, the return status should be `EFI_SUCCESS`.

21.4.3 Configure()

Number	GUID	Assertion	Test Description
5.25.4.3.1	0xbebb71c0, 0xe62e, 0x400d, 0x9e, 0xaf, 0x3e, 0xbf, 0xb0, 0x23, 0xb2, 0xd6	EFI_TCP6_PROTOCOL.Configure() – Configure() returns EFI_INVALID PARAMETERS when the station address is invalid.	Call Configure() when StationAddress is 2000::1(2000::1 is not configured for the testing environment), the return status should be EFI_INVALID_PARAMETER.
5.25.4.3.2	0xabff27d2, 0x86ef, 0x4399, 0xbd, 0x90, 0x57, 0x8e, 0x8e, 0x08, 0x37, 0xb4	EFI_TCP6_PROTOCOL.Configure() – Configure() returns EFI_INVALID PARAMETERS when the remote address is invalid.	Call Configure() when RemoteAddress is ff02::1(link local multicast address, not a valid unicast address), the return status should be EFI_INVALID_PARAMETER.
5.25.4.3.3	0x1f16d3cc, 0x5ccf, 0x4177, 0x8b, 0xf2, 0x56, 0xde, 0x33, 0xe0, 0xd1, 0xf7	EFI_TCP6_PROTOCOL.Configure() – Configure() returns EFI_INVALID PARAMETERS when the remote access point is invalid.	5.25.4.3.3 to 5.25.4.3.4 belong to one case 1. Call Configure() when RemoteAddress is :: and RemotePort is 8888, the return status should be EFI_INVALID_PARAMETER.
5.25.4.3.4	0xae7a2155, 0x192e, 0x4bbb, 0x92, 0xc5, 0xad, 0x6d, 0x17, 0x57, 0xbc, 0xeb	EFI_TCP6_PROTOCOL.Configure() – Configure() returns EFI_INVALID PARAMETERS when the remote access point is invalid.	2. Call Configure() when RemoteAddress is 2002::1 and RemotePort is 0, the return status should be EFI_INVALID_PARAMETER.
5.25.4.3.5	0x3fea1f75, 0xce53, 0x4c85, 0xb8, 0xe5, 0x8e, 0x5a, 0x7c, 0x42, 0xeb, 0x64	EFI_TCP6_PROTOCOL.Configure() – Configure() returns EFI_INVALID PARAMETERS when the access point has already been used by another instance.	1. Create Child1 and call Configure() with valid parameters. 2. Create Child2 and call Configure() with the same access point. The return status should be EFI_INVALID_PARAMETER.
5.25.4.3.6	0xd8bc8edb, 0xfe65, 0x4457, 0xb5, 0x5a, 0xeb, 0xd4, 0xfa, 0xde, 0x7b, 0x7d	EFI_TCP6_PROTOCOL.Configure() – Configure() returns EFI_ACCESS DENIED when updating the configuration without reset.	1. Call Configure() with valid parameters. 2. Call Configure() with valid parameters for the same instance. The return status should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.25.4.3.7	0xad816e3d, 0xf3e6, 0x443b, 0xa1, 0x54, 0x08, 0x51, 0xa5, 0x64, 0x63, 0xb4	EFI_TCP6_PROTOCOL.Configure() – Configure() returns EFI_SUCCESS with valid parameters.	5.25.4.3.7 to 5.25.4.3.8 belong to one case 1. Call Configure() with valid parameters. The return status should be EFI_SUCCESS.
5.25.4.3.8	0x85d67600, 0xf53b, 0x4363, 0x98, 0x34, 0xb9, 0x21, 0xaa, 0xf8, 0x8f, 0x08	The Configure() should correctly set the data as expected.	2. Call GetModeData() and check whether the data is set as expected.
5.25.4.3.9	0x51b04624, 0xaa43, 0x4424, 0xa9, 0xb4, 0xee, 0x2f, 0x26, 0x24, 0xf5, 0x2f	The Tcp instance should enter into Tcp_Listen state after being configured.	5.25.4.3.9 to 5.25.4.3.13 belong to one case 1. Call Configure() with valid parameters. 2. Call GetModeData() to examine whether the Tcp_ConnectionState is Tcp_Listen.
5.25.4.3.10	0x3d93a121, 0xde18, 0x4496, 0x87, 0xc2, 0xb7, 0x83, 0x0a, 0x92, 0xee, 0x0e	EFI_TCP6_PROTOCOL.Configure() – Configure() returns EFI_SUCCESS with valid parameters.	3. Call Configure() when TcpConfigData is NULL. The instance should be reset correctly.
5.25.4.3.11	0x9f6ad319, 0x0b1c, 0x40a0, 0x91, 0xee, 0xf9, 0x4e, 0x1a, 0xff, 0x9e, 0x09	The Tcp instance should enter into Tcp_Closed state after being reset. Call GetModeData() and the return value should be EFI NOT STARTED	4. Call GetModeData(). The return value should be EFI NOT STARTED.
5.25.4.3.12	0xea63c75a, 0x839f, 0x47b4, 0xad, 0x6c, 0x6f, 0xcf, 0x5f, 0xfd, 0x97, 0xfc	EFI_TCP6_PROTOCOL.Configure() – Configure() returns EFI_SUCCESS with valid parameters.	5. Call Configure() with valid parameters.
5.25.4.3.13	0x0275b281, 0xf70e, 0x478d, 0xa6, 0x20, 0xa3, 0x28, 0x52, 0x5a, 0xd8, 0x07	The Configure() should correctly set the data as expected.	6. Call GetModeData() and check whether the data is set as expected.

21.4.4 Connect()

Number	GUID	Assertion	Test Description
5.25.4.4.1	0xa092e680, 0x27e9, 0x483b, 0xb3, 0xdb, 0x07, 0xb8, 0x69, 0x1a, 0xb7, 0xfc	`EFI_TCP6_PROTOCOL.Connect() - Connect()` returns `EFI_NOT STARTED` when the instance hasn't been configured.	Call `Connect()` before the instance is configured, the return status should be `EFI_NOT_STARTED`.
5.25.4.4.2	0x1e456f02, 0x7477, 0x4933, 0x84, 0xf9, 0x12, 0x9a, 0x8f, 0x64, 0x80, 0xa5	`EFI_TCP6_PROTOCOL.Connect() - Connect()` returns `EFI_INVALID PARAMETER` when the token is `NULL`.	Call `Connect()` with the `NULL` token, the return status should be `EFI_NOT_STARTED`.
5.25.4.4.3	0x3b5e2748, 0x1549, 0x465f, 0x98, 0x37, 0x67, 0xd9, 0x48, 0xdf, 0x50, 0x9f	`EFI_TCP6_PROTOCOL.Connect() - Connect()` returns `EFI_INVALID PARAMETER` when the token's event is `NULL`.	Call `Connect()` when the token's event is `NULL`, the return status should be `EFI_NOT_STARTED`.
5.25.4.4.4	0x73f9316d, 0xbfcb, 0x4c3a, 0xbd, 0x75, 0x56, 0xb7, 0x03, 0x1d, 0x58, 0x30	`EFI_TCP6_PROTOCOL.Connect() - Connect()` returns `EFI_ACCESS DENIED` when the instance is configured in passive mode.	1. Call `Configure()` to configure the instance as passive mode. 2. Call `Connect()` with valid parameters, the return status should be `EFI_ACCESS_DENIED`.
5.25.4.4.5	0xd15151a5, 0xf62b, 0x4203, 0x8e, 0x16, 0x47, 0x3b, 0x4a, 0x13, 0xd0, 0x89	`EFI_TCP6_PROTOCOL.Connect() - Connect()` returns `EFI_ACCESS DENIED` when the instance is not in `TCP_CLOSED` state.	5.25.4.4.5 to 5.25.4.4.6 belong to one case 1. Call `Configure()` to configure the instance as active mode. 2. Call `GetModeData()` to check that the instance's state should be `TCP_SYN_SENT`.
5.25.4.4.6	0xf9de93e5, 0x4d4d, 0x45ab, 0x95, 0x0d, 0xc1, 0x53, 0x75, 0x51, 0xec, 0xb5	`EFI_TCP6_PROTOCOL.Connect() - Connect()` returns `EFI_ACCESS DENIED` when the instance is not in `TCP_CLOSED` state.	3. Call `Connect()` when the instance's state is not in `TCP_SYN_SENT`, The return status should be `EFI_ACCESS_DENIED`.

Number	GUID	Assertion	Test Description
5.25.4.4.7	0xfb14d45a, 0xa20d, 0x4c96, 0x94,0xc7, 0x86,0xc6, 0xc1,0x09, 0x9d,0xa4	EFI_TCP6_PROTOCOL.Connect() – Connect() must return EFI_CONNECTION_REFUSED when the instance is in SYN-RCVD state & receive a RST	1. Call EFI_TCP6_SERVICE_BINDING_PROTOCOL.CreateChild() to create a newTcp6 child. 2. Call EFI_TCP6_PROTOCOL.Configure() to configure the new instance. 3. Call EFI_TCP6_PROTOCOL.Connect() Receive SYN & Send a SYN to put TCP state machine in SYN-RCVD state. 4. Send a RST & check Connection Token state to be changed to EFI_CONNECTION_REFUSED 5. Call EFI_TCP6_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp6 child and clean up the environment.
5.25.4.4.8	0x3caf2371, 0x32e9, 0x4e29, 0x87, 0x64, 0x44, 0x12, 0x14, 0xcb, 0xa1, 0x63	EFI_TCP6_PROTOCOL.Connect() - Connect() returns EFI_SUCCESS with valid parameters.	5.25.4.4.8 to 5.25.4.4.12 belong to one case 1. Call Connect() with valid parameters, the return status should be EFI_SUCCESS.
5.25.4.4.9	0xcd1704c9, 0xbabe, 0x4447, 0xaf, 0xda, 0xd2, 0x08, 0xc6, 0x9b, 0xd8, 0x8f	After the EFI_TCP6_PROTOCOL.Connect() is called, the EFI should send SYN packet successfully.	2. Check whether the SYN packet is sent by SCT successfully.
5.25.4.4.10	0x6e521181, 0x2a24, 0x4697, 0xbb, 0x83, 0x4b, 0xd9, 0xde, 0x5b, 0x89, 0xc0	The TCP instance should acknowledge EMS's SYN packet successfully.	3. EMS send SYN packet to SCT side. 4. Check whether the ACK packet is sent by SCT successfully.
5.25.4.4.11	0x1944bcf5, 0x9123, 0x469b, 0x86, 0xc2, 0x5c, 0x98, 0x7a, 0x39, 0xfe, 0x59	The connection token's event should be signaled successfully after 3-way handshakes are done.	5. Check whether the token's event is signaled after the 3-way handshake are done.

Number	GUID	Assertion	Test Description
5.25.4.4.12	0xcdae7179, 0xf66e, 0x4980, 0x9c, 0x08, 0x89, 0x0a, 0xe2, 0xcc, 0x4d, 0x46	The connection token's status should be modified to `EFI_SUCCESS` after 3-way handshakes are done.	6. Check whether the token's status is modified as expected after the 3-way handshake are done.

21.4.5 Accept()

Number	GUID	Assertion	Test Description
5.25.4.5.1	0x30ec775a, 0xcefa, 0x4d56, 0x8c, 0x88, 0xa2, 0xdc, 0x75, 0x13, 0x56, 0x9c	EFI_TCP6_PROTOCOL.Accept() - Accept() returns EFI_NOT STARTED when the instance hasn't been configured.	Call Accept() before the instance is configured, the return status should be EFI_NOT_STARTED.
5.25.4.5.2	0x08809174, 0x9447, 0x4956, 0x93, 0x0d, 0xa7, 0xb2, 0xa7, 0x63, 0x80, 0x9f	EFI_TCP6_PROTOCOL.Accept() - Accept() returns EFI_ACCESS DENIED when the instance isn't in passive mode.	Call Accept() with the instance in active mode, the return status should be EFI_ACCESS_DENIED.
5.25.4.5.3	0x8f109af6, 0x55fe, 0x4f5c, 0x8b, 0x84, 0x22, 0xa8, 0x42, 0x4b, 0xc7, 0xf9	EFI_TCP6_PROTOCOL.Accept() - Accept() returns EFI_ACCESS DENIED when the listen token has already been queued.	1. Call Accept() with valid parameters. 2. Call Accept() with the same token again, the return status should be EFI_ACCESS_DENIED.
5.25.4.5.4	0xfc47ef2f, 0xc11c, 0x488c, 0x88, 0x21, 0xc8, 0xef, 0x3e, 0x2f, 0x3e, 0x7e	EFI_TCP6_PROTOCOL.Accept() - Accept() returns EFI_INVALID PARAMETER when the listen token is NULL.	Call Accept() when the listen token is NULL, the return status should be EFI_INVALID_PARAMETER.
5.25.4.5.5	0xf336471a, 0x6809, 0x4886, 0x95, 0x37, 0x2f, 0xf8, 0xb7, 0x5e, 0x5e, 0x8d	EFI_TCP6_PROTOCOL.Accept() - Accept() returns EFI_INVALID PARAMETER when the event in the listen token is NULL.	Call Accept() when the event in the listen token is NULL, the return status should be EFI_INVALID_PARAMETER.
5.25.4.5.6	0x19464085, 0x7ccc, 0x42a8, 0xbd, 0x81, 0x8a, 0x21, 0x0a, 0xf4, 0x70, 0xcd	EFI_TCP6_PROTOCOL.Accept() - Accept() returns EFI_SUCCESS with valid parameters.	5.25.4.5.6 to 5.25.4.5.14 belong to one case 1. Call Accept() with valid parameters, the return status should be EFI_SUCCESS.
5.25.4.5.7	0x2953f594, 0x8f06, 0x42f6, 0x8e, 0x7b, 0xc7, 0x8f, 0xf5, 0xc2, 0x4e, 0xa9	The TCP instance should acknowledge EMS's SYN packet successfully.	2. EMS sent SYN packet to SCT side. 3. Check whether SCT accepts the SYN packet and send back SYN to EMS.

Number	GUID	Assertion	Test Description
5.25.4.5.8	0x04df3e6d, 0x599b, 0x43df, 0xb9, 0xb4, 0xf4, 0xaf, 0xc8, 0x3f, 0x48, 0x49	The listen token's event should be signaled successfully after 3-way handshakes are done.	4. Check whether the token's event is signaled after the 3-way handshake are done.
5.25.4.5.9	0x727bb534, 0xd41f, 0x4132, 0x88, 0xbb, 0x8e, 0x02, 0xc6, 0x84, 0x2c, 0xbf	The listen token's status should be modified to **EFI_SUCCESS** after 3-way handshakes are done.	5. Check whether the token's status is modified as expected after the 3-way handshake are done.
5.25.4.5.10	0xf88ff924, 0xfb1c, 0x4252, 0x9a, 0xa9, 0x18, 0xff, 0x46, 0xae, 0x75, 0x90	The child handle contained in the listen token should not be **NULL**.	6. Check whether the child handle contained in the token is **NULL**.
5.25.4.5.11	0x1bff0f74, 0x465c, 0x4e25, 0xa6, 0x80, 0x8d, 0x2d, 0x43, 0x52, 0x28, 0x4d	The child handle contained in the listen token should be in **TCP_ESTABLISHED** state.	7. Check whether the child handle contained in the token is in correct state.
5.25.4.5.112	0x06850748, 0xc64f, 0x4d44, 0xba, 0x43, 0x4e, 0xfb, 0xde, 0x2d, 0x2c, 0x7d	The child handle contained in the listen token should share the same configuration with its parent handle	8. Check whether the child handle contained in the token has the same configuration as its parent handle.
5.25.4.5.113	0x7415d9d3, 0x054f, 0x4a18, 0xb8, 0xbf, 0x6f, 0x6a, 0xae, 0xf4, 0xbc, 0x3f	Data communication should be correct on the child handle – Return value should be correct.	9. **Receive()** with valid parameters, The return status should be **EFI_SUCCESS**. 10. Check whether the event is signaled and the status is modified correctly.
5.25.4.5.114	0x72834f64, 0x41fe, 0x46ab, 0x8b, 0x39, 0x64, 0xe3, 0x9f, 0x28, 0x6f, 0x71	Data communication should be correct on the child handle – Data content should be as expected.	11. Check whether the data length and data content for the **Receive()** is correct.

21.4.6 Transmit()

Number	GUID	Assertion	Test Description
5.25.4.6.1	0xef652675, 0x3d29, 0x4c9c, 0xbe, 0x90, 0xd3, 0xd6, 0x53, 0xac, 0x7b, 0x3c	`EFI_TCP6_PROTOCOL.Transmit()` - `Transmit()` returns `EFI_NOT_STARTED` with the instance hasn't been configured.	Call `Transmit()` before the instance is configured, the return status should be `EFI_NOT_STARTED`.
5.25.4.6.2	0x31cbe783, 0xdea8, 0x4d05, 0x9b, 0x0b, 0xf0, 0x87, 0x5d, 0x3b, 0x07, 0x24	`EFI_TCP6_PROTOCOL.Transmit()` - `Transmit()` returns `EFI_INVALID_PARAMETER` when the token is `NULL`.	Call `Transmit()` when the token is `NULL`, the return status should be `EFI_INVALID_PARAMETER`.
5.25.4.6.3	0xcbb9c387, 0x96ef, 0x4834, 0xba, 0xeb, 0xe1, 0x9e, 0xca, 0x99, 0xae, 0xc7	`EFI_TCP6_PROTOCOL.Transmit()` - `Transmit()` returns `EFI_INVALID PARAMETER` when event in the token is `NULL`.	Call `Transmit()` when the event in token is `NULL`, the return status should be `EFI_INVALID_PARAMETER`.
5.25.4.6.4	0xfdd4086f, 0xeffd, 0x4e7a, 0x93, 0xd2, 0x73, 0x74, 0x6d, 0x0f, 0x63, 0x18	`EFI_TCP6_PROTOCOL.Transmit()` - `Transmit()` returns `EFI_INVALID PARAMETER` when the `TxData` is `NULL`.	Call `Transmit()` when `TxData` is `NULL`, the return status should be `EFI_INVALID_PARAMETER`.
5.25.4.6.5	0xb3528e10, 0xd5ae, 0x4960, 0xb5, 0x03, 0xdd, 0x89, 0xd0, 0xf7, 0x6a, 0x09	`EFI_TCP6_PROTOCOL.Transmit()` - `Transmit()` returns `EFI_INVALID PARAMETER` when the `FragmentCount` is `0`.	Call `Transmit()` when `FragmentCount` is `0`, the return status should be `EFI_INVALID_PARAMETER`.
5.25.4.6.6	0xa8598edc, 0x469c, 0x4803, 0xbd, 0xf4, 0x37, 0xbf, 0x06, 0x8f, 0x41, 0x87	`EFI_TCP6_PROTOCOL.Transmit()` - `Transmit()` returns `EFI_INVALID PARAMETER` when the data length is not equal to the sum of all fragment buffers' length.	Call `Transmit()` when the data length is not equal to the sum of all fragment buffers' length, the return status should be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.25.4.6.7	0x6231d7c6, 0xf61c, 0x4d6b, 0x94, 0xc4, 0xc6, 0xfc, 0x73, 0x59, 0xb6, 0xe2	`EFI_TCP6_PROTOCOL.Transmit() - Transmit()` returns `EFI_ACCESS_DENIED` when the event has already been queued.	1. Call `Transmit()` with valid parameters to send a data packet larger than MSS. The packet will be segmented to several bulks. 2. No `ACK` will be sent by EMS for the first segment. Hence, the event for the transmit token will stay in the queue. 3. Call `Transmit()` with the same event and valid other parameters again, the return status should be `EFI_ACCESS_DENIED`.
5.25.4.6.8	0x5172270a, 0xf411, 0x4197, 0xbd, 0x34, 0x82, 0xc5, 0xc0, 0xe9, 0xa7, 0xcf	`EFI_TCP6_PROTOCOL.Transmit() - Transmit()` returns `EFI_ACCESS_DENIED` when the instance has not been connected in active mode.	Call `Transmit()` in active mode before the 3-way handshake establishes, the return status should be `EFI_ACCESS_DENIED`.
5.25.4.6.9	0x13fa7b6c, 0xdc0f, 0x4f9e, 0xae, 0x4a, 0x9e, 0x3e, 0x11, 0x02, 0xe2, 0x98	`EFI_TCP6_PROTOCOL.Transmit() - Transmit()` returns `EFI_ACCESS_DENIED` when the instance has not been accepted in passive mode.	Call `Transmit()` in passive mode before the 3-way handshake establishes, the return status should be `EFI_ACCESS_DENIED`.
5.25.4.6.10	0x9192cade, 0x7b3d, 0x44bf, 0x8a, 0xe7, 0x36, 0x28, 0x89, 0xd8, 0x76, 0x23	`EFI_TCP6_PROTOCOL.Transmit() - Transmit()` returns `EFI_ACCESS_DENIED` when the instance has been closed.	Call `Transmit()` with valid parameters when the instance has been closed, the return status should be `EFI_ACCESS_DENIED`.
5.25.4.6.11	0x8652c924, 0xf3d0, 0x43cc, 0x8b, 0xda, 0x8c, 0xd7, 0x16, 0xdc, 0xb3, 0xa0	`EFI_TCP6_PROTOCOL.Transmit() - Transmit()` returns `EFI_SUCCESS` with valid parameters.	5.25.4.6.11 to 5.26.4.6.15 belong to one case 1. Call `Transmit()` with valid parameters, the return status should be `EFI_SUCCESS`.
5.25.4.6.12	0x096d60c6, 0xf036, 0x46be, 0xb0, 0xb2, 0x95, 0x13, 0xcf, 0xf1, 0x80, 0x81	The transmitted packet should be delivered to network after the `Transmit()` is called.	2. Check whether EMS could receive the transmitted packets in time.

Number	GUID	Assertion	Test Description
5.25.4.6.13	0x0d441d88, 0xd3eb, 0x4b97, 0x9c, 0x3d, 0xc9, 0xbe, 0xec, 0x2d, 0xeb, 0xc5	The token event should be signaled after the packet is sent.	3. Check whether the token event is signaled.
5.25.4.6.14	0x9b0d226f, 0x4bc4, 0x4e1c, 0xb7, 0x07, 0xa1, 0x8e, 0x3a, 0x7b, 0x30, 0xf6	The token status should be changed to `EFI_SUCCESS` after the packet is sent.	4. Check whether the token status is changed to `EFI_SUCCESS`.
5.25.4.6.15	0xfaca42a2, 0xa769, 0x4af9, 0x90, 0xcb, 0xf0, 0xd0, 0x5f, 0xf0, 0x8e, 0x03	The packet length and content for the transmission should be correct.	5. Check whether the packet length and content is correct.

21.4.7 Receive()

Number	GUID	Assertion	Test Description
5.25.4.7.1	0xd54cf9ed, 0x80e9, 0x44c0, 0x81, 0x25, 0xa7, 0x85, 0x2b, 0xbf, 0xec, 0x83	EFI_TCP6_PROTOCOL.Receive() - Receive() returns EFI_NOT_STARTED when the instance hasn't been configured.	Call Receive() before the instance is configured, the return status should be EFI_NOT_STARTED.
5.25.4.7.2	0xa682e94a, 0x5d64, 0x4646, 0x98, 0x8d, 0x1e, 0x7a, 0xb1, 0x68, 0x8d, 0xb1	EFI_TCP6_PROTOCOL.Receive() - Receive() returns EFI_INVALID_PARAMETER when the token is NULL.	Call Receive() when the token is NULL, the return status should be EFI_INVALID_PARAMETER.
5.25.4.7.3	0xad9f6b64, 0xd0a0, 0x4bef, 0xbe, 0xdb, 0xf0, 0x42, 0x9b, 0x00, 0xfd, 0x76	EFI_TCP6_PROTOCOL.Receive() - Receive() returns EFI_INVALID PARAMETER when event in the token is NULL.	Call Receive() when the event in token is NULL, the return status should be EFI_INVALID_PARAMETER.
5.25.4.7.4	0xc9a6cae7, 0x6e5e, 0x4c04, 0x9b, 0x1e, 0x27, 0xf3, 0x61, 0x34, 0x83, 0x8a	EFI_TCP6_PROTOCOL.Receive() - Receive() returns EFI_INVALID PARAMETER when the RxData is NULL.	Call Receive() when RxData is NULL, the return status should be EFI_INVALID_PARAMETER.
5.25.4.7.5	0x0cb365ff, 0xf855, 0x4ef5, 0xb8, 0xe5, 0xef, 0x2b, 0xc2, 0xd4, 0x6a, 0x7d	EFI_TCP6_PROTOCOL.Receive() - Receive() returns EFI_INVALID PARAMETER when the FragmentCount is 0.	Call Receive() when FragmentCount is 0, the return status should be EFI_INVALID_PARAMETER.
5.25.4.7.6	0x3ad62087, 0xfaf8, 0x4864, 0x9b, 0xd9, 0xad, 0xb1, 0x16, 0x6a, 0x54, 0x62	EFI_TCP6_PROTOCOL.Receive() - Receive() returns EFI_INVALID PARAMETER when the data length is not equal to the sum of all fragment buffers' length.	Call Receive() when the data length is not equal to the sum of all fragment buffers' length, the return status should be EFI_INVALID_PARAMETER.
5.25.4.7.7	0x4b325e98, 0x9ae8, 0x4a2b, 0x9e, 0x3e, 0x0a, 0xcf, 0x4a, 0x7e,0x69, 0x53	EFI_TCP6_PROTOCOL.Receive() - Receive() returns EFI_ACCESS_DENIED when the event has already been queued.	1. Call Receive() with valid parameters but no packet is sent from EMS. The receiving token will stay in the queue. 2. Call Receive() with the same event and other valid parameters again, The return status should be EFI_ACCESS_DENIED.

Number	GUID	Assertion	Test Description
5.25.4.7.8	0xddef303a, 0x3180, 0x466f, 0x80, 0x55, 0x26, 0xa4, 0x2f, 0x12, 0x1b, 0x78	EFI_TCP6_PROTOCOL.Receive() - Receive() returns EFI_ACCESS_DENIED when the instance has not been connected in active mode.	Call Receive() in active mode before the 3-way handshake establishes, the return status should be EFI_ACCESS_DENIED.
5.25.4.7.9	0x59b5cc95, 0xb0e9, 0x4cd6, 0xb1, 0x1d, 0x74, 0xcc, 0x26, 0x72, 0x33, 0x67	EFI_TCP6_PROTOCOL.Receive() - Receive() returns EFI_ACCESS_DENIED when the instance has not been accepted in passive mode.	Call Receive() in passive mode before the 3-way handshake establishes, the return status should be EFI_ACCESS_DENIED.
5.25.4.7.10	0xd985c3a0, 0xb98c, 0x4ad9, 0xb9, 0x9c, 0x1c, 0x5c, 0xfc, 0x4b, 0xea, 0xad	EFI_TCP6_PROTOCOL.Receive() - Receive() returns EFI_ACCESS_DENIED when the instance has been closed.	Call Receive() with valid parameters when the instance has been closed, The return status should be EFI_ACCESS_DENIED.
5.25.4.7.11	0xdcae30da, 0x090c, 0x441f, 0xbd, 0xa9, 0x02, 0x28, 0x4d, 0x2e, 0xab, 0xcb	EFI_TCP6_PROTOCOL.Receive() – Receive() must return EFI_CONNECTION_FIN. When the communication peer has closed the connection and there is no any buffered data in the receive buffer of this instance	1. Call EFI_TCP6_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new Tcp6 child. 2. Call EFI_TCP6_PROTOCOL.Configure() to configure the new instance. 3. Call EFI_TCP6_PROTOCOL.Connect() & complete a 3-Way handshake 4. Send a *FIN/ACK* to close this connection 5.Call EFI_TCP6_SERVICE_BINDING_PROTOCOL.Receive() & check if its return status is EFI_CONNECTION_REFUSED 6. Call EFI_TCP6_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the created Tcp6 child and clean up the environment.
5.25.4.7.12	0x2003bb96, 0xf32d, 0x48ca, 0x8e, 0x5a, 0x2c, 0x71, 0x6e, 0x95, 0x33, 0xf7	EFI_TCP6_PROTOCOL.Receive() - Receive() returns EFI_SUCCESS with valid parameters.	5.25.4.7.12 to 5.26.4.7.15 belong to one case 1. Call Receive() with valid parameters, the return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.25.4.7.13	0x5df1bf20, 0x8c5d, 0x4ef4, 0xb3, 0x70, 0xfd, 0x78, 0x14, 0xf2, 0x0a, 0x88	The token event should be signaled after the packet is received.	2. Check whether the token event is signaled.
5.25.4.7.14	0xb65c6862, 0xebad, 0x4d51, 0xa1, 0xac, 0x73, 0xc0, 0x19, 0x24, 0x00, 0x8d	The token status should be changed to `EFI_SUCCESS` after the packet is received.	3. Check whether the token status is changed to `EFI_SUCCESS`.
5.25.4.7.15	0xfc18f3ec, 0xe779, 0x4730, 0x82, 0x24, 0xea, 0xdd, 0x9a, 0x4f, 0xd4, 0xf9	The packet length and content for the received packet should be correct.	4. Check whether the packet length and content is correct.

21.4.8 Close()

Number	GUID	Assertion	Test Description
5.25.4.8.1	0x97e34ed, 0x8b15, 0x479c, 0x9d, 0xa9, 0x57, 0x26, 0x58, 0x18, 0x72, 0x2d	EFI_TCP6_PROTOCOL.Close() - Close() returns EFI_NOT_STARTED with the instance hasn't been configured.	Call Close() before the instance is configured, the return status should be EFI_NOT_STARTED.
5.25.4.8.2	0x49ea02d4, 0x0022, 0x49c6, 0xac, 0x02, 0x3d, 0xe9, 0x96, 0x86, 0x48, 0xb9	EFI_TCP6_PROTOCOL.Close() - Close() returns EFI_INVALID_PARAMETER when the token is NULL.	Call Close() when the token is NULL, the return status should be EFI_INVALID_PARAMETER.
5.25.4.8.3	0x43dd8f75, 0x40d1, 0x4f54, 0x81, 0x5c, 0x81, 0x3e, 0xed, 0x71, 0x37, 0x89	EFI_TCP6_PROTOCOL.Close() - Close() returns EFI_INVALID PARAMETER when event in the token is NULL.	Call Close() when the event in token is NULL, the return status should be EFI_INVALID_PARAMETER.
5.25.4.8.4	0xed7c5cd6, 0x0d5b, 0x4951,0xaa, 0x37, 0x96, 0xea, 0xe8, 0xa2, 0x7b, 0x89	EFI_TCP6_PROTOCOL.Close() - Transmit() returns EFI_ACCESS DENIED when the token event has already been used.	1. Call Close() with valid parameters to perform a graceful close, but the EMS will send back no ACK. Hence the close event will stay in the queue. 2. Call Close() with the same event and valid other parameters, the return status should be EFI_ACCESS_DENIED.
5.25.4.8.5	0x772e9c64, 0xc345, 0x4470, 0x9d, 0x93, 0x61, 0x71, 0xf8, 0x95, 0x52, 0x71	EFI_TCP6_PROTOCOL.Close() - Transmit() returns EFI_ACCESS DENIED when the last close has not been finished.	1. Call Close() with valid parameters to perform a graceful close, but the EMS will send back no ACK. Hence the close event will stay in the queue and the first close will keep unfinished. 2. Call Close() with different event and valid other parameters, the return status should be EFI_ACCESS_DENIED.
5.25.4.8.6	0x45385c8f, 0xa54a, 0x481d, 0xb2, 0x64, 0x3f, 0xc8, 0x12, 0xd1, 0x50, 0x39	EFI_TCP6_PROTOCOL.Close() - Close() returns EFI_SUCCESS with valid parameters.	5.25.4.8.6 to 5.26.4.8.11 belong to one case 1. Call Close() with valid parameters, the return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.25.4.8.7	0x764114c1, 0x2ba3, 0x4791, 0x96, 0x33, 0x35, 0xb2, 0x0b, 0x88, 0x43, 0xf4	The FIN packet should be sent by SCT correctly.	2. Check whether the FIN packet is sent out in time.
5.25.4.8.8	0x10e12a40, 0x97c5, 0x467d, 0x97, 0x90, 0x0f, 0x58, 0x11, 0x84, 0xf1, 0x21	The last ACK packet should be sent out correctly by SCT after receiving EMS's FIN packet.	3. After EMS receives the FIN packet. It sends out FIN/ACK packet to SCT. 4. Check whether the last ACK packet is sent out by SCT.
5.25.4.8.9	0x333bdd81, 0x801d, 0x4aa1, 0x8c, 0x71, 0x31, 0x1d, 0x0f, 0x15, 0x89, 0x57	The event in close token should be signaled.	5. After the 4-way handshake finishes, check whether the close token's event is signaled.
5.25.4.8.10	0x33fa7b0c, 0x9e89, 0x4138, 0xa9, 0xaf, 0x3e, 0xee, 0x54, 0xa3, 0x90, 0x04	The status of close token should be changed to EFI SUCCESS.	6. Check whether the close token's status is changed to EFI_SUCCESS.
5.25.4.8.11	0x1cdb5be1, 0xf8d0, 0x4570, 0x8e, 0x99, 0x7c, 0x6b, 0x6b, 0xb9, 0x76, 0x73	The status of the TCP instance should be TCP_CLOSED after the successful close().	7. Check whether the instance's state is changed to TCP_CLOSED.
5.25.4.8.12	0x134177f3, 0x458a, 0x4088, 0x8e, 0x29, 0x84, 0x75, 0x1d, 0x68, 0x41, 0x43	EFI_TCP6_PROTOCOL.Close() - Close() returns EFI_SUCCESS with valid parameters when there is tokens in the queue.	5.25.4.8.12 to 5.26.4.8.16 belong to one case 1. Transmit a large packet including several segments from SCT. EMS sends out ACK to the segments except for the last one. Hence the transmit token will pending in the queue. 2. Call Close() to close the connection, the return status should be EFI_SUCCESS.
5.25.4.8.13	0xb124b733, 0x1f2e, 0x4493, 0x95, 0xf6, 0x8e, 0xa3, 0x93, 0x1a, 0x8d, 0x6f	The FIN packet should be sent out immediately the last ACK is received.	3. EMS sends out ACK for the last segment. 4. Check whether the SCT sends out FIN.

Number	GUID	Assertion	Test Description
5.25.4.8.14	0xede2639e, 0xa23b, 0x4ae5, 0xa0, 0xb3, 0x9d, 0x1c, 0x1b, 0x27, 0x90, 0x3d	The close token's event should be signaled and status be changed correctly after the 4-way handshake finishes.	5. EMS sends out `FIN` packet back to finish the 4-way handshake. 6. Check whether the close token's event is signaled. 7. Check whether the close token's status is changed to `EFI_SUCCESS`.
5.25.4.8.15	0x7c552532, 0x55ea, 0x46ac, 0x86, 0xf8, 0x0d, 0x1c, 0x27, 0x34, 0x71, 0xed	The TCP instance's state should be `TCP_CLOSED` after the 4-way handshake finishes.	8. Check whether the instance's state is changed to `TCP_CLOSED` after the 4-way handshake finishes.
5.25.4.8.16	0xdfe82050, 0x3325, 0x4dcf, 0xa0, 0xdc, 0xb7, 0x20, 0xa6, 0x72, 0xe9, 0xf0	The pending transmit token should be signaled after the close finishes.	9. Check whether the pending token is signaled or not.
5.25.4.8.17	0x362144c2, 0xd822, 0x445a, 0x8d, 0x8d, 0x1a, 0x27, 0xcd, 0xf3, 0x17, 0x40	`EFI_TCP6_PROTOCOL.Close() - Close()` to close and pending tokens should be signaled.	1. Call `Receive()` to receive a incoming packet when there's no packet sent from EMS. The receiving token will stay in the queue. 2. Call `Close()` to close the connection gracefully. 3. Check whether the receiving token is signaled and its state modified.

21.5 EFI_IP6_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_IP6_PROTOCOL Section.

21.5.1 CreateChild()

Number	GUID	Assertion	Test Description
5.25.5.1.1	0xc5a98289, 0xf32c, 0x4433, 0x81, 0xae, 0xa9, 0x10, 0xa3, 0x51, 0x0c, 0x32	EFI_IP6_SERVICE_BINDING_PROTOCOL.CreateChild() - CreateChild() returns EFI_INVALID_PARAMETER with a NULL ChildHandle.	Call CreateChild() with a NULL ChildHandle, the return status should be EFI_INVALID_PARAMETER.
5.25.5.1.2	0x29d8f02c, 0xd19f, 0x48ec, 0xab, 0x8e, 0xb9, 0x10, 0x54, 0x10, 0x34, 0xc4	EFI_IP6_SERVICE_BINDING_PROTOCOL.CreateChild() - CreateChild() returns EFI_SUCCESS with 1st valid ChildHandle.	5.25.5.1.2 to 5.25.5.1.5 belong to one case 1. Call CreateChild() with the 1st valid ChildHandle, the return status should be EFI_SUCCESS.
5.25.5.1.3	0x3e7a34ce, 0x0a96, 0x4029, 0xa0, 0x0a, 0xd2, 0x7c, 0x75, 0x9c, 0xf0, 0x2d	EFI_IP6_SERVICE_BINDING_PROTOCOL.CreateChild() - CreateChild() returns EFI_SUCCESS with 2nd valid ChildHandle.	2. Call CreateChild() with the 2nd valid ChildHandle, the return status should be EFI_SUCCESS.
5.25.5.1.4	0x8e7bf890, 0x6109, 0x4d71, 0xa5, 0xb7, 0x83, 0x85, 0x0c, 0x5f, 0x78, 0x00	EFI_IP6_SERVICE_BINDING_PROTOCOL.DestroyChild() - DestroyChild() returns EFI_SUCCESS with 2nd valid ChildHandle.	3. Call DestroyChild() with the 2nd valid ChildHandle, the return status should be EFI_SUCCESS.
5.25.5.1.5	0x974cd2fd, 0x79da, 0x4008, 0x92, 0x5a, 0x5c, 0x29, 0xa3, 0x7e, 0xd7, 0xb3	EFI_IP6_SERVICE_BINDING_PROTOCOL.DestroyChild() - DestroyChild() returns EFI_SUCCESS with 1st valid ChildHandle.	3. Call DestroyChild() with the 1st valid ChildHandle, the return status should be EFI_SUCCESS.

21.5.2 DestoryChild()

Number	GUID	Assertion	Test Description
5.25.5.2.1	0x5b7d1b2f, 0x41f1, 0x4787, 0xa6, 0xb5, 0xfa, 0x28, 0x9e, 0x34, 0xcd, 0xd3	**EFI_IP6_SERVICE_BINDING_PROTOCOL.DestoryChild() - DestoryChild()** returns **EFI_INVALID_PARAMETER** with a **NULL** *ChildHandle*.	Call **DestoryChild()** with a **NULL** *ChildHandle*, the return status should be **EFI_INVALID_PARAMETER.**

21.5.3 GetModeData()

Number	GUID	Assertion	Test Description
5.25.5.3.1	0xc8a6f564, 0x2320, 0x46fa, 0xbf, 0x2a, 0x0b, 0x77, 0x3c, 0x71, 0x1d, 0xf6	**EFI_IP6_PROTOCOL.GetModeData() - GetModeData()** returns **EFI_SUCCESS** with valid parameters	5.25.5.3.1 to 5.25.5.3.2 belong to one case 1. Call **GetModeData()** with valid parameters, the return status should be **EFI_SUCCESS.**
5.25.5.3.2	0x3919816b, 0xf3bd, 0x4177, 0x8d, 0x90, 0xf3, 0xca, 0xba, 0x20, 0x9a, 0xc2	Validate the *IP6ModeData.IsConfigured*	2. The value of *IP6ModeData.IsConfigured* should be **FALSE**.

21.5.4 Configure()

Number	GUID	Assertion	Test Description
5.25.5.4.1	0x99fe5cde, 0xdccb, 0x4d55, 0xab, 0xb4, 0xa1, 0xdf, 0x73, 0x30, 0x2d, 0x4b	`EFI_IP6_PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` when *`Ip6ConfigData.StationAddress`* is neither zero nor a valid unicast Ipv6 address.	Call `Configure()` when *`IpConfigData.StationAddress`* is neither zero nor a valid unicast Ipv6 address, the return status should be `EFI_INVALID_PARAMETER`.
5.26.5.4.2	0xa0998aa3, 0x7f5e, 0x401f, 0x8f, 0x3d, 0xeb, 0xe9, 0x09, 0x5c, 0xbd, 0x7b	`EFI_IP6_PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` when *`Ip6ConfigData.StationAddress`* is neither zero nor one of configured Ipv6 address.	Call `Configure()` when *`Ip6ConfigData.StationAddress`* is neither zero nor one of configured Ipv6 address, the return status should be `EFI_INVALID_PARAMETER`.
5.26.5.4.3	0xafca1a79, 0xc38f, 0x4e5a, 0x8b, 0xa9, 0x33, 0xaf, 0xd9, 0x04, 0x7b, 0xbf	`EFI_IP6_PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` when *`Ip6ConfigData.DefaultProtocol`* is invalid.	Call `Configure()` when *`Ip6ConfigData.DefaultProtocol`* is invalid, the return status should be `EFI_INVALID_PARAMETER`.
5.25.5.4.4	0xcc598692, 0xc3e7, 0x4008, 0x91, 0xc2, 0x29, 0xf6, 0xc4, 0x0f, 0x74, 0x41	`EFI_IP6_PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` when *`Ip6ConfigData.DefaultProtocol`* is invalid.	Call `Configure()` when *`Ip6ConfigData.DefaultProtocol`* is invalid, the return status should be `EFI_INVALID_PARAMETER`.
5.25.5.4.5	0x6aa9538e, 0x3e88, 0x4309, 0xab, 0x52, 0x94, 0xc5, 0x09, 0x3e, 0x9a, 0x34	`EFI_IP6_PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` when *`Ip6ConfigData.DefaultProtocol`* is invalid.	Call `Configure()` when *`Ip6ConfigData.DefaultProtocol`* is invalid, the return status should be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.25.5.4.6	0x84a3a2cb, 0x3bc5, 0x47f9, 0xab, 0xb4, 0xd5, 0xa6, 0x89, 0xfa, 0x1a, 0x80	`EFI_IP6_PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` when *`Ip6ConfigData.DefaultProtocol`* is invalid.	Call `Configure()` when ***`Ip6ConfigData.DefaultProtocol`*** is invalid, the return status should be `EFI_INVALID_PARAMETER`.
5.25.5.4.7	0x43804768, 0xca58, 0x4f59, 0xa8, 0x18, 0x1b, 0x0e, 0x9a, 0x0f, 0xc1, 0xa6	`EFI_IP6_PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` when *`Ip6ConfigData.DefaultProtocol`* is invalid.	Call `Configure()` when *`Ip6ConfigData.DefaultProtocol`* is invalid, the return status should be `EFI_INVALID_PARAMETER`.
5.25.5.4.8	0xecfe10f7, 0xce1f, 0x4711, 0xb0, 0xc8, 0xd8, 0x56, 0xe5, 0x35, 0x4a, 0x82	`EFI_IP6_PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` when *`Ip6ConfigData.DefaultProtocol`* is invalid.	Call `Configure()` when ***`Ip6ConfigData.DefaultProtocol`*** is invalid, the return status should be `EFI_INVALID_PARAMETER`.
5.25.5.4.9	0xa9c4db07, 0x17f3, 0x43e3, 0xa7, 0x43, 0x78, 0xe9, 0x51, 0xb7, 0x35, 0xce	`EFI_IP6_PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` when *`Ip6ConfigData.DefaultProtocol`* is invalid.	Call `Configure()` when *`Ip6ConfigData.DefaultProtocol`* is invalid, the return status should be `EFI_INVALID_PARAMETER`.
5.25.5.4.10	0x64e2f4e1, 0x4431, 0x490a, 0xa0, 0x2f, 0xe3, 0xb4, 0x0c, 0x80, 0x12, 0xbb	`EFI_IP6 PROTOCOL.Configure() - Configure()` returns `EFI_INVALID_PARAMETER` when *`Ip6ConfigData.DefaultProtocol`* is invalid.	Call `Configure()` when *`Ip6ConfigData.DefaultProtocol`* is invalid, the return status should be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.25.5.4.11	0x1224d773, 0x44fb, 0x44db, 0xba, 0xb5, 0x63, 0x75, 0x5d, 0x11, 0x20, 0xdb	EFI_IP6_PROTOCOL.Configure() - Configure() returns EFI_INVALID_PARAMETER when *Ip6ConfigData.DefaultProtocol* is invalid.	Call Configure() when *Ip6ConfigData.DefaultProtocol* is invalid, the return status should be EFI_INVALID_PARAMETER.
5.25.5.4.12	0xf380d0c6, 0x2b60, 0x4674, 0xa8, 0xec, 0x94, 0x8c, 0x21, 0xbd, 0xc7, 0xd7	EFI_IP6_PROTOCOL.Configure() - Configure() returns EFI_ALREADY_STARTED with valid *Ip6ConfigData* which isn't NULL but the instance has been configured.	Call Configure() with valid *Ip6ConfigData* which isn't NULL when the instance has been configured, the returns status should be EFI_ALREADY_STARTED.
5.25.5.4.13	0x217fe9de, 0x908c, 0x4eb8, 0xac, 0xaa, 0x74, 0x96, 0x23, 0xf5, 0x25, 0x98	EFI_IP6_PROTOCOL.Configure() - Configure() returns EFI_SUCCESS with valid parameters.	5.25.5.4.13 to 5.25.5.4.16 belong to one case. 1. Call Configure() with valid parameters; the returns status should be EFI_SUCCESS.
5.25.5.4.14	0xc53003dd, 0xd76d, 0x47ca, 0xae, 0x09, 0x1a, 0xed, 0x49, 0x00, 0xc6, 0x9c	EFI_IP6_PROTOCOL.GetModeData() - GetModeData() returns EFI_SUCCESS with valid parameters.	2. Call GetModeData() with valid parameters after the child configured, the returns status should be EFI_SUCCESS.
5.25.5.4.15	0x48f68c63, 0x4860, 0x4993, 0x8f, 0xc2, 0x1b, 0x73, 0x28, 0x21, 0xcb, 0x22	Validate the *IP6ModeData.ConfigData*.	3. Validate the *IP6ModeData.ConfigData*. The *IP6ModeData.ConfigData* should be the same as the data which have been configured before. The returns status should be EFI_SUCCESS.
5.25.5.4.16	0x8287365d, 0x46e5, 0x406b, 0x98, 0x2c, 0x75, 0xdc, 0x39, 0x99, 0xd7, 0x5b	Validate the *IP6ModeData.IsConfiged*.	4. Call Configure() with NULL and then Call GetModeData() with valid parameters, and validate the *IP6ModeData.IsConfiged*. It should be FALSE.

21.5.5 Groups()

Number	GUID	Assertion	Test Description
5.25.5.5.1	0x756d489b, 0x1d6d, 0x4ab5, 0x99, 0x72, 0xd1, 0x96, 0x4a, 0x7b, 0x28, 0x0f	EFI_IP6_PROTOCOL.Groups() - Groups() returns EFI_NOT_STARTED with a not configured *ChildHandle*.	Call Groups() with a not configured *ChildHandle*; the return status should be EFI_NOT_STARTED.
5.25.5.5.2	0x2c1abd64, 0x7657, 0x4f78, 0x9f, 0x2c, 0xfa, 0x48, 0xf2, 0xd7, 0xbb, 0x66	EFI_IP6_PROTOCOL.Groups() - Groups() returns EFI_INVALID_PARAMETER when *JoinFlag* is TRUE and *GroupAddress* is NULL	Call Groups() when *JoinFlag* is TRUE and *GroupAddress* is NULL.The return status should be EFI_INVALID_PARAMETER.
5.25.5.5.3	0x6053a2b7, 0x391a, 0x4b46, 0xa7, 0x34, 0x1e, 0x2e, 0x86, 0x5c, 0x39, 0x82	EFI_IP6_PROTOCOL.Groups() - Groups() returns EFI_INVALID_PARAMETER when *GroupAddress* is not NULL and *GroupAddress* is not a multicast IPv6 address.	Call Groups() when *GroupAddress* is not NULL and *GroupAddress* is not a multicast IPv6 address. The returned status should be EFI_INVALID_PARAMETER.
5.25.5.5.4	0x1644ec0d, 0x4ef0, 0x42b8, 0xad, 0x6b, 0x8b, 0xbd, 0xd5, 0x3f, 0x84, 0x1d	EFI_IP6_PROTOCOL.Groups() - Groups() returns EFI_ALREADY_STARTED when *JoinFlag* is TRUE and *GroupAddress* is in the group table.	Call Groups() when *JoinFlag* is TRUE and *GroupAddress* is in the group table, the return status should be EFI_ALREADY_STARTED.
5.25.5.5.5	0xc1fe68df, 0xca52, 0x42c4, 0xbe, 0xd4, 0xc0, 0x34, 0xf9, 0xf0, 0x03, 0x18	EFI_IP6_PROTOCOL.Groups() - Groups() returns EFI_NOT_FOUND when *JoinFlag* is FALSE and *GroupAddress* is not in the group table.	Call Groups() when *JoinFlag* is FALSE and *GroupAddress* is not in the group table, the return status should be EFI_NOT_FOUND.

Number	GUID	Assertion	Test Description
5.25.5.5.6	0xbf971751, 0xbc7e, 0x421a, 0x86, 0xbe, 0xda, 0x67, 0x16, 0x03, 0xb0, 0xf0	**EFI_IP6_PROTOCOL.Groups() - Groups()** returns **EFI_SUCCESS** with **TRUE** *JoinFlag* and an valid *GroupAddress*.	5.25.5.5.6 to 5.25.5.5.10 belong to one case. 1. Call **Groups()** with **TRUE** *JoinFlag* and a valid *GroupAddress*, the return status should be **EFI_SUCCESS**.
5.25.5.5.7	0x3542d69e, 0xc8eb, 0x4da6, 0x8e, 0x41, 0xdd, 0x49, 0x43, 0x17, 0xa7, 0x80	Check the *Ip6ModeData.GroupCount* field.	2. The value of *Ip6ModeData.GroupCount* should be 1.
5.25.5.5.8	0x65dafab8, 0xe505, 0x4f4a, 0xa7, 0xaf, 0x54, 0x42, 0x68, 0x42, 0xca, 0xa8	Check the *Ip6ModeData.GroupTable* field.	3. The value of *Ip6ModeData.GroupTable* should be the same as the route entry we added.
5.25.5.5.9	0x25af1861, 0x25e5, 0x4137, 0xb1, 0xb0, 0x56, 0x5f, 0xfa, 0x32, 0xee, 0x44	**EFI_IP6_PROTOCOL.Groups () - Groups()** returns **EFI_SUCCESS** with **FALSE** *JoinFlag* and and *GroupAddress* is in the group table.	4. Call **Groups()** with **FALSE** *JoinFlag* and and *GroupAddress* is in the group table, the return status should be **EFI_SUCCESS**.
5.25.5.5.10	0x882ddbc2, 0x4372, 0x41ff, 0x95, 0x5c, 0x89, 0x15, 0x56, 0x73, 0xb3, 0x5d	Check the *Ip6ModeData.GroupCount* field.	5. Call **GetModeData()** with valid parameters, the value of *Ip6ModeData.GroupCount* should be 0.

21.5.6 Routes()

Number	GUID	Assertion	Test Description
5.25.5.6.1	0xe5a50efc, 0x831b, 0x4dc1, 0x8a, 0x78, 0xb5, 0x36, 0xa2, 0x39, 0xd8, 0x8d	**EFI_IP6_PROTOCOL.Routes() - Routes()** returns **EFI_NOT_STARTED** with a not configured *ChildHandle*.	Call **Routes ()** with a not configured *ChildHandle*, the return status should be **EFI_NOT_STARTED**.
5.25.5.6.2	0x9a9fadb0, 0x6651, 0x4070, 0xac, 0x63, 0x2b, 0xa0, 0x92, 0xc5, 0xe0, 0x0b	**EFI_IP6_PROTOCOL.Routes() - Routes()** returns **EFI_INVALID_PARAMETER** when *DeleteRoute* is **TRUE**, both *Destiniation* and *GatewayAddress* are **NULL**.	Call **Routes()** when *DeleteRoute* is **TRUE**, both *Destiniation* and *GatewayAddress* are **NULL**, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.6.3	0x38dabbd5, 0x37fb, 0x4744, 0xab, 0x18, 0xac, 0xcf, 0x5d, 0x0e, 0x25, 0xf1	**EFI_IP6_PROTOCOL.Routes() - Routes()** returns **EFI_INVALID_PARAMETER** when *DeleteRoute* is **FALSE**, *Destiniation* is **NULL** and *GatewayAddress* is not **NULL**.	Call **Routes()** when ***DeleteRoute*** is **FALSE**, *Destiniation* is **NULL** and *GatewayAddress* is not **NULL**, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.6.4	0xb3ea5648, 0x9a8c, 0x4761, 0x9f, 0x9c, 0x9b, 0x44, 0x87, 0xca, 0x14, 0x0a	**EFI_IP6_PROTOCOL.Routes() - Routes()** returns **EFI_INVALID_PARAMETER** when *DeleteRoute* is **FALSE**, ***Destiniation*** is not **NULL** and *GatewayAddress* is **NULL**.	Call **Routes()** when ***DeleteRoute*** is **FALSE**, *Destiniation* is not **NULL** and *GatewayAddress* is **NULL**, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.6.5	0xef4878ab, 0x02e1, 0x4a3f, 0x9b, 0x0c, 0x0a, 0xea, 0x7d, 0x25, 0xf2, 0x46	**EFI_IP6_PROTOCOL.Routes() - Routes()** returns **EFI_INVALID_PARAMETER** when *GatewayAddress* is not a valid unicast IPv6 address.	Call **Routes()** when *GatewayAddress* is not a valid unicast IPv6 address, the return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.25.5.6.6	0x67ab6941, 0xfe7d, 0x4046, 0x9f, 0xc4, 0x61, 0x6c, 0x50, 0xb9, 0xd3, 0x72	**EFI_IP6_PROTOCOL.Routes() - Routes()** returns **EFI_INVALID_PARAMETER** when *GatewayAddress* is one of configured local IPv6 addresses.	Call **Routes()** when *GatewayAddress* is one of configured local IPv6 addresses, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.6.7	0x2359c3c5, 0x5789, 0x4c12, 0xbc, 0x1c, 0x5b, 0x94, 0x18, 0x5d, 0x24, 0x39	**EFI_IP6_PROTOCOL.Routes() - Routes()** returns **EFI_NOT_FOUND** when *DeleteRoute* is **TRUE** and this entry is not in current routing table.	Call **Routes()** when *DeleteRoute* is **TRUE** and this entry is not in current routing table, the return status should be **EFI_NOT_FOUND**.
5.25.5.6.8	0x9c9e4191, 0xbd67, 0x42d7, 0x8e, 0x64, 0x22, 0xe4, 0xc3, 0x4b, 0x8c, 0x2e	**EFI_IP6_PROTOCOL.Routes() - Routes()** returns **EFI_ACCESS_DENIED** when ***DeleteRoute*** is **FALSE** and the entry is already in current routing table.	Call **Routes()** when *DeleteRoute* is **FALSE** and the entry is already in current routing table, the return status should be **EFI_ACCESS_DENIED**.
5.25.5.6.9	0x576be5b1, 0xc50e, 0x44d3, 0x80, 0x99, 0xa0, 0x67, 0x56, 0x0b, 0x24, 0x10	**EFI_IP6_PROTOCOL.Routes() - Routes()** returns **EFI_SUCCESS** with valid parameters.	5.25.5.6.9 to 5.25.5.6.13 belong to one case. 1. Call **Routes()** with valid parameters, the return status should be **EFI_SUCCESS**.
5.25.5.6.10	0x8c3d2c17, 0xc282, 0x4daa, 0x96, 0xfb, 0x1d, 0x1c, 0xdc, 0xd2, 0x9f, 0x99	Check *Ip6ModeData.RouteCount* field	2. The value of *Ip6ModeData.RouteCount* should more than zero.
5.25.5.6.11	0xb7cc7815, 0x7a38, 0x4904, 0xb2, 0x4d, 0x22, 0x09, 0x00, 0xb5, 0xf7, 0xcc	Check *Ip6ModeData.RouteTable* field.	3. *Ip6ModeData.RouteTable* should contain the route we added before.
5.25.5.6.12	0x709e8127, 0x1a36, 0x4c08, 0xac, 0x22, 0xd1, 0xb5, 0x0f, 0x82, 0x5a, 0x14	**EFI_IP6_PROTOCOL.Routes() - Routes()** returns **EFI_SUCCESS** with valid parameter .	4. Call **Routes()** with valid parameters to delete the route we added before, the return status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.25.5.6.13	0xe30d8352, 0x4f0c, 0x43fe, 0xb2, 0x0e, 0xcf, 0xeb, 0xfb, 0x45, 0xb4, 0x42	Check `Ip6ModeData.RouteCount` field.	5. The value of `Ip6ModeData.RouteCount` should be decreased by 1.

21.5.7 Neighbors()

Number	GUID	Assertion	Test Description
5.25.5.7.1	0x4f6a49b0, 0xff4f, 0x4ba8, 0xa6, 0x31, 0x94, 0x8d, 0x23, 0xbc, 0x15, 0x00	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_NOT_STARTED** with a not configured *ChildHandle*.	Call **Neighbors()** with a not configured *ChildHandle*, the return status should be **EFI_NOT_STARTED**.
5.25.5.7.2	0x35ffe726, 0x0b87, 0x480e, 0xa2, 0xeb, 0x1c, 0x7d, 0xed, 0x16, 0x99, 0x4e	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_INVALID_PARAMETER** when *TargetIp6Address* is **NULL**.	Call **Neighbors()** when *TargetIp6Address* is **NULL**, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.7.3	0x3360d9f1, 0x674a, 0x445f, 0xab, 0x8a, 0x3b, 0xca, 0xde, 0xae, 0xed, 0x2b	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_INVALID_PARAMETER** when *TargetLinkAddress* is **NULL** and *DeleteFlag* is **TRUE**.	Call **Neighbors()** when *TargetLinkAddress* is **NULL** and *DeleteFlag* is **TRUE**, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.7.4	0xc0556979, 0x5ab6, 0x4c65, 0xb6, 0x49, 0xc7, 0xbe, 0x34, 0x9f, 0x04, 0xed	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_INVALID_PARAMETER** when *TargetLinkAddress* is invalid.	Call **Neighbors()** when *TargetLinkAddress* is invalid, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.7.5	0x98c0eda5, 0xf1b5, 0x4bf3, 0xa1, 0x58, 0xbb, 0x68, 0xdc, 0xe3, 0xb4, 0x5c	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_INVALID_PARAMETER** when *TargetIpAddress* is not a valid unicast Ipv6 Address.	Call **Neighbors()** when *TargetIpAddress* is not a valid unicast Ipv6 Address, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.7.6	0xe60636fa, 0x47f1, 0x433e, 0xa0, 0x79, 0x50, 0x92, 0xcf, 0x59, 0x0b, 0xb1	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_INVALID_PARAMETER** when *TargetIpAddress* is one of configured local Ipv6 address.	Call **Neighbors()** when *TargetIpAddress* is one of configured local Ipv6 address, the return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.25.5.7.7	0xd88a65be, 0x37ff, 0x41e2, 0xa8, 0xbd, 0x3e, 0x92, 0x1b, 0xf5, 0x89, 0x87	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_NOT_FOUND** when *DeleteFlag* is **TRUE** and this entry isn't in current neighbor cache.	Call **Neighbors()** when *DeleteFlag* is **TRUE** and this entry isn't in current neighbor cache, the return status should be **EFI_NOT_FOUND**.
5.25.5.7.8	0x7a528a8e, 0x1339, 0x4618, 0x92, 0x9e, 0xf5, 0x60, 0xb6, 0xd1, 0x98, 0xd0	**EFI_IP6 PROTOCOL.Neighbors() - Neighbors()** returns **EFI_ACCESS_DENIED** when *DeleteFlag* is **FALSE** and this entry isn't in current neighbor cache.	Call **Neighbors()** when *DeleteFlag* is **FALSE** and this entry isn't in current neighbor cache, the return status should be **EFI_ ACCESS_DENIED**.
5.25.5.7.9	0xb0c66678, 0x6552, 0x42f7, 0xa4, 0x5a, 0x36, 0x3d, 0xde, 0xa5, 0x75, 0xbd	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_NOT_FOUND** when *DeleteFlag* is **FALSE** and the ***TargetLinkAddress*** is **NULL**.	Call **Neighbors()** when ***DeleteFlag*** is **FALSE** and the *TargetLinkAddress* is **NULL**, the return status should be **EFI_NOT_FOUND**.
5.25.5.7.10	0xf339086f, 0xd826, 0x48b4, 0xbf, 0x77, 0xd7, 0x71, 0xba, 0xb6, 0x28, 0xb5	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_SUCCESS** with valid parameters.	5.25.5.7.10 to 5.25.5.7.15 belong to one case 1. Call **Neighbors()** with valid parameters to add a neighbor cache, the return status should be **EFI_SUCCESS**.
5.25.5.7.11	0xa5389777, 0xd3d2, 0x41da, 0xa7, 0x22, 0xbf, 0xbe, 0xe2, 0xc8, 0x78, 0x4e	Check *Ip6ModeData.NeighborCount* field.	2. The value of *Ip6ModeData.NeighborCount* should be 1.
5.25.5.7.12	0x179fa1e4, 0xa408, 0x481d, 0xbb, 0x3a, 0x72, 0x81, 0x2e, 0xcd, 0x2a, 0xde	Check *Ip6ModeData.NeighborsCache.Neighbor* field.	3. The value of *Ip6ModeData.NeighborsCache.Neighbor* should be the same as we added.

Number	GUID	Assertion	Test Description
5.25.5.7.13	0x6991227c, 0x3562, 0x4875, 0x82, 0x2e, 0x7d, 0xe3, 0xf3, 0xcf, 0x90, 0x59	Check *Ip6ModeData.NeighborsCache.LinkAddress* field.	4. The value of *Ip6ModeData.NeighborsCache.LinkAddress* should be the same as we added.
5.25.5.7.14	0x823ca277, 0xdaa3, 0x4917, 0xa2, 0x58, 0xc9, 0xe3, 0x30, 0xef, 0xb6, 0xd1	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_SUCCESS** with valid parameters.	5. Call **Neighbors()** with valid parameters to delete a neighbor cache, the return status should be **EFI_SUCCESS**.
5.25.5.7.15	0x971bf190, 0x49c5, 0x4b5b, 0x83, 0x20, 0x0c, 0x74, 0xc3, 0x5c, 0xc9, 0x91	Check *Ip6ModeData.NeighborCount* field.	6. The value of *Ip6ModeData.NeighborCount* should be 0 after delete.
5.25.5.7.16	0x0379e4c1, 0x2b4f, 0x41e2, 0xb6, 0x44, 0xda, 0xf5, 0x4a, 0x53, 0xd9, 0xdd	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_SUCCESS** with valid parameters.	5.25.5.7.16 to 5.25.5.7.22 belong to one case 1. Call **Neighbors()** with valid parameters to add a neighbor cache, the return status should be **EFI_SUCCESS.**
5.25.5.7.17	0xeb7f4f6f, 0x521e, 0x452c, 0xbc, 0x6e, 0xdf, 0xbf, 0xb9, 0x22, 0x2e, 0x3b	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_SUCCESS** with valid parameters.	2. Call **Neighbors()** with valid parameters to update a neighbor cache, the return status should be **EFI_SUCCESS.**
5.25.5.7.18	0x53567ad3, 0x2cfe, 0x4bfd, 0xba, 0x97, 0xea, 0xca, 0xad, 0xdd, 0x2d, 0x00	Check *Ip6ModeData.NeighborCount* field.	3. The value of *Ip6ModeData.NeighborCount* should be 1 after added**.**
5.25.5.7.19	0x6be12cd9, 0xcdf7, 0x4b0c, 0x82, 0xb5, 0x5b, 0xee, 0x3c, 0xfd, 0x52, 0xe8	Check *Ip6ModeData.NeighborsCache.Neighbor* field.	4. The value of *Ip6ModeData.NeighborsCache.Neighbor* should be the same as we added.

Number	GUID	Assertion	Test Description
5.25.5.7.20	0x8dfbc45e, 0x5b6d, 0x4c1d, 0x9c, 0x0a, 0x2f, 0xcc, 0xb6, 0x1e, 0xeb, 0xfa	Check *Ip6ModeData.NeighborsCache.LinkAddress* field.	5. The value of *Ip6ModeData.NeighborsCache.LinkAddress* should be the same as we added.
5.25.5.7.21	0xe9aa5a6e, 0x9b98, 0x4e3d, 0xa2, 0xc1, 0x49, 0x31, 0x85, 0x14, 0x72, 0xde	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_SUCCESS** with valid parameters.	6. Call **Neighbors()** with valid parameters to delete a neighbor cache, the return status should be **EFI_SUCCESS.**
5.25.5.7.22	0x2d82ca70, 0xc383, 0x458e, 0x93, 0x1d, 0x84, 0xfd, 0x2b, 0xb2, 0x7c, 0xfd	Check *Ip6ModeData.NeighborCount* field.	7. The value of *Ip6ModeData.NeighborCount* should be 0 after deleted.
5.25.5.7.23	0x5646fc4f, 0x06cb, 0x49ba, 0xbe, 0xb0, 0x3d, 0xf0, 0xde, 0x02, 0xda, 0xbf	**EFI_IP6_PROTOCOL.Neighbors() - Neighbors()** returns **EFI_SUCCESS** with valid parameters.	5.25.5.7.23 to 5.25.5.7.27 belong to the same case 1. Call **Neighbors()** with valid parameters to add a neighbor cache, the return status should be **EFI_SUCCESS.**
5.25.5.7.24	0x4baa627a, 0x0019, 0x4eda, 0xbd, 0x27, 0xbb, 0xd2, 0xdd, 0x5f, 0x9f, 0x19	Check *Ip6ModeData.NeighborCount* field.	2. The value of *Ip6ModeData.NeighborCount* should be 1 after added.
5.25.5.7.25	0xa93cf6a1, 0x3548, 0x41e8, 0x94, 0xdc, 0x07, 0xe8, 0x30, 0x72, 0x34, 0xd5	Check *Ip6ModeData.NeighborsCache.Neighbor* field.	The value of *Ip6ModeData.NeighborsCache.Neighbor* should be the same as we added.
5.25.5.7.26	0xe0297637, 0x7b3d, 0x4894, 0x80, 0x8d, 0x2c, 0x7d, 0x64, 0xa9, 0x19, 0x46	Check *Ip6ModeData.NeighborsCache.LinkAddress* field.	The value of *Ip6ModeData.NeighborsCache.LinkAddress* should be the same as we added.

Number	GUID	Assertion	Test Description
5.25.5.7.27	0xa03dc0e3, 0xffe3, 0x4bff, 0x82, 0x9f, 0xb0, 0x99, 0xb3, 0xe2, 0x57, 0x64	Check *Ip6ModeData.NeighborCount* field.	The value of *Ip6ModeData.NeighborCount* should be 0 after time out.

21.5.8 Transmit()

Number	GUID	Assertion	Test Description
5.25.5.8.1	0x255fe450, 0xc537, 0x4b0a, 0xbe, 0x80, 0xc8, 0x73, 0x95, 0x66, 0x26, 0x16	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_NOT_STARTED** with a not configured *ChildHandle*.	Call **Transmit()** with a not configured *ChildHandle*, the return status should be **EFI_NOT_STARTED**.
5.25.5.8.2	0x8347ebcd, 0x4f16, 0x4bfd, 0x83, 0xf6, 0x0f, 0x8a, 0xdc, 0x6a, 0x89, 0x2e	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with a **NULL** *Token*.	Call **Transmit()** with a **NULL** *Token*, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.8.3	0xc7cf4815, 0x9c64, 0x4074, 0x94, 0x3f, 0xf5, 0x6d, 0x2e, 0x9d, 0x79, 0x5d	**EFI_IP6 PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with a **NULL** *Token->Event*.	Call **Transmit()** with a **NULL** *Token->Event*, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.8.4	0x2ccfe480, 0x452c, 0x4706, 0x88, 0x69, 0x97, 0xb7, 0x7b, 0x03, 0xa9, 0x26	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with a **NULL** *Token->Packet.TxData*.	Call **Transmit()** with a **NULL** *Token->Packet.TxData*, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.8.5	0xede110b2, 0x8455, 0x4ec8, 0xbb, 0x22, 0x19, 0x94, 0x59, 0x54, 0x11, 0x46	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** when *Token->Packet.TxData->ExtHdrs* is **NULL**.	Call **Transmit()** when *Token->Packet.TxData->ExtHdrs* is **NULL**, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.8.6	0xd4f4a746, 0xaff3, 0x4490, 0xa6, 0xd9, 0xef, 0x38, 0x06, 0x69, 0x0a, 0x94	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** when *Token->Packet.TxData->FragmentCount* is Zero.	Call **Transmit()** when *Token->Packet.TxData->FragmentCount* is Zero, the return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.25.5.8.7	0xa2dc1ca1, 0x37ef, 0x4147, 0xa6, 0x90, 0x4d, 0x4e, 0xd1, 0x4c, 0x99, 0xf9	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** when *Token->Packet.TxData->FragmentTable[0].FragmentLength* is Zero.	Call **Transmit()** when *Token->Packet.TxData->FragmentTable[0].FragmentLength* is Zero, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.8.8	0xef828012, 0xdeda, 0x4f91, 0xb1, 0x10, 0x38, 0x26, 0x92, 0x50, 0xf3, 0xc8	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with a **NULL** *Token->Packet.TxData->FragmentTable[0].FragmentBuffer*.	Call **Transmit()** with a **NULL** *Token->Packet.TxData->FragmentTable[0].FragmentBuffer*, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.8.9	0x8db7ffb3, 0x47fb, 0x4281, 0x97, 0xa5, 0x8a, 0xa7, 0xe1, 0x98, 0x87, 0x72	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** *Token->Packet.TxData->DataLength* is zero.	Call **Transmit()** when *Token->Packet.TxData->DataLength* is zero, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.8.10	0x63c9939b, 0x7aa6, 0x4565, 0xab, 0x11, 0xdc, 0x13, 0x32, 0x38, 0x1b, 0x32	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with an invalid *Token->Packet.TxData->DataLength* which is not equal to the sum of fragments length.	Call **Transmit()** with an invalid *Token->Packet.TxData->DataLength* which is not equal to the sum of the fragments length, the return status should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.25.5.8.11	0x220f2e8c, 0xae0c, 0x4f9c, 0x89, 0x1b, 0x74, 0x54, 0xaa, 0x63, 0xf0, 0xce	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with a non-zero *Token->Packet.TxData->Udp6sessionData->DestinationAddress* which is not specified in configure process.	Call **Transmit()** with a non-zero *Token->Packet.TxData->Udp6sessionData->DestinationAddress* which is not specified in configure process, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.8.12	0xc7353218, 0xc96e, 0x4236, 0x92, 0x53, 0x86, 0x85, 0x41, 0x0a, 0x47, 0x0c	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with a zero *Token->Packet.TxData->Udp6sessionData->DestinationAddress* when *DestinationAddress* is unspecified when doing configure process.	Call **Transmit()** with a zero *Token->Packet.TxData->Udp6sessionData->DestinationAddress* when *DestinationAddress* is unspecified when doing configure process, the return status should be **EFI_INVALID_PARAMETER**.
5.25.5.8.13	0x2ac52cba, 0xbe4e, 0x4c9e, 0xae, 0xe5, 0x4d, 0x10, 0x6b, 0x95, 0x1b, 0xc4	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_ACCESS_DENIED** with a *Token->Event* which has already been in the transmit queue.	Call **Transmit()** with a *Token->Event* which has already been in the transmit queue, the return status should be **EFI_ACCESS_DENIED**.
5.25.5.8.14	0xfeaa4963, 0x24c0, 0x477a, 0x8a, 0xc7, 0xa9, 0xac, 0xe5, 0xbb, 0xf4, 0x53	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_NOT_FOUND** with no route entry to the destination.	Call **Transmit()** with no route entry for the destination, the return status should be **EFI_NOT_FOUND**.

Number	GUID	Assertion	Test Description
5.25.5.8.15	0xda08e7a1, 0x7ab6, 0x4b23, 0x9b, 0xb6, 0x27, 0xae, 0x0a, 0xb7, 0xb6, 0xc3	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_BAD_BUFFER_SIZE** with a *Token->Packet.TxData->DataLength* which beyond the maximum udp6 packet size.	Call **Transmit()** with a *Token->Packet.TxData->DataLength* which beyond the maximum udp6 packet size, the return status should be **EFI_BAD_BUFFER_SIZE**.
5.25.5.8.17	0x4660050c, 0x749c, 0x428f, 0xa5, 0xd9, 0x9a, 0x4c, 0x8e, 0xa4, 0x20, 0xe5	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_SUCCESS** with valid parameters.	5.25.5.8.17 to 5.25.5.8.21 belong to one case. 1. Call **Transmit()** with valid parameters, the return status should be **EFI_SUCCESS**.
5.25.5.8.18	0xb67c0483, 0x7b89, 0x446c, 0xac, 0xba, 0x17, 0xb8, 0x7f, 0x4e, 0xcb, 0x5f	*Token->Event* should be signaled.	2. *Token->Event* should be signaled.
5.25.5.8.19	0x9a61d143, 0x7ddf, 0x4d4e, 0xa7, 0x97, 0x5f, 0xfc, 0x85, 0x09, 0x0e, 0xb4	*Token->Status* should be **EFI_SUCCESS**.	3. *Token->Status* should be **EFI_SUCCESS**.
5.25.5.8.20	0x8916816a, 0x6876, 0x4e76, 0xa2, 0xc2, 0x3d, 0xc6,0x3f, 0xcd, 0x00, 0x7a	The packet should be received by the other side.	4. The packet should be received by the other side.
5.25.5.8.21	0x088ed948, 0x0276, 0x4bb4, 0x98, 0x96, 0xe3, 0xa7, 0x67, 0x21, 0x74, 0x2f	The received packet content should be reasonable.	5. The received packet content should be reasonable.
5.25.5.8.22	0x3cf5b8eb, 0xc742, 0x4d34, 0x97, 0x65, 0xf8, 0xcc, 0x32, 0x49, 0x4e, 0x92	**EFI_IP6_PROTOCOL.Transmit() - Transmit()** returns **EFI_SUCCESS** with valid parameters.	5.25.5.8.22 to 5.25.5.8.28 belong to one case. 1. Call **Transmit()** with valid parameters, the return status should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.25.5.8.23	0x8f8f115e, 0xd436, 0x41a1, 0xaa, 0x42, 0x11, 0xe7, 0x04, 0xe0, 0x29, 0x11	`Token->Event` should be signaled.	2. `Token->Event` should be signaled.
5.25.5.8.24	0x612b38d1, 0x37cb, 0x419d, 0x8d, 0xfe, 0x44, 0xc7, 0x35, 0xef, 0xe0, 0x17	`Token->Status` should be **`EFI_SUCCESS`**.	3. `Token->Status` should be **`EFI_SUCCESS`**.
5.25.5.8.25	0x464f35de, 0xd546, 0x4140, 0xa7, 0x5e, 0x23, 0xfd, 0xa1, 0xce, 0x2a, 0xd5	The packet should be received by the other side.	4. The packet should be received by the other side.
5.25.5.8.26	0x0c8799bb, 0xeb02, 0x4172, 0x97, 0xe5, 0xec, 0x6b, 0xaf, 0xe6, 0xe5, 0xa6	The first fragment of received packet content should be reasonable.	5. The first fragment of received packet content should be reasonable.
5.25.5.8.27	0xe3ececa3, 0x8f49, 0x4bb9, 0xb0, 0xc9, 0x55, 0x85, 0x00, 0x28, 0xc3, 0x1a	The second fragment of received packet content should be reasonable.	6. The second fragment of received packet content should be reasonable.
5.25.5.8.28	0xcf73acd9, 0x0893, 0x4b22, 0x88, 0xcf, 0x42, 0x98, 0x22, 0x0e, 0xc0, 0x6c	Total length should be the sum of two fragment length.	7. Total length should be the sum of two fragment length.

21.5.9 Receive()

Number	GUID	Assertion	Test Description
5.25.5.9.1	0xa1ca863c, 0x8c68, 0x4afc, 0x8a, 0x97, 0xff, 0x60, 0x3e, 0xef, 0xb4, 0xc9	EFI_IP6_PROTOCOL.Receive() - Receive() returns EFI_NOT_STARTED with a not configured *ChildHandle*.	Call Receive() with a not configured *ChildHandle*, the return status should be EFI_NOT_STARTED.
5.25.5.9.2	0xa9231505, 0xf3ec, 0x462e, 0xb7, 0x0b, 0x14, 0xb2, 0xc6, 0xa2, 0x23, 0xd8	EFI_IP6_PROTOCOL.Receive() - Receive() returns EFI_INVALID_PARAMETER with a NULL *Token*.	Call Receive() with a NULL *Token*, the return status should be EFI_INVALID_PARAMETER.
5.25.5.9.3	0xdf7d75d2, 0x4288, 0x4a50, 0xa5, 0xdf, 0x01, 0x85, 0x98, 0x74, 0xb8, 0x29	EFI_IP6_PROTOCOL.Receive() - Receive() returns EFI_INVALID_PARAMETER with a NULL *Token->Event*.	Call Receive() with a NULL *Token->Event*, the return status should be EFI_INVALID_PARAMETER.
5.25.5.9.4	0x1bbc8695, 0x6552, 0x422d, 0xb1, 0x32, 0xda, 0x58, 0x03, 0x0e, 0xf5, 0xb6	EFI_IP6_PROTOCOL.Receive() - Receive() returns EFI_ACCESS_DENIED with a *Token->Event* which has already been in the receive queue.	Call Receive() with a *Token->Event* which has already been in the receive queue, the return status should be EFI_ACCESS_DENIED.
5.25.5.9.5	0x5b0a58f2, 0x6668, 0x4247, 0xae, 0x25, 0xae, 0x7e, 0x24, 0x75, 0x02, 0xd7	EFI_IP6_PROTOCOL.Receive() - Receive() returns EFI_SUCCESS with valid parameters.	5.25.5.9.5 to 5.25.5.9.11 belong to one case. 1. Call Receive() with valid parameters, the return status should be EFI_SUCCESS.
5.25.5.9.6	0x019b2b66, 0xfbce, 0x4cab, 0xab, 0x09, 0xd8, 0xdd, 0x34, 0x70, 0x4e, 0xe9	*Token->Event* should be signaled.	2. *Token->Event* should be signaled.
5.25.5.9.7	0x5750bf3b, 0xcead, 0x49a9, 0xad, 0x33, 0xb4, 0x6e, 0x85, 0xc9, 0x78, 0xea	*Token->Status* should be EFI_SUCCESS.	3. *Token->Status* should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.25.5.9.8	0x155874a6, 0x0dc9, 0x4b67, 0x9d, 0xb7, 0xda, 0xc9, 0x24, 0xad, 0xc4, 0x4a	Check IPv6 `Headlength`.	4. The value of IPv6 `Headlength` should be 40.
5.25.5.9.9	0x7f6044dc, 0x1767, 0x48fc, 0x8a, 0x24, 0xa5, 0x85, 0x6e, 0x82, 0x8e, 0x94	Check IPv6 `RxData.Datalength.`	5. `RxData.Datalength` should be the same as we expected.
5.25.5.9.10	0x022b38cd, 0x5928, 0x4c36, 0x98, 0xd4, 0xd3, 0x67, 0xef, 0x04, 0x55, 0xc7	`RxData.FragmentCount` should be 1.	6. `RxData.FragmentCount` should be 1.
5.25.5.9.11	0x4b71edc9, 0x9c61, 0x45b2, 0xa5, 0x02, 0x05, 0x3a, 0x97, 0x71, 0x19, 0xf3	The content of Ipv6 header should be the same as we expected.	7. The content of Ipv6 header should be the same as we expected.
5.25.5.9.12	0x48cbff74, 0x89a1, 0x4021, 0xa5, 0x81, 0x40, 0xc1, 0x56, 0xc7, 0x2f, 0x36	**`EFI_IP6_PROTOCOL.Receive() – Receive()`** returns **`EFI_SUCCESS`** with valid parameters.	5.25.5.9.12 to 5.25.5.9.18 belong to one case 1. Call **`Receive()`** with valid parameters, the return status should be **`EFI_SUCCESS`**.
5.25.5.9.13	0xa433bb6d, 0x152c, 0x4de8, 0xa6, 0x01, 0x95, 0x31, 0x4d, 0xc3, 0x08, 0xd1	`Token->Event` should be signaled.	2. `Token->Event` should be signaled.
5.25.5.9.14	0x0011751a, 0x87f4, 0x4572, 0xad, 0x75, 0xa5, 0x13, 0x84, 0xbf, 0x01, 0x0a	`Token->Status` should be **`EFI_SUCCESS`**.	3. `Token->Status` should be **`EFI_SUCCESS`**.

Number	GUID	Assertion	Test Description
5.25.5.9.15	0xa2d00870, 0xe59f, 0x4b55, 0xbe, 0x36, 0xda, 0x81, 0x15, 0xe4, 0x57, 0x41	Check IPv6 `Headlength`.	4. The value of IPv6 `Headlength` should be 40.
5.25.5.9.16	0x99aef759, 0xcd2e, 0x46bd, 0x8d, 0x8a, 0x6c, 0xe7, 0x90, 0x8a, 0xf9, 0xa0	Check IPv6 `RxData.Datalength`.	5. `RxData.Datalength` should be the same as we expected.
5.25.5.9.17	0x1f01211f, 0x1c55, 0x4ee8, 0xb5, 0xe7, 0x14, 0x72, 0xcd, 0xf7, 0x60, 0x64	`RxData.FragmentCount` should be 2.	6. `RxData.FragmentCount` should be 1.
5.25.5.9.18	0x72f6a9fd, 0xb4bf, 0x47f2, 0x85, 0x07, 0x37, 0x99, 0x02, 0x2f, 0x06, 0xea	The content of Ipv6 header should be the same as we expected.	7. The content of Ipv6 header should be the same as we expected.

21.5.10 Cancel()

Number	GUID	Assertion	Test Description
5.25.5.10.1	0x136f34b0, 0x4806, 0x4150, 0x98, 0x3c, 0x0c, 0x54, 0x1d, 0x7e, 0x8e, 0x2f	EFI_IP6_PROTOCOL.Cancel() – Cancel() returns EFI_NOT_STARTED with a not configured *ChildHandle*.	Call Cancel() with a Receive Token and a not configured *ChildHandle*, the return status should be EFI_NOT_STARTED.
5.25.5.10.2	0x9c7cacd0, 0xcb07, 0x4181, 0x93, 0x80, 0x90, 0x12, 0xbb, 0x60, 0xe6, 0xe3	EFI_IP6_PROTOCOL.Cancel() – Cancel() returns EFI_NOT_STARTED with a not configured *ChildHandle*.	Call Cancel() with a Transmit Token and a not configured *ChildHandle*, the return status should be EFI_NOT_STARTED.
5.25.5.10.3	0x5e2ebb02, 0xe419, 0x4ed4, 0xa7, 0xd3, 0xa3, 0xa7, 0xba, 0xb4, 0xee, 0x46	EFI_IP6_PROTOCOL.Cancel() – Cancel() returns EFI_NOT_FOUND with a *Token* which hasn't been inserted into receive queue.	Call Cancel() with a *Token* which hasn't been inserted into receive queue, the return status should be EFI_NOT_FOUND.
5.25.5.10.4	0x7ceb17ac, 0x03bf, 0x427e, 0xbe, 0xe6, 0x98, 0x7f, 0xda, 0x4f, 0x5c, 0x36	EFI_IP6_PROTOCOL.Cancel() – Cancel() returns EFI_NOT_FOUND with a *Token* which hasn't been inserted into transmit queue.	Call Cancel() with a *Token* which hasn't been inserted into transmit queue, the return status should be EFI_NOT_FOUND.
5.25.5.10.5	0x02c484a9, 0x86aa, 0x4484, 0x91, 0xa5, 0x50, 0x0f, 0xd7, 0x0c, 0x3c, 0x84	EFI_IP6_PROTOCOL.Cancel() – Cancel() returns EFI_NOT_FOUND with a *Token* which has been removed from receive queue.	Call Cancel() with a *Token* which has been removed from receive queue, the return status should be EFI_NOT_FOUND.
5.25.5.10.6	0xf1955578, 0x07ba, 0x4119, 0xbe, 0xa2, 0xe0, 0xb1, 0x2b, 0x41, 0x77, 0x59	EFI_IP6_PROTOCOL.Cancel() – Cancel() returns EFI_NOT_FOUND with a *Token* which has been removed from transmit queue.	Call Cancel() with a *Token* which has been removed from transmit queue, the return status should be EFI_NOT_FOUND.

Number	GUID	Assertion	Test Description
5.25.5.10.7	0xdb1f8413, 0x7d91, 0x4366, 0x94, 0xe7, 0x96, 0xec, 0xf9, 0xd6, 0x0e, 0xbb	**EFI_IP6_PROTOCOL.Receive() – Receive()** returns **EFI_SUCCESS** with valid parameters.	5.25.5.10.7 to 5.25.5.10.10 belong to one case. 1. Call **Receive()** with valid parameters, the return status should be **EFI_SUCCESS**.
5.25.5.10.8	0xb5c49851, 0x0ea9, 0x4d1c, 0x9a, 0xbd, 0x98, 0x5f, 0x94, 0x98, 0x32, 0xf1	**EFI_IP6_PROTOCOL.Cancel() – Cancel()** returns **EFI_SUCCESS** with valid parameters.	2. Call **Cancel()** with valid parameters, the return status should be **EFI_SUCCESS**.
5.25.5.10.9	0xff8a1c8f, 0xdf30, 0x4e95, 0xbf, 0x98, 0x11, 0x46, 0xc0, 0xa3, 0xec, 0x50	*Token->Status* should be **EFI_ABORTED**.	*Token->Status* should be **EFI_ABORTED**.
5.25.5.10.10	0x53bb7192, 0xe93a, 0x4a4b, 0xba, 0x2f, 0x58, 0x26, 0x6c, 0xe9, 0xdc, 0x80	*Token->Event* should be signaled.	*Token->Event* should be signaled.

21.5.11 Poll()

Number	GUID	Assertion	Test Description
5.25.5.11.1	0xf0a862e2, 0xf222, 0x4742, 0x9e, 0x3f, 0x26, 0xa9, 0x18, 0xd6, 0x9e, 0xf1	**EFI_IP6_PROTOCOL.Poll() – Poll()** returns **EFI_NOT_STARTED** with a not configured *ChildHandle*.	Call **Poll()** with a not configured *ChildHandle*, the return status should be **EFI_NOT_STARTED**.
5.25.5.11.2	0x6ee2f2aa, 0x0a9f, 0x4690, 0xa5, 0x42, 0x95, 0x02, 0x1e, 0x5e, 0xd8, 0xbf	**EFI_IP6_PROTOCOL.Poll() – Poll()** returns **EFI_NOT_READY** with no income and outcome packets.	Call **Poll()** with no income and outcome packets, the return status should be **EFI_NOT_READY**.

21.6 EFI_IP6CONFIG_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_IP6_CONFIG_PROTOCOL Section.

21.6.1 SetData()

Number	GUID	Assertion	Test Description
5.25.6.1.1	0x7a224cce, 0xb79b, 0x472a,0x9b, 0x8c,0xa4, 0x7e,0x07, 0x4d,0x5e, 0xef	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_INVALID_PARAMETER with Data being NULL	Call SetData() with Data is NULL, The return status should be EFI_INVALID_PARAMETER.
5.25.6.1.2	0x46f12872, 0x61f2, 0x46e4,0xa2, 0xf9,0x5f, 0x68,0x5b, 0x41,0x94, 0x79	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_INVALID_PARAMETER with ManualAddress being ::.	5.25.6.1.2 to 5.25.6.1.7 belong to one case. 1. Call SetData() with valid parameters except invalid ManualAddress(::), The return status should be EFI_INVALID_PARAMETER.
5.25.6.1.3	0x1cac93d3, 0x732a, 0x4e30,0x89, 0x4d,0xee, 0x63,0xb6, 0xf4,0x86, 0xa0	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_INVALID_PARAMETER with ManualAddress containing duplicated entries.	2. Call SetData() with valid parameters except invalid ManualAddress (2002::5000,2002::5001,2002::5000), The return status should be EFI_INVALID_PARAMETER.
5.25.6.1.4	0xd005ebf3, 0xcfd6, 0x498a, 0x90,0x05, 0xc2,0xb3, 0x70,0x2e, 0xb4,0xfc	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_INVALID_PARAMETER with Gateway being multicast.	3. Call SetData() with valid parameters except invalid Gateway (ff02::1), The return status should be EFI_INVALID_PARAMETER.
5.25.6.1.5	0x389806d5, 0x4506, 0x4319, 0x8d,0x17, 0x9b,0x4f, 0xc9,0xd9, 0x7e,0x25	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_INVALID_PARAMETER with Gateway containing duplicated entries.	4. Call SetData() with valid parameters except invalid Gateway (2002::5000,2002::5001,2002::5000), The return status should be EFI_INVALID_PARAMETER.
5.25.6.1.6	0x5aefdb0c, 0x322f, 0x49c3, 0x9d,0xd2, 0xdf,0xe2, 0x1b,0x66, 0xb3,0x08	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_INVALID_PARAMETER with DnsServer being multicast.	5. Call SetData() with valid parameters except invalid DnsServer (ff02::1), The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.25.6.1.7	0xd339988f, 0x2595, 0x4fb5, 0x81,0xae, 0xa9,0x4d, 0xc4,0x70, 0xb2,0x34	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_INVALID_PARAMETER with DnsServer containing duplicated entries.	6. Call SetData() with valid parameters except invalid DnsServer (2002::5000,2002::5001,2002::5000), The return status should be EFI_INVALID_PARAMETER.
5.25.6.1.8	0x4319a43b, 0x7641, 0x47c0, 0x84,0xbb, 0x98,0x5c, 0x47,0x99, 0x02,0xa2	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_WRITE_PROTECTED when trying to set InterfaceInfo.	Call SetData() to set InterfaceInfo, The return status should be EFI_WRITE_PROTECTED.
5.25.6.1.9	0x01f3b344, 0xeb52, 0x4086, 0xb9,0x49, 0x55,0xd7, 0xe4,0xdc, 0x5b,0xde	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_WRITE_PROTECTED when trying to set ManualAddress under Automatic policy.	5.25.6.1.9 to 5.25.6.1.11 belong to one case. 1. Call SetData() to set MaualAddress(2002::5000) under Automatic policy, The return status should be EFI_WRITE_PROTECTED.
5.25.6.1.10	0xf612af26, 0x2519, 0x497c, 0xb2,0x05, 0x37,0xa2, 0x91,0x4a, 0xee,0x05	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_WRITE_PROTECTED when trying to set Gateway under Automatic policy.	2. Call SetData() to set Gateway(2002::5001) under Automatic policy, The return status should be EFI_WRITE_PROTECTED.
5.25.6.1.11	0x592c1f3d, 0x249e, 0x4654, 0xb4,0xb1, 0x60,0x04, 0x21,0x62, 0x4d,0xd1	EFI_IP6CONFIG PROTOCOL. SetData() - SetData() returns EFI_WRITE_PROTECTED when trying to set DnsServer under Automatic policy.	3. Call SetData() to set DnsServer(2002::5001) under Automatic policy, The return status should be EFI_WRITE_PROTECTED.
5.25.6.1.12	0xd70bce29, 0x8026, 0x4e1b, 0xba,0x8b, 0x36,0xa3, 0x13,0xb4, 0x58,0x59	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_BAD BUFFER SIZE when trying to set ManualAddress with wrong DataSize.	5.25.6.1.12 to 5.25.6.1.17 belong to one case. 1. Call SetData() to set ManualAddress(2002::5000) with DataSize being 16, The return status should be EFI_BAD_BUFFER_SIZE.

Number	GUID	Assertion	Test Description
5.25.6.1.13	0xfe793490, 0x53f8, 0x4991, 0x83,0x48, 0xe6,0x24, 0x53,0x0e, 0x83,0xe9	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_BAD BUFFER SIZE when trying to set Gateway with wrong DataSize.	2. Call SetData() to set Gateway(2002::5001) with DataSize being 8, The return status should be EFI_BAD_BUFFER_SIZE.
5.25.6.1.14	0x42ccb2ef, 0xd706, 0x4d1a, 0xb2,0x47, 0xf4,0x2b, 0xba,0x99, 0xf7,0x07	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_BAD BUFFER SIZE when trying to set DnsServer with wrong DataSize.	3. Call SetData() to set Gateway(2002::5002) with DataSize being 8, The return status should be EFI_BAD_BUFFER_SIZE.
5.25.6.1.15	0x9168cb20, 0xc891, 0x42da, 0xbb,0x9f, 0x7a,0xdb, 0xe4,0x88, 0xb0,0x12	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_BAD BUFFER SIZE when trying to set AltInterfaceId with wrong DataSize.	4. Call SetData() to set AltInterfaceId with DataSize being 1, The return status should be EFI_BAD_BUFFER_SIZE.
5.25.6.1.16	0xad058d87, 0x1015, 0x4b2d, 0xa3,0x51, 0x5b,0xd4, 0xb0,0x93, 0x0b,0x7b	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_BAD BUFFER SIZE when trying to set DadXmits with wrong DataSize.	5. Call SetData() to set DadXmits(3) with DataSize being 1, The return status should be EFI_BAD_BUFFER_SIZE.
5.25.6.1.17	0x388be3f6, 0xd63e, 0x4cbf, 0xa3,0xd9, 0x3d,0x94, 0x18,0x23, 0x25,0x9b	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_BAD BUFFER SIZE when trying to set Policy with wrong DataSize.	6. Call SetData() to set Policy(Manual) with DataSize being 1, The return status should be EFI_BAD_BUFFER_SIZE.
5.25.6.1.18	0x2886bae1, 0x383a, 0x400f, 0x8f,0x88, 0x66,0x37, 0x6b,0x2a, 0x0f,0xf5	EFI_IP6CONFIG PROTOCOL.SetData() - SetData() returns EFI_UNSUPPORTED when trying to set Maximum	Call SetData() to set Maximum, The return status should be EFI_UNSUPPORTED.

Number	GUID	Assertion	Test Description
5.25.6.1.19	0xd2c61f06, 0x8822, 0x4a09, 0x89,0xa1, 0x7f,0x06, 0x67,0xfc, 0xaf,0x0e	`EFI_IP6CONFIG PROTOCOL.SetData() - SetData()` returns `EFI_ACCESS DENIED` when trying to set valid `ManualAddress` with last asynchronous setting not finished.	Intiate asynchronous `ManualAddress` setting process with `DadXmits 20`. Before the former setting finishes, Call `SetData()` to set valid `ManualAddress`, The return status should be `EFI_ACCESS_DENIED.`
5.25.6.1.20	0x0a5902da, 0x4142, 0x4494, 0xac,0x66, 0x2b,0x73, 0x1f,0xfe, 0xa6,0x71	`EFI_IP6CONFIG PROTOCOL.SetData() - SetData()` returns `EFI_SUCCESS` when trying to set valid `InterfaceId.`	5.25.6.1.20 to 5.25.6.1.23 belong to one case. 1. Call `SetData()` to set `InterfaceId(0:1:2:3:4:5:6:7),` The return status should be `EFI_SUCCESS.`
5.25.6.1.21	0xd9a9ef5e, 0xd819, 0x49d0, 0xbb,0x12, 0x25,0xad, 0xec,0x52, 0xdd,0xb3	Check the set `InterfaceId` to be as desired	2. Call `GetData()` to retrieve `InterfaceId` and validate it to be `(0:1:2:3:4:5:6:7)`, The compare result should be equal.
5.25.6.1.22	0x14e96019, 0x0815, 0x4486, 0x91,0x6c, 0xe4,0x40, 0xe1,0x66, 0x62,0x8e	`EFI_IP6CONFIG PROTOCOL.SetData() - SetData()` returns `EFI_SUCCESS` when trying to set valid `DadXmits.`	3. Call `SetData()` to set `DadXmits(3)`, The return status should be `EFI_SUCCESS.`
5.25.6.1.23	0x3458bbe0, 0x0d7e, 0x48ec, 0xb3,0x80, 0x2a,0x88, 0x5f,0x44, 0xe1,0x04	Check the set `DadXmits` to be as desired	4. Call `GetData()` to retrieve `DadXmits` and validate it to be `3`, The compare result should be equal.

21.6.2 GetData()

Number	GUID	Assertion	Test Description
5.25.6.2.1	0xd15e421d, 0x6228, 0x4fea, 0x8d,0x5a, 0x33,0x0f, 0xff,0x3f, 0x80,0xd2	EFI_IP6CONFIG PROTOCOL.GetData() - GetData() returns EFI_INVALID_PARAMETER with DataSize being NULL	Call GetData() with DataSize is NULL, The return status should be EFI_INVALID_PARAMETER.
5.25.6.2.2	0x38b36c04, 0x12e9, 0x4e96, 0xb2,0x4f, 0xc4,0x53, 0x85,0x1e, 0x6c,0x1d	EFI_IP6CONFIG PROTOCOL.GetData() - GetData() returns EFI_INVALID_PARAMETER with Data NULL and DataSize not zero	Call GetData() with Data NULL and DataSize is not zero, The return status should be EFI_INVALID_PARAMETER.
5.25.6.2.3	0xd05a6c59, 0x617f, 0x4549, 0x96,0x59, 0x4e,0x0c, 0xfc,0x3c, 0x33,0x36	EFI_IP6CONFIG PROTOCOL.GetData() - GetData() returns EFI_BUFFER TOO SMALL with DataSize smaller than Data's actual size.	5.25.6.2.3 to 5.25.6.2.4 belong to one case 1. Call GetData() to get ManualAddress with DataSize is 16, The return status should be EFI_BUFFER_TOO_SMALL.
5.25.6.2.4	0xed45c2fe, 0x9ec1, 0x4553, 0xaf,0xa4, 0x77,0x1e, 0x9d,0x4f, 0x76,0x11	The DataSize returned by GetData() should be equal to the actual size of the specific data type	2. Check the DataSize returned by GetData(), it should be equal to (sizeof EFI_IP6_CONFIG_MANUAL_ADDRESS).
5.25.6.2.5	0x59118c46, 0x2f2a, 0x4029, 0xab,0xd6, 0x76,0x74, 0x18,0x92, 0x03,0x69	EFI_IP6CONFIG PROTOCOL.GetData() - GetData() returns EFI_NOT FOUND when the data type doesn't exist.	Call GetData() to get Maximum, The return status should be EFI_NOT_FOUND.
5.25.6.2.6	0x55955d09, 0xc806, 0x4777, 0x9f,0xf0, 0x95,0xc0, 0x0e,0x79, 0xac,0x28	EFI_IP6CONFIG PROTOCOL.GetData() - GetData() returns EFI_NOT READY when trying to get valid ManualAddress with last asynchronous setting not finished.	Intiate asynchronous ManualAddress setting process with DadXmits 20. Before the former setting finishes, Call GetData() to get valid ManualAddress, The return status should be EFI_NOT_READY.

Number	GUID	Assertion	Test Description
5.25.6.2.7	0xfeaac1a0, 0x95bd, 0x4dcb, 0x91,0xc3, 0x9f,0x08, 0x50,0x4b, 0xef,0xa1	`EFI_IP6CONFIG PROTOCOL.GetData() - GetData()` returns `EFI_SUCCESS` when trying to get valid `InterfaceId.`	5.25.6.2.7 to 5.25.6.2.10 belong to one case. 1. Call `SetData()` to set `InterfaceId(0:1:2:3:4:5:6:7)` 2. Call `GetData()` to get `InterfaceId`,The return status should be `EFI_SUCCESS.`
5.25.6.2.8	0x3649d729, 0xd6d0, 0x456e, 0x84,0xae, 0xc7,0xe7, 0xb8,0x46, 0x43,0x43	Check the set `InterfaceId` to be as desired	3. Validate the retrieved `InterfaceId` to be `(0:1:2:3:4:5:6:7)`, The compare result should be equal.
5.25.6.2.9	0x165e79b4, 0xc987, 0x4100, 0x8a,0xa2, 0x8a,0xb1, 0x15,0xb0, 0x7f,0xad	`EFI_IP6CONFIG PROTOCOL.GetData() - GetData()` returns `EFI_SUCCESS` when trying to get valid `DadXmits.`	4. Call `SetData()` to set `DadXmits(3).` 5. Call `GetData()` to get `DadXmits`. The return status should be `EFI_SUCCESS.`
5.25.6.2.10	0xdb420311, 0x17f7, 0x40cf, 0xa0,0xb1, 0x02,0x94, 0xd5,0xdc, 0xcc,0x92	Check the set `DadXmits` to be as desired	6. Validate the retrieved `DadXmits` to be `3`, The compare result should be equal.

21.6.3 RegisterDataNotify()

Number	GUID	Assertion	Test Description
5.25.6.3.1	0x7e3f6157, 0xec75, 0x4ecd, 0xa7,0x9b, 0x49,0x26, 0xf3,0xaa, 0x1c,0x0d	EFI_IP6CONFIG PROTOCOL.RegisterDataNotify() - RegisterDataNotify() returns EFI_INVALID_PARAMETER with Event being NULL	Call RegisterDataNotify() with Event is NULL, The return status should be EFI_INVALID_PARAMETER.
5.25.6.3.2	0x70dc8c71, 0xc54d, 0x4446, 0x8a,0xd9, 0xba,0xc0, 0x86,0xe4, 0x3d,0x17	EFI_IP6CONFIG PROTOCOL.RegisterDataNotify() - RegisterDataNotify() returns EFI_UNSUPPORTED with Datatype not supported	Call RegisterDataNotify() with Datatype being Maximum, The return status should be EFI_UNSUPPORTED.
5.25.6.3.3	0x2d88f18b, 0x0bef, 0x4616, 0xbd,0xe5, 0xca,0x4e, 0x00,0x86, 0xe1,0xd3	EFI_IP6CONFIG PROTOCOL.RegisterDataNotify() - RegisterDataNotify() returns EFI_ACCESS_DENIED with Event already be registered on the same DataType.	1. Call RegisterDataNotify() with Datatype being Policy successfully. 2. Call RegisterDataNotify() with Datatype being Policy and the same Event again, The return status should be EFI_ACCESS_DENIED.
5.25.6.3.4	0x9a98dc85, 0xd018, 0x45aa, 0xb8,0x51, 0x34,0xee, 0x2f,0x67, 0x16,0xd4	EFI_IP6CONFIG PROTOCOL.RegisterDataNotify() - RegisterDataNotify() returns EFI_SUCCESS with valid parameters	5.25.6.3.4 to 5.25.6.3.5 belong to one case 1. Call RegisterDataNotify() with Datatype being ManualAddress successfully.
5.25.6.3.5	0x39f7fb37, 0x9f9f, 0x485e, 0x8d,0xbc, 0x0f,0x31, 0x91,0xda, 0x99,0x09	After the data is set, the Event should be signaled correctly.	2. The Event should be signaled and the context of the Event should be changed.
5.25.6.3.6	0xa13da599, 0x37e7, 0x474a, 0x93,0x43, 0x83,0xc9, 0xef,0xe8, 0x08,0x93	EFI_IP6CONFIG PROTOCOL.RegisterDataNotify() - RegisterDataNotify() returns EFI_SUCCESS with valid parameters	5.25.6.3.6 to 5.25.6.3.9 belong to one case. 1. Call RegisterDataNotify() with Datatype being Policy successfully.

Number	GUID	Assertion	Test Description
5.25.6.3.7	0x5428bdd5, 0x4332, 0x4e3b, 0x84,0x1f, 0x3e,0x60, 0x54,0x0a, 0xa3,0x5d	`EFI_IP6CONFIG PROTOCOL.RegisterDataNotify() – RegisterDataNotify()` returns `EFI_SUCCESS` with the same `Event`.	2. Call `RegisterDataNotify()` with `Datatype` being `DadXmits` and the same `Event` successfully.
5.25.6.3.8	0x1844a7c8, 0x730c, 0x4927, 0x8e,0x02, 0xce,0x0a, 0x6c,0xa0, 0x8d,0xcc	After the data is set, the `Event` should be signaled correctly.	3. Call `SetData()` to set `Policy`. The `Event` should be signaled and the context should be changed.
5.25.6.3.9	0xb0e66591, 0x9076, 0x48e3, 0x8d,0xf6, 0x2a,0x1d, 0x59,0xa5, 0x72,0xdb	After the data is set, the `Event` should be signaled correctly.	4. Call `SetData()` to set DadXmits. The `Event` should be signaled and the context should be changed.

21.6.4 UnregisterDataNotify()

Number	GUID	Assertion	Test Description
5.25.6.4.1	0x8ab0e5a2, 0xa4e1, 0x4282, 0x87,0xb5, 0xe3,0x77, 0xc7,0x63, 0xad,0x2f	EFI_IP6CONFIG PROTOCOL.Unregister DataNotify() - UnregisterDataNotif y() returns EFI_INVALID_PARAMET ER with Event being NULL	Call UnregisterDataNotify() with Event is NULL, The return status should be EFI_INVALID_PARAMETER.
5.25.6.4.2	0x5c68228f, 0xaaae, 0x4d0b, 0x99,0x27, 0x76,0x64, 0x47,0x6e, 0xf3,0x60	EFI_IP6CONFIG PROTOCOL.Unregister DataNotify() - UnregisterDataNotif y() returns EFI_NOT FOUND with no Event registered for the Datatype.	Call UnregisterDataNotify() with Datatype being ManualAddress and the Event not registered for the Datatype before, The return status should be EFI_NOT_FOUND.
5.25.6.4.3	0x55d8193e, 0xf58e, 0x4800, 0x92,0x4b, 0x73,0xc9, 0x02,0x09, 0x8d,0xd8	EFI_IP6CONFIG PROTOCOL.Unregister DataNotify() - UnregisterDataNotif y() returns EFI_NOT FOUND with Event first registered and then unregistered for the Datatype.	1. Call RegisterDataNotify() with Datatype being ManualAddress successfully. 2. Call UnregisterDataNotify() with Datatype being ManualAddress successfully. 3. Call UnregisterDataNotify() with Datatype being ManualAddress and the same Event again, The return status should be EFI_NOT_FOUND.
5.25.6.4.4	0x42eb4628, 0x8df6, 0x4704, 0x81,0xe5, 0xf7,0xea, 0xe6,0xcb, 0xb2,0x70	EFI_IP6CONFIG PROTOCOL.Unregister DataNotify() - UnregisterDataNotif y() returns EFI_SUCCESS with valid parameters.	1. Call RegisterDataNotify() with Datatype being Policy successfully. 2. Call UnregisterDataNotify() with Datatype being Policy successfully.
5.25.6.4.5	0x174cec07, 0xe573, 0x434b, 0x8e,0x99, 0x77,0xf8, 0xae,0x9c, 0x55,0xb5	EFI_IP6CONFIG PROTOCOL.Unregister DataNotify() - UnregisterDataNotif y() returns EFI_SUCCESS with valid parameters.	5.25.6.4.5 to 5.25.6.4.7 belong to one case. 1.Call RegisterDataNotify() with Datatype being Policy successfully. 2.Call RegisterDataNotify() with Datatype being DadXmits successfully. 3.Call UnregisterDataNotify() with Datatype being Policy successfully.

Number	GUID	Assertion	Test Description
5.25.6.4.6	0x1f5ef1af, 0x8a19, 0x48d6, 0x83,0x1f, 0x51,0xbe, 0x00,0xb3, 0x2a,0xa5	After the data is set, the unregistered `Event` should not be signaled correctly.	4. Call `SetData()` to set `Policy`. The `Event` should not be signaled and the context should not be changed.
5.25.6.4.7	0x388c8838, 0x7790, 0x4a1f, 0x9d,0xb7, 0x50,0x17, 0xd7,0xaa, 0x60,0xdb	After the data is set, the registered `Event` should be signaled correctly.	5. Call `SetData()` to set `DadXmits`. The `Event` should be signaled and the context should be changed.

21.7 EFI_IPSEC_CONFIG_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_IPSEC_CONFIG_PROTOCOL Section.

21.7.1 SetData()

Number	GUID	Assertion	Test Description
5.25.7.1.1	0x235a63c3, 0x2ba4, 0x4d1d, 0x8e, 0x25, 0xc8, 0x7e, 0x47, 0x35, 0x36, 0x1c	EFI_IPSEC_CONFIG PROTOCOL.SetData() - SetData() returns EFI_UNSUPPORTED with an invalid DataType (>2)	Call SetData() with an invalid DataType (>2), The return status should be EFI_UNSUPPORTED.
5.25.7.1.2	0x77f0b145, 0x48a3, 0x4780, 0x8c, 0x0e, 0x63, 0x5b, 0x91, 0x6f, 0x4d, 0xf5	EFI_IPSEC_CONFIG PROTOCOL.SetData() - SetData() returns EFI_SUCCESS with valid DataType(0)/Selector/ Data.	5.25.7.1.2 to 5.25.7.1.4 belong to one case. 1. Call SetData() with valid DataType(0)/Selector/Data. The return status should be EFI_SUCCESS.
5.25.7.1.3	0x8739610b, 0xabf3, 0x4994, 0x96, 0xee, 0x87, 0xd4, 0x95, 0x27, 0x45, 0x67	EFI_IPSEC_CONFIG PROTOCOL. SetData () - returns EFI_SUCCESS with valid DataType(0) /Selector and NULL Data.	2. Call SetData() with valid DataType(0) /Selector and NULL Data, The return status should be EFI_SUCCESS.
5.25.7.1.4	0xeb931bcf, 0x074a, 0x4e69, 0x83, 0xee, 0xd3, 0xc6, 0x39, 0xc6, 0x84, 0xef	EFI_IPSEC_CONFIG PROTOCOL.SetData() - After flush given selector configuration by SetData, GetData() returns EFI_NOT_FOUND with valid DataType(0) / Selector/DataSize.	3. Call GetData() with valid DataType(0) /Selector/DataSize, The return status should be EFI_NOT_FOUND.
5.25.7.1.5	0x35ec56a7, 0x1c1a, 0x4c84, 0xb0, 0x68, 0x40, 0x53, 0x7c, 0x45, 0x95, 0x41	EFI_IPSEC_CONFIG PROTOCOL.SetData() - SetData() returns EFI_SUCCESS with valid DataType(1)/Selector/ Data.	5.25.7.1.5 to 5.25.7.1.7 belong to one case. 1. Call SetData() with valid DataType(1)/Selector/Data. The return status should be EFI_SUCCESS.
5.25.7.1.6	0x8b6ddfbf, 0x8de1, 0x418d, 0xb0, 0x76, 0xf4, 0x48, 0x07, 0x46, 0xb6, 0x3a	EFI_IPSEC_CONFIG PROTOCOL. SetData () - SetData() returns EFI_SUCCESS with valid DataType(1) /Selector and NULL Data.	2. Call SetData() with valid DataType(1) /Selector and NULL Data, The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.25.7.1.7	0xa510e599, 0x2cdd, 0x4c14, 0xbe, 0xc9, 0xbd, 0x2f, 0xd8, 0x7d, 0x50, 0x60	EFI_IPSEC_CONFIG PROTOCOL.SetData() – After flush given selector configuration by SetData, GetData() returns EFI_NOT_FOUND with valid DataType(1) / Selector/DataSize.	3. Call GetData() with valid DataType(1) /Selector/DataSize, The return status should be EFI_NOT_FOUND.
5.25.7.1.8	0x69d0edc5, 0xd259, 0x42ea, 0xa6, 0x97, 0x47, 0x8c, 0x2a, 0x32, 0x0c, 0x08	EFI_IPSEC_CONFIG PROTOCOL.SetData() – SetData() returns EFI_SUCCESS with valid DataType(2)/Selector/ Data.	5.25.7.1.8 to 5.25.7.1.10 belong to one case. 1. Call SetData() with valid DataType(2)/Selector/Data. The return status should be EFI_SUCCESS.
5.25.7.1.9	0xe389a40e, 0x4c21, 0x4cf1, 0x88, 0xb3, 0xae, 0x86, 0x9b, 0x0b, 0xc2, 0x35	EFI_IPSEC_CONFIG PROTOCOL. SetData () – SetData() returns EFI_SUCCESS with valid DataType(2) /Selector and NULL Data.	2. Call SetData() with valid DataType(2) /Selector and NULL Data, The return status should be EFI_SUCCESS.
5.25.7.1.10	0x4d6b9807, 0x4d26, 0x43aa, 0x8a, 0x53, 0xd1, 0xff, 0xe5, 0x2b, 0xb0, 0xde	EFI_IPSEC_CONFIG PROTOCOL.SetData() – After flush given selector configuration by SetData, GetData() returns EFI_NOT_FOUND with valid DataType(2) / Selector/DataSize.	3. Call GetData() with valid DataType(2) /Selector/DataSize, The return status should be EFI_NOT_FOUND.
5.25.7.1.11	0x5747257a, 0xabff, 0x4ac4, 0xa9, 0xb0, 0xfc, 0x82, 0xf7, 0xd0, 0xce, 0xa2	EFI_IPSEC_CONFIG PROTOCOL.SetData() – SetData() returns EFI_SUCCESS with valid DataType(0)/Selector/ Data.	5.25.7.1.11 to 5.25.7.1.13 belong to one case. 1. Call SetData() with valid DataType(0)/Selector/Data. The return status should be EFI_SUCCESS.
5.25.7.1.12	0x808d03fc, 0x2d68, 0x4c51, 0x90, 0x31, 0x01, 0x32, 0x64, 0xf5, 0xf7, 0x85	EFI_IPSEC_CONFIG PROTOCOL. SetData () – SetData() returns EFI_SUCCESS with valid DataType(0) /Data and NULL Selector.	2. Call SetData() with valid DataType(0) /Data and NULL Selector, The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.25.7.1.13	0x2f5d587d, 0x4216, 0x42dd, 0x92, 0x41, 0x72, 0x60, 0xe9, 0x65, 0xa6, 0xf6	EFI_IPSEC_CONFIG PROTOCOL.SetData() - After flush entire configuration by SetData, GetData() returns EFI_NOT_FOUND with valid DataType(0) / Selector/DataSize.	3. Call GetData() with valid DataType(0) /Selector/DataSize, The return status should be EFI_NOT_FOUND.
5.25.7.1.14	0x39a5db14, 0xebb0, 0x460f, 0x92, 0x99, 0x36, 0x28, 0x3f, 0x51, 0x9d, 0xff	EFI_IPSEC_CONFIG PROTOCOL.SetData() - SetData() returns EFI_SUCCESS with valid DataType(1)/Selector/ Data.	5.25.7.1.14 to 5.25.7.1.16 belong to one case. 1. Call SetData() with valid DataType(1)/Selector/Data. The return status should be EFI_SUCCESS.
5.25.7.1.15	0xdee52264, 0x3da1, 0x4f5d, 0xa2, 0x43, 0xa1, 0x15, 0xad, 0xd3, 0x3f, 0x40	EFI_IPSEC_CONFIG PROTOCOL. SetData () - SetData() returns EFI_SUCCESS with valid DataType(1) /Data and NULL Selector.	2. Call SetData() with valid DataType(1) /Data and NULL Selector, The return status should be EFI_SUCCESS.
5.25.7.1.16	0xd76b9b01, 0x6649, 0x4b43, 0xa0, 0x05, 0x1a, 0x64, 0x69, 0xc3, 0xef, 0x0f	EFI_IPSEC_CONFIG PROTOCOL.SetData() - After flush entire configuration by SetData, GetData() returns EFI_NOT_FOUND with valid DataType(1) / Selector/DataSize.	3. Call GetData() with valid DataType(1) /Selector/DataSize, The return status should be EFI_NOT_FOUND.
5.25.7.1.17	0x5f9e36d3, 0xa945, 0x4b20, 0xa2, 0x9b, 0x30, 0x3e, 0x9b, 0xd5, 0x6c, 0xcd	EFI_IPSEC_CONFIG PROTOCOL.SetData() - SetData() returns EFI_SUCCESS with valid DataType(2)/Selector/ Data.	5.25.7.1.17 to 5.25.7.1.19 belong to one case. 1. Call SetData() with valid DataType(2)/Selector/Data. The return status should be EFI_SUCCESS.
5.25.7.1.18	0xaec61686, 0xf303, 0x4697, 0xb0, 0x7d, 0xe2, 0x08, 0x8e, 0x52, 0x05, 0x58	EFI_IPSEC_CONFIG PROTOCOL. SetData () - SetData() returns EFI_SUCCESS with valid DataType(2) /Data and NULL Selector.	2. Call SetData() with valid DataType(2) /Data and NULL Selector, The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.25.7.1.19	0x69c4e05f, 0x7b94, 0x4c82, 0x81, 0x47, 0xd9, 0x14, 0x57, 0x86, 0x24, 0x3f	EFI_IPSEC_CONFIG PROTOCOL.SetData() – After flush entire configuration by SetData, GetData() returns EFI_NOT_FOUND with valid DataType(2) / Selector/DataSize.	3. Call GetData() with valid DataType(2) /Selector/DataSize, The return status should be EFI_NOT_FOUND.
5.25.7.1.20	0x486c7a3e, 0x4a65, 0x4da6, 0x8e, 0x52, 0x6b, 0x64, 0x48, 0xc3, 0x68, 0xaa	EFI_IPSEC_CONFIG PROTOCOL.SetData() returns EFI_SUCCESS valid DataType(1)/ Selector/SA_Data2	5.25.7.1.20 to 5.25.7.1.22 belong to one case. 1. Call SetData() with valid DataType(1)/Selector/ SA_Data2. The return status should be EFI_SUCCESS
5.25.7.1.21	0x92302107, 0x20fa, 0x49b9, 0x84, 0x5f, 0xec, 0xc6, 0xe0, 0x28, 0x31, 0xf3	EFI_IPSEC_CONFIG PROTOCOL. SetData () – SetData() returns EFI_SUCCESS with valid DataType(1)/Selector and NULL Data.	2. Call SetData() with valid DataType(1) /Selector and NULL Data, The return status should be EFI_SUCCESS.
5.25.7.1.22	0x03b2df9d, 0xe5c1, 0x47b3, 0xaa, 0x7a, 0xa0, 0xbb, 0x1d, 0xf2, 0xf0, 0x9b	EFI_IPSEC_CONFIG PROTOCOL.SetData() – After flush given selector configuration by SetData, GetData() returns EFI_NOT_FOUND with valid DataType(1) / Selector/DataSize.	3. Call GetData() with valid DataType(1)/Selector/DataSize, The return status should be EFI_NOT_FOUND.

21.7.2 GetData()

Number	GUID	Assertion	Test Description
5.25.7.2.1	0xa8339798, 0x45fa, 0x47a8, 0xaf, 0x9e, 0x74, 0x17, 0xcd, 0x78, 0xef, 0x40	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns EFI_INVALID_PARAMETER with NULL Selector.	Call GetData() with NULL Selector, The return status should be EFI_INVALID_PARAMETER.
5.25.7.2.2	0x1d04e3e9, 0xfc36, 0x4321, 0xa8, 0x22, 0x51, 0xb2, 0x59, 0x01, 0xbf, 0xb0	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns EFI_UNSUPPORTED with an invalid DataType (>2)	Call SetData() with an invalid DataType (>2), The return status should be EFI_UNSUPPORTED.
5.25.7.2.3	0x4da58bcc, 0x1ae2, 0x450d, 0xbc, 0x1b, 0x0d, 0x76, 0x77, 0x3a, 0xab, 0x79	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns EFI_INVALID_PARAMETER with NULL Data.	Call GetData() with NULL Data, The return status should be EFI_INVALID_PARAMETER.
5.25.7.2.4	0x39962424, 0x200d, 0x40cd, 0x8f, 0x5b, 0xfd, 0x3f, 0xf8, 0xaa, 0x51, 0x96	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns EFI_INVALID_PARAMETER with NULL DataSize.	Call GetData() with NULL DataSize, The return status should be EFI_INVALID_PARAMETER.
5.25.7.2.5	0x1ef8f8fb, 0xf494, 0x4411, 0x87, 0xd2, 0x73, 0x43, 0x88, 0x6a, 0x14, 0xe7	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns EFI_BUFFER_TOO_SMALL with small DataSize.	Call GetData() with small DataSize, The return status should be EFI_BUFFER_TOO_SMALL.
5.25.7.2.6	0xddc718a3, 0xb10d, 0x4f05, 0x9d, 0x97, 0x65, 0xda, 0x75, 0xd9, 0x02, 0xca	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns EFI_INVALID_PARAMETER with NULL Data.	Call GetData() with NULL Data, The return status should be EFI_INVALID_PARAMETER.
5.25.7.2.7	0xc6d16b39, 0x34f6, 0x438a, 0xa5, 0x77, 0xbf, 0xd3, 0x13, 0xbc, 0x9e, 0xe8	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns EFI_INVALID_PARAMETER with NULL DataSize.	Call GetData() with NULL DataSize, The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.25.7.2.8	0xa5fecb65, 0x0501, 0x4d66, 0xbe, 0x1c, 0x37, 0xac, 0xb7, 0x8a, 0xd4, 0xe8	EFI_IPSEC_CONFIG PROTOCOL.GetData() – GetData() returns EFI_BUFFER_TOO_SMALL with small DataSize.	Call GetData() with small DataSize, The return status should be EFI_BUFFER_TOO_SMALL.
5.25.7.2.9	0x6b1c7e3e, 0x47e7, 0x40ef, 0x85, 0xec, 0x3b, 0x8c, 0x0f, 0xa6, 0x08, 0x1f	EFI_IPSEC_CONFIG PROTOCOL.GetData() – GetData() returns EFI_INVALID_PARAMETER with NULL Data.	Call GetData() with NULL Data, The return status should be EFI_INVALID_PARAMETER.
5.25.7.2.10	0xb4138aae, 0xccfb, 0x45af, 0xa6, 0x41, 0x0a, 0x1c, 0x7f, 0x9d, 0x86, 0x1b	EFI_IPSEC_CONFIG PROTOCOL.GetData() – GetData() returns EFI_INVALID_PARAMETER with NULL DataSize.	Call GetData() with NULL DataSize, The return status should be EFI_INVALID_PARAMETER.
5.25.7.2.11	0xea851d2d, 0x4031, 0x4966, 0x91, 0x8e, 0x24, 0xda, 0x2a, 0x56, 0xc3, 0xb7	EFI_IPSEC_CONFIG PROTOCOL.GetData() – GetData() returns EFI_BUFFER_TOO_SMALL with small DataSize.	Call GetData() with small DataSize, The return status should be EFI_BUFFER_TOO_SMALL.
5.25.7.2.12	0xd2cabfe5, 0x85a0, 0x47a1, 0x8d,0x71, 0x3c,0x3f, 0x64,0x4a, 0x41,0xf3	EFI_IPSEC_CONFIG PROTOCOL.GetData() – GetData() returns EFI_INVALID_PARAMETER with NULL SA_DATA2	Call GetData() with NULL SA_DATA2, The return status should be EFI_INVALID_PARAMETER .
5.25.7.2.13	0x91591c0, 0x5a13, 0x448e, 0xbf, 0x21, 0x1d, 0x12, 0xb3, 0x8c, 0x9e, 0x6d	EFI_IPSEC_CONFIG PROTOCOL.GetData() – GetData() returns EFI_INVALID_PARAMETER with NULL SA_DATA2 datasize	Call GetData() with NULL SA_DATA2 datasize, The return status should be EFI_INVALID_PARAMETER
5.25.7.2.14	0x64ec8c85, 0x7661, 0x4364, 0xa1, 0xf3, 0x56, 0x62, 0x69, 0x3d, 0x8a, 0x7a	EFI_IPSEC_CONFIG PROTOCOL.GetData() – GetData() returns EFI_BUFFER_TOO_SMALL with small SA_DATA2 datasize	Call GetData() with small SA_DATA2 datasize, The return status should be EFI_BUFFER_TOO_SMALL .

Number	GUID	Assertion	Test Description
5.25.7.2.15	0x437749ac, 0x27bc, 0x46ac, 0xb7, 0xa1, 0x1b, 0x39, 0xee, 0xcc, 0x58, 0xc0	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns EFI_SUCCESS with Valid DataType(0)/Selector/ DataSize.	Call GetData() with Valid DataType(0)/Selector/DataSize. The return status should be EFI_SUCCESS.
5.25.7.2.16	0xe53c2379, 0x58fb, 0x402f, 0xbb, 0x47, 0x12, 0xd7, 0xe3, 0x55, 0x8d, 0x01	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns the right values which are set before.	Call GetData() with Valid DataType(0)/Selector/DataSize. The right values should be same as the values which are set before.
5.25.7.2.17	0x37f06d59, 0x2e1f, 0x4ccd, 0x83, 0xbc, 0x1b, 0xf2, 0xcf, 0x4b, 0x92, 0x4e	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns EFI_SUCCESS with Valid DataType(1)/Selector/ DataSize.	Call GetData() with Valid DataType(1)/Selector/DataSize. The return status should be EFI_SUCCESS.
5.25.7.2.18	0x077a8be2, 0xdd60, 0x48b5, 0xaf, 0x2e, 0x05, 0xcd, 0xc7, 0x07, 0x64, 0xf0	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns the right values which are set before.	Call GetData() with Valid DataType(1)/Selector/DataSize. The right values should be same as the values which are set before.
5.25.7.2.19	0x35adfec2, 0x5c65, 0x431f, 0x87, 0x86, 0x7b, 0x70, 0x81, 0x69, 0x71, 0xba	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns EFI_SUCCESS with Valid DataType(2)/Selector/ DataSize.	Call GetData() with Valid DataType(2)/Selector/DataSize. The return status should be EFI_SUCCESS.
5.25.7.2.20	0x26a81e68, 0x1aec, 0x4f1f, 0x9c, 0xe5, 0xc1, 0x59, 0xf2, 0xf3, 0xea, 0x12	EFI_IPSEC_CONFIG PROTOCOL.GetData() - GetData() returns the right values which are set before.	Call GetData() with Valid DataType(2)/Selector/DataSize. The right values should be same as the values which are set before.
5.25.7.2.21	0x378cd479, 0x2dd4, 0x4bc8,0x9b, 0xd8, 0x8c, 0x23, 0xfd, 0xda, 0x5d, 0x20	EFI_IPSEC_CONFIG_PR OTO COL.GetData- GetData() returns EFI_SUCCESS with valid DataType(1)/ Selector/DataSize	Call GetData() with valid DataType(1)/Selector/DataSize, The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.25.7.2.22	0x34fc6d63, 0xb2ec, 0x4c20, 0xb7, 0x7d, 0xa8, 0xf8, 0xf, 0x74, 0x7b, 0xa3	**EFI_IPSEC_CONFIG_PROTO COL.GetData-GetData()** returns **EFI_SUCCESS** & the right **SA_DATA2** which are set before	Call **GetData()** returns the right **SA_DATA2** which are set before, The return status should be **EFI_SUCCESS.**

21.7.3 GetNextSelector ()

Number	GUID	Assertion	Test Description
5.25.7.3.1	0xf85ce018, 0x2fad, 0x4b4e, 0xbb, 0xbb, 0x1c, 0x59, 0x57, 0x12, 0x85, 0xac	EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI_UNSUPPORTED with an invalid DataType (>2)	Call GetNextSelector() with an invalid DataType (>2). The return status should be EFI_UNSUPPORTED.
5.25.7.3.2	0x17a12f39, 0xba49, 0x4abb, 0x8f, 0x52, 0x3a, 0x32, 0x24, 0x8e, 0x04, 0xdd	EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI_INVALID_PARAMETER with NULL SelectorSize.	Call GetNextSelector() with NULL SelectorSize. The return status should be EFI_INVALID_PARAMETER.
5.25.7.3.3	0xc404ce41, 0x6802, 0x415d, 0x8b, 0x76, 0x41, 0x26, 0x65, 0x1d, 0x56, 0x29	EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI_INVALID_PARAMETER with NULL Selector.	Call GetNextSelector() with NULL Selector. The return status should be EFI_INVALID_PARAMETER.
5.25.7.3.4	0x23b72aad, 0xa975, 0x4500, 0x95, 0x19, 0x2e, 0x6d, 0xc4, 0x5f, 0x23, 0x27	EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI_BUFFER_TOO_SMALL with valid DataType(0)/ Selector and SelectorSize is 0.	Call GetNextSelector() with valid DataType(0)/Selector and SelectorSize is 0. The return status should be EFI_BUFFER_TOO_SMALL.
5.25.7.3.5	0xa11a6002, 0x911b, 0x4702, 0x85, 0xa7, 0xc9, 0x73, 0x91, 0xa6, 0xdb, 0x6d	EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI_BUFFER_TOO_SMALL with valid DataType(1)/ Selector and SelectorSize is 0.	Call GetNextSelector() with valid DataType(1)/Selector and SelectorSize is 0. The return status should be EFI_BUFFER_TOO_SMALL.

Number	GUID	Assertion	Test Description
5.25.7.3.6	0xccbcee8b, 0xf23b, 0x4c70, 0x8e, 0x3b, 0x19, 0xdb, 0xa6, 0xd1, 0xa8, 0x51	EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI_BUFFER_TOO_SMALL with valid DataType(2)/ Selector and SelectorSize is 0.	Call GetNextSelector() with valid DataType(2)/Selector and SelectorSize is 0. The return status should be EFI_BUFFER_TOO_SMALL.
5.25.7.3.7	0x502ad851, 0x41ae, 0x483e, 0xaa, 0xcd, 0x8d, 0x23, 0x73, 0x04, 0x91, 0xcf	EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI_SUCCESS with valid DataType(0)/ Selector and SelectorSize.	Call GetNextSelector() with valid DataType(0)/Selector and SelectorSize. The return status should be EFI_SUCCESS.
5.25.7.3.8	0x2f0d92f8, 0x2371, 0x4547, 0xa9, 0x5e, 0x79, 0x09, 0xc8, 0x62, 0xee, 0x26	EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI_SUCCESS with valid DataType(0)/ Selector and SelectorSize.	Call GetNextSelector() with valid DataType(0)/Selector and SelectorSize. The return status should be EFI_NOT_FOUND.
5.25.7.3.9	0xdaa5a475, 0x0d4a, 0x4e58, 0xa4, 0xd4, 0xfe, 0x33, 0xe7, 0x13, 0xd5, 0xbd	EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI_SUCCESS with valid DataType(1)/ Selector and SelectorSize.	Call GetNextSelector() with valid DataType(1)/Selector and SelectorSize. The return status should be EFI_SUCCESS.
5.25.7.3.10	0x78ea1b63, 0x979e, 0x41fe, 0xab, 0xb1, 0xc3, 0xb3, 0x42, 0x38, 0xc2, 0xa0	EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector() returns EFI_SUCCESS with valid DataType(1)/ Selector and SelectorSize.	Call GetNextSelector() with valid DataType(1)/Selector and SelectorSize. The return status should be EFI_NOT_FOUND.

Number	GUID	Assertion	Test Description
5.25.7.3.11	0xd570e742, 0x8122, 0x4abc, 0xbb, 0xe8, 0x34, 0xcf, 0x8f, 0x6e, 0x00, 0xdd	**EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector()** returns **EFI_SUCCESS** with valid **DataType**(2)/ **Selector** and **SelectorSize**.	Call **GetNextSelector()** with valid **DataType**(2)/**Selector** and **SelectorSize**. The return status should be **EFI_SUCCESS**.
5.25.7.3.12	0xb3a7efaa, 0x0c6e, 0x4686, 0xad, 0x77, 0xab, 0xd2, 0x62, 0xb4, 0x71, 0xfb	**EFI_IPSEC_CONFIG PROTOCOL. GetNextSelector() – GetNextSelector()** returns **EFI_SUCCESS** with valid **DataType**(2)/ **Selector** and **SelectorSize**.	Call **GetNextSelector()** with valid **DataType**(2)/**Selector** and **SelectorSize**. The return status should be **EFI_NOT_FOUND**.

21.7.4 RegisterDataNotify ()

Number	GUID	Assertion	Test Description
5.25.7.4.1	0x22857d7f, 0xa20c, 0x467f, 0xa5, 0x70, 0x54, 0xbd, 0x56, 0x3d, 0x93, 0x7e	EFI_IPSEC_CONFIG PROTOCOL. RegisterDataNotify() - RegisterDataNotify() returns EFI_INVALID_PARAMETER with NULL Event.	Call RegisterDataNotify() with NULL Event. The return status should be EFI_INVALID_PARAMETER.
5.25.7.4.2	0x9361ecca, 0xf59a, 0x4d4c, 0xb5, 0x9d, 0x1a, 0xc8, 0xf3, 0x7b, 0x75, 0x1a	EFI_IPSEC_CONFIG PROTOCOL. RegisterDataNotify() - RegisterDataNotify() returns EFI_UNSUPPORTED with invalid DataType(>2).	Call RegisterDataNotify() with invalid DataType(>2). The return status should be EFI_UNSUPPORTED.
5.25.7.4.3	0x9bd0dce3, 0x15c1, 0x4104, 0x82, 0x3f, 0x35, 0x80, 0x97, 0x00, 0x49, 0xcb	EFI_IPSEC_CONFIG PROTOCOL. RegisterDataNotify() - RegisterDataNotify() returns EFI_SUCCESS with valid DataType/ Event.	Call RegisterDataNotify() with valid DataType/Event. The return status should be EFI_SUCCESS.
5.25.7.4.4	0x53fe8163, 0xb212, 0x4c7e, 0x88, 0xa0, 0xe9, 0x90, 0x0a, 0x10, 0x20, 0x75	EFI_IPSEC_CONFIG PROTOCOL. RegisterDataNotify() - RegisterDataNotify() returns EFI_ ACCESS_DENIED with valid DataType/Event.	Call RegisterDataNotify() with valid DataType/Event. The return status should be EFI_ACCESS_DENIED.
5.25.7.4.5	0xe3ef592d, 0xb247, 0x417f, 0xad, 0x54, 0x4e, 0xfc, 0x0b, 0x7a, 0x03, 0x02	EFI_IPSEC_CONFIG PROTOCOL. RegisterDataNotify() - RegisterDataNotify() returns EFI_SUCCESS with valid DataType/ Event.	Call RegisterDataNotify() with valid DataType/Event. The return status should be EFI_SUCCESS.

21.7.5 UnregisterDataNotify ()

Number	GUID	Assertion	Test Description
5.25.7.5.1	0x4fd58448, 0x8d87, 0x4bd0, 0xbf, 0xd1, 0xe0, 0xa5, 0x7a, 0x70, 0xce, 0x0c	EFI_IPSEC_CONFIG PROTOCOL. UnregisterDataNotify() - UnregisterDataNotify() returns EFI_INVALID_PARAMETER with NULL Event.	Call UnregisterDataNotify() with NULL Event. The return status should be EFI_INVALID_PARAMETER.
5.25.7.5.2	0x12dd249e, 0xa481, 0x4a9a, 0x87, 0x45, 0xa9, 0xfd, 0x26, 0xac, 0xb1, 0xc8	EFI_IPSEC_CONFIG PROTOCOL. UnregisterDataNotify() - UnregisterDataNotify() returns EFI_UNSUPPORTED with invalid DataType(>2).	Call UnregisterDataNotify() with invalid DataType(>2). The return status should be EFI_UNSUPPORTED.
5.25.7.5.3	0xa561620c, 0xfc80, 0x478d, 0xab, 0x8c, 0x2c, 0xdb, 0xc8, 0x47, 0x46, 0xc4	EFI_IPSEC_CONFIG PROTOCOL. UnregisterDataNotify() - UnregisterDataNotify() returns EFI_NOT_FOUND with valid DataType/Event.	Call UnregisterDataNotify() with valid DataType/Event. The return status should be EFI_NOT_FOUND.
5.25.7.5.4	0x3053b6d9, 0xa5ba, 0x41c1, 0xad, 0x8f, 0x49, 0xf3, 0x37, 0x9f, 0x90, 0x55	EFI_IPSEC_CONFIG PROTOCOL. UnregisterDataNotify() - UnregisterDataNotify() returns EFI_SUCCESS with valid DataType/ Event.	Call UnregisterDataNotify() with valid DataType/Event. The return status should be EFI_SUCCESS.
5.25.7.5.5	0xa829c13e, 0x551d, 0x443e, 0xaf, 0xa0, 0x1d, 0x8d, 0x0a, 0xea, 0x61, 0x98	EFI_IPSEC_CONFIG PROTOCOL. UnregisterDataNotify() - UnregisterDataNotify() returns EFI_NOT_FOUND with valid DataType/Event.	Call UnregisterDataNotify() with valid DataType/Event. The return status should be EFI_NOT_FOUND.

21.8 EFI_IPSEC2_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_IPSEC2_PROTOCOL Section.

21.8.1 ProcessExt()

Number	GUID	Assertion	Test Description
5.25.8.1.1	0x5de601fb, 0xc3c4, 0x4bff, 0x89, 0x3e, 0xdd, 0x40, 0x67, 0xd1, 0xe1, 0x6b	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_INVALID_PARAMETER with NULL OptionsBuffer Input	1. Call ProcessExt() with NULL OptionsBuffer Input. 2.The return code should be EFI_INVALID_PARAMETER
5.25.8.1.2	0xd7cf3852, 0xcb7c, 0x4f68, 0x9b, 0x28, 0x56, 0x64, 0x72, 0xbe, 0xe3, 0x3d	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_INVALID_PARAMETER with NULL OptionsLength Input	1. Call ProcessExt() with NULL OptionsLength Input. 2. The return code should be EFI_INVALID_PARAMETER
5.25.8.1.3	0xf33aeb54, 0xe1be, 0x4541, 0xac, 0x79, 0x4e, 0xc1, 0xbc, 0x23, 0x87, 0x2b	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_INVALID_PARAMETER with NULL FragmentTable Input	1. Call ProcessExt() with Null FragmentTable Input. 2.The return code should be EFI_INVALID_PARAMETER
5.25.8.1.4	0x861f3f9, 0x4361, 0x4a23, 0x98, 0x41, 0xf0, 0x2d, 0x14, 0x97, 0x33, 0xb6	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_INVALID_PARAMETER with NULL FragmentCount Input	1. Call ProcessExt() with NULL FragmentCount Input. 2. The return code should be EFI_INVALID_PARAMETER
5.25.8.1.5	0x2b45f62a, 0xb9f, 0x473d, 0xbb, 0x5f, 0xcf, 0x59, 0x35, 0xed, 0xae, 0x4a	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Transport Mode OutBound Call to do IP4 IPSEC with Encrypt Algorithm {SHA1HMAC, 3DESCBC}	1. Call ProcessExt() in Transport Mode OutBound Call to do IP4 IPSEC with Encrypt Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI_SUCCESS
5.25.8.1.6	0xd486fd03, 0x7888, 0x42ed, 0x8f, 0xdd, 0xc5, 0xb, 0x40, 0xae, 0x25, 0xd7	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC} and check if Packet Header content is intact	1.Call ProcessExt() in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI_SUCCESS and Packet Header content is intact.

Number	GUID	Assertion	Test Description
5.25.8.1.7	0xfd4a5c6f, 0x9072, 0x463a, 0xb6, 0x5, 0x80, 0x72, 0x80, 0x14, 0x13, 0xc9	`EFI_IPSEC2_PROTOCOL.ProcessExt-ProcessExt()` returns `EFI_SUCCESS` in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. Check if Packet Payload Content is intact	1.Call `ProcessExt()` in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be `EFI_SUCCESS` and Packet Payload Content is intact.
5.25.8.1.8	0xbcddcd9a, 0xc0d9, 0x450c, 0xbc, 0xdb, 0xe0, 0xeb, 0x1c, 0xb7, 0x98, 0x3d	`EFI_IPSEC2_PROTOCOL.ProcessExt-ProcessExt()` returns `EFI_SUCCESS` in Transport Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC}	1.Call `ProcessExt()` in Transport Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC} 2.The return code should be `EFI_SUCCESS`.
5.25.8.1.9	0xd89ad072, 0xfd5e, 0x42af, 0x83, 0x4a, 0xf2, 0xde, 0xcb, 0xfd, 0x9, 0x2d	`EFI_IPSEC2_PROTOCOL.ProcessExt-ProcessExt()` returns `EFI_SUCCESS` in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. Check if Packet Header content is intact	1.Call `ProcessExt()` in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be `EFI_SUCCESS` and Packet Header content is intact
5.25.8.1.10	0x530369c, 0xaf77, 0x4064, 0xbc, 0xc1, 0x70, 0x68, 0x31, 0x4, 0x76, 0x94	`EFI_IPSEC2_PROTOCOL.ProcessExt-ProcessExt()` returns `EFI_SUCCESS` in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. Check if Packet Header content is intact	1.Call `ProcessExt()` in Transport Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be `EFI_SUCCESS` and Packet Header content is intact
5.25.8.1.11	0x6d729b2d, 0x1524, 0x49ae, 0xb6, 0xb9, 0xfa, 0xee, 0x59, 0x51, 0xe1, 0x61	`EFI_IPSEC2_PROTOCOL.ProcessExt-ProcessExt()` returns `EFI_SUCCESS` in Tunnel Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, 3DESCBC}	1.Call `ProcessExt()` in Tunnel Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.25.8.1.12	0x79eba4f0, 0xcfd0, 0x42fa, 0xb7, 0x94, 0x21, 0xa2, 0xd9, 0xac, 0xfa, 0x34	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. Check Returned Packet Header is set ZERO	1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI_SUCCESS and Returned Packet Header is set ZERO.
5.25.8.1.13	0xd23154b3, 0xbe46, 0x4924, 0x86, 0xfa, 0x1b, 0x16, 0x25, 0x24, 0xfe, 0xc6	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. Check IP4 Packet InnerHeader is correct.	1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI_SUCCESS and IP4 Packet InnerHeader is correct.
5.25.8.1.14	0xf5503af0, 0x8305, 0x40ce, 0x88, 0xf3, 0x29, 0x1a, 0xe, 0x32, 0x5b, 0x9d	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. Check IP4 Packet PayLoad is intact.	1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI_SUCCESS and Check IP4 Packet PayLoad is intact.
5.25.8.1.15	0x123fa8ee, 0xa9ff, 0x4fa3, 0x92, 0xef, 0x5c, 0x31, 0x60, 0x8c, 0x9e, 0x65	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Tunnel Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC}	1.Call ProcessExt() in Tunnel Mode OutBound Call to do IP4 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC} 2.The return code should be EFI_SUCCESS.
5.25.8.1.16	0xbb52fb61, 0xdba9, 0x45b0, 0x9e, 0xb4, 0x2b, 0xfa, 0x1e, 0xa3, 0xa6, 0xde	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. Check Returned Packet Header is set ZERO	1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be EFI_SUCCESS & Returned Packet Header should be set ZERO.

Number	GUID	Assertion	Test Description
5.25.8.1.17	0x6fc08962, 0xcf2, 0x445b, 0x9f, 0x54, 0x59, 0x12, 0x79, 0xc3, 0xd9, 0x56	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. Check IP4 Packet InnerHeader is correct	1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be EFI_SUCCESS & IP4 Packet InnerHeader is correct.
5.25.8.1.18	0x16dc1d54, 0x755b, 0x482b, 0xa2, 0xca, 0x9d, 0xce, 0xf7, 0xf, 0xa8, 0x8b	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in IPSEC Encrypt & Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC} and Check IP4 Packet PayLoad is intact	1.Call ProcessExt() in Tunnel Mode IPSEC InBound to Decrypt IP4 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be EFI_SUCCESS & IP4 Packet PayLoad is intact.
5.25.8.1.19	0x5c8f633, 0xea97, 0x4c28, 0xb6, 0xf6, 0x4a, 0xa3, 0x8, 0x7c, 0x9b, 0x52	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Transport Mode OutBound Call to do IPSEC IP6 Packet Encrypt Algorithm {SHA1HMAC, 3DESCBC}	1.Call ProcessExt() in Transport Mode OutBound Call to do IPSEC IP6 Packet Encrypt Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI_SUCCESS.
5.25.8.1.20	0x25181e14, 0xb84b, 0x4aae, 0x89, 0xdd, 0x4a, 0xe, 0xe0, 0x27, 0xca, 0xc1	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Transport Mode IPSEC IP6 InBound to Decrypt Algorithm {SHA1HMAC, 3DESCBC}. Check if Packet Header content is intact	1.Call ProcessExt() in Transport Mode IPSEC IP6 InBound to Decrypt Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI_SUCCESS & Packet Header content is intact.
5.25.8.1.21	0xf6ee80b9, 0x622c, 0x4306, 0xae, 0xd2, 0xb6, 0xf8, 0x42, 0x87, 0x92, 0x11	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}. Check if Packet Payload Content is intact	1.Call ProcessExt() in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI_SUCCESS & Packet Payload Content is intact.

Number	GUID	Assertion	Test Description
5.25.8.1.22	0xf251fd3b, 0xf026, 0x4040, 0x8d, 0x8, 0xc9, 0x22, 0x22, 0xaf, 0xe9, 0xbb	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Transport Mode OutBound Call to do IPSEC IP6 packet Encrypt Algorithm {SHA1HMAC, AESCBC}.	1.Call ProcessExt() in Transport Mode OutBound Call to do IPSEC IP6 packet Encrypt Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be EFI_SUCCESS.
5.25.8.1.23	0x5b865ed2, 0x95a6, 0x47bf, 0xbb, 0x35, 0x1a, 0x3b, 0x5, 0x3, 0xb6, 0x80	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. Check if Packet Header content is intact.	1.Call ProcessExt() in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be EFI_SUCCESS & Packet Header content is intact.
5.25.8.1.24	0xed35f3c3, 0x2222, 0x4d4c, 0xb1, 0x16, 0x4c, 0x38, 0x25, 0x29, 0x88, 0x4f	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. Check if Packet Payload Content is intact.	1.Call ProcessExt() in Transport Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be EFI_SUCCESS & Packet Payload Content is intact.
5.25.8.1.25	0xb20f0b, 0xdce8, 0x4c22, 0x98, 0x20, 0xcc, 0xb6, 0x5a, 0x40, 0x14, 0xbe	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Tunnel Mode OutBound Call to do IP6 IPSEC Tunnel Mode Encrypt Algorithm {SHA1HMAC, 3DESCBC}.	1.Call ProcessExt() in Tunnel Mode OutBound to do IP6 IPSEC Tunnel Mode Encrypt Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI_SUCCESS.
5.25.8.1.26	0x52ae482f, 0x4882, 0x4945, 0x88, 0xfd, 0x75, 0xe5, 0x8a, 0x14, 0x4a, 0x4f	EFI_IPSEC2_PROTOCOL. ProcessExt- ProcessExt() returns EFI_SUCCESS in Tunnel Mode IPSEC Tunnel Mode InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}. Check IP6 Packet InnerHeader is correct.	1.Call ProcessExt() in Tunnel Mode IPSEC Tunnel Mode InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be EFI_SUCCESS & IP6 Packet InnerHeader is correct.

Number	GUID	Assertion	Test Description
5.25.8.1.27	0xead97223, 0x1dca, 0x4895, 0xa5, 0x9a, 0xc0, 0x3e, 0x8, 0x80, 0x61, 0x54	`EFI_IPSEC2_PROTOCOL.ProcessExt-ProcessExt()` returns `EFI_SUCCESS` in IPSEC Tunnel Mode InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}. Check IP6 Packet PayLoad is intact.	1.Call `ProcessExt()` in IPSEC Tunnel Mode InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, 3DESCBC}. 2.The return code should be `EFI_SUCCESS` & IP6 Packet PayLoad is intact.
5.25.8.1.28	0xd4f53e8f, 0xe53, 0x44ae, 0xbc, 0xef, 0x7e, 0x28, 0xd2, 0x85, 0xc6, 0xf	`EFI_IPSEC2_PROTOCOL.ProcessExt-ProcessExt()` returns EFI_SUCCESS in Tunnel Mode OutBound Call to do IP6 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC}.	1.Call `ProcessExt()` in Tunnel Mode OutBound Call to do IP6 IPSEC Encrypt Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be `EFI_SUCCESS`.
5.25.8.1.29	0xd96aaf71, 0xca6f, 0x4cc7, 0x89, 0xf4, 0x99, 0x1a, 0xb1, 0xb5, 0x22, 0xe9	`EFI_IPSEC2_PROTOCOL.ProcessExt-ProcessExt()` returns `EFI_SUCCESS` in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. Check Returned Packet Header is set ZERO.	1.Call `ProcessExt()` in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be `EFI_SUCCESS` & Returned Packet Header is set ZERO.
5.25.8.1.30	0xc0ca611c, 0x97bb, 0x4c4e, 0x90, 0x84, 0xff, 0x90, 0x94, 0x20, 0xd9, 0x6e	`EFI_IPSEC2_PROTOCOL.ProcessExt-ProcessExt()` returns `EFI_SUCCESS` in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. Check IP6 Packet InnerHeader is correct.	1.Call `ProcessExt()` in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be `EFI_SUCCESS` & IP6 Packet InnerHeader is correct.
5.25.8.1.31	0x6098f2af, 0xe85c, 0x4201, 0xbb, 0xc9, 0xf9, 0x10, 0x2b, 0xcb, 0x94, 0xe7	`EFI_IPSEC2_PROTOCOL.ProcessExt-ProcessExt()` returns `EFI_SUCCESS` in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. Check IP6 Packet PayLoad is intact.	1.Call `ProcessExt()` in Tunnel Mode IPSEC InBound to Decrypt IP6 packet Algorithm {SHA1HMAC, AESCBC}. 2.The return code should be `EFI_SUCCESS` & IP6 Packet PayLoad is intact.

22 Network Protocols UDP and MTFTP

22.1 EFI_UDP4_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_UDP4_PROTOCOL Section.

22.1.1 GetModeData()

Number	GUID	Assertion	Test Description
5.26.1.1.1	0xfc4d1b7b, 0x4abd, 0x47d3, 0xbd, 0x64, 0xe0, 0x98, 0x86, 0x29, 0x73, 0xec	`EFI_UDP4_PROTOCOL.GetModeData()` – invokes `GetModeData()` to get all mode data before configuration.	1. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_UDP4_PROTOCOL` child. 2. Call `EFI_UDP4_PROTOCOL.GetModeData()` to get all mode data. The return status should be `EFI_NOT_STARTED`. 3. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created `EFI_UDP4_PROTOCOL` child and clean up the environment.
5.26.1.1.2	0x15c32ffb, 0x2cdf, 0x4b5b, 0xab, 0x3e, 0x5a, 0xed, 0x7f, 0xc5, 0x25, 0xe7	`EFI_UDP4_PROTOCOL.GetModeData()` – invokes `GetModeData()` to get `EFI_UDP4_PROTOCOL` mode data before configuration.	1. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_UDP4_PROTOCOL` child. 2. Call `EFI_UDP4_PROTOCOL.GetModeData()` to get `EFI_UDP4_PROTOCOL` mode data. The return status should be `EFI_NOT_STARTED`. 3. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created `EFI_UDP4_PROTOCOL` child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.1.3	0xcdcd0bb9, 0x455a, 0x4525, 0xb8, 0xf2, 0x0e, 0xe0, 0x4b, 0xaa, 0x80, 0x14	**EFI_UDP4_PROTOCOL. GetModeData()** – invokes **GetModeData()** to get **EFI_IP4_PROTOCOL** mode data before configuration.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.GetModeData()** to get **EFI_IP4_PROTOCOL** mode data. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.1.4	0xcc19f3f7, 0x80b9, 0x46e8, 0xb2, 0xaa, 0xb6, 0xdd, 0x81, 0x66, 0xd8, 0x93	**EFI_UDP4_PROTOCOL. GetModeData()** – invokes **GetModeData()** to get **EFI_MANAGED_NETWORK_PROTOCOL** mode data before configuration.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.GetModeData()** to get **EFI_MANAGED_NETWORK_PROTOCOL** mode data. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.1.5	0xd291d441, 0x2d3b, 0x4575, 0xa3, 0xf3, 0x05, 0xe1, 0x5a, 0x34, 0x62, 0xc0	EFI_UDP4_PROTOCOL. GetModeData() – invokes GetModeData() to get EFI_SIMPLE_NETWORK _PROTOCOL mode data before configuration.	1. Call EFI_UDP4_SERVICE_BINDING_P ROTOCOL.CreateChild() to create a new EFI_UDP4_PROTOCOL child. 2. Call EFI_UDP4_PROTOCOL.GetModeDa ta() to get EFI_SIMPLE_NETWORK_PROTOCO L mode data. The return status should be EFI_SUCCESS. 3. Call EFI_UDP4_SERVICE_BINDING_P ROTOCOL.DestroyChild() to destroy the created EFI_UDP4_PROTOCOL child and clean up the environment.
5.26.1.1.6	0xf28448b0, 0xd525, 0x40f7, 0x92, 0xf1, 0xed, 0x6d, 0xaa, 0x59, 0xe4, 0xb4	EFI_UDP4_PROTOCOL. GetModeData() – invokes GetModeData() to get all mode data after configuration.	1. Call EFI_UDP4_SERVICE_BINDING_P ROTOCOL.CreateChild() to create a new EFI_UDP4_PROTOCOL child. 2. Call EFI_UDP4_PROTOCOL.Configur e() to configure the new instance. 3 Call EFI_UDP4_PROTOCOL.GetModeD ata() to get all mode data. The return status should be EFI_SUCCESS. 4. Call EFI_UDP4_SERVICE_BINDING_P ROTOCOL.DestroyChild() to destroy the created EFI_UDP4_PROTOCOL child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.1.7	0x182f712c, 0x1b2a, 0x4850, 0xbd, 0x78, 0xa6, 0xe6, 0xb6, 0xf6, 0x73, 0x54	**EFI_UDP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** to get **EFI_UDP4_PROTOCOL** mode data after configuration.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild**() to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new instance. 3 Call **EFI_UDP4_PROTOCOL.GetModeData()** to get **EFI_UDP4_PROTOCOL** mode data. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.1.8	0x8aa1ebeb, 0xb735, 0x421e, 0x92, 0x1d, 0xf8, 0x76, 0xd2, 0xae, 0xdf, 0x1c	**EFI_UDP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** to get **EFI_IP4_PROTOCOL** mode data after configuration.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild**() to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new instance. 3 Call **EFI_UDP4_PROTOCOL.GetModeData()** to get **EFI_IP4_PROTOCOL** mode data. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.1.9	0xba1c7d49, 0x4490, 0x42e1, 0xa8, 0x92, 0xc3, 0x61, 0xef, 0x5d, 0x94, 0x79	**EFI_UDP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** to get **EFI_MANAGED_NETWORK_PROTOCOL** mode data after configuration.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild**() to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new instance. 3 Call **EFI_UDP4_PROTOCOL.GetModeData()** to get **EFI_MANAGED_NETWORK_PROTOCOL** mode data. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.1.10	0x5df96df3, 0x6404, 0x4486, 0xb6, 0xb7, 0x00, 0xb9, 0x2d, 0x81, 0x21, 0x26	**EFI_UDP4_PROTOCOL.GetModeData()** – invokes **GetModeData()** to get **EFI_SIMPLE_NETWORK_PROTOCOL** mode data after configuration.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild**() to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new instance. 3 Call **EFI_UDP4_PROTOCOL.GetModeData()** to get **EFI_SIMPLE_NETWORK_PROTOCOL** mode data. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

22.1.2 Configure()

Number	GUID	Assertion	Test Description
5.26.1.2.1	0x13a8fd73, 0x6b66, 0x4418, 0x85, 0x4c, 0xda, 0x63, 0xff, 0x42, 0x75, 0x4f	`EFI_UDP4_PROTOCOL. Configure()` – invokes `Configure()` with a *StationAddress* value of a multicast address.	1. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild`() to create a new `EFI_UDP4_PROTOCOL` child. 2. Call `EFI_UDP4_PROTOCOL.Configure()` to configure the new `EFI_UDP4_PROTOCOL` instance with a *StationAddress* value of a multicast address. The return status should be `EFI_INVALID_PARAMETER`. 3. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created `EFI_UDP4_PROTOCOL` child and clean up the environment.
5.26.1.2.2	0xd8b6f8bd, 0x1ba8, 0x48c1, 0x90, 0x30, 0x5a, 0x37, 0x18, 0x0c, 0x06, 0x01	`EFI_UDP4_PROTOCOL. Configure()` – invokes `Configure()` with an invalid *SubnetMask* value.	1. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild`() to create a new `EFI_UDP4_PROTOCOL` child. 2. Call `EFI_UDP4_PROTOCOL.Configure()` to configure the new `EFI_UDP4_PROTOCOL` instance with an invalid *SubnetMask* value. The return status should be `EFI_INVALID_PARAMETER`. 3. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created `EFI_UDP4_PROTOCOL` child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.2.3	0xb4a98a30, 0x35e9, 0x4460, 0x81, 0x5d, 0x42, 0x33, 0x7c, 0x17, 0x6c, 0x44	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** to reconfigure the *AcceptPromiscuous* before the instance has been stopped or reset.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild**() to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Configure()** to reconfigure the *AcceptPromiscuous*. The return status should be **EFI_ALREADY_STARTED**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.2.23	0x349fc21a, 0x37db, 0x406e, 0xbd, 0xc8, 0xf6, 0x12, 0x2c, 0xa9, 0xe9, 0xfc	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** with the parameter *RemoteAddress* being a multicast address.The return status should be **EFI_INVALID_PARAMETER**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild**() to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** with the parameter *RemoteAddress* being a multicast address.The return status should be **EFI_INVALID_PARAMETER**. 3. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.2.4	0xa36f507b, 0x7526, 0x441e, 0xaf, 0x48, 0x4a, 0xc4, 0xf4, 0x31, 0xe6, 0xbd	**EFI_UDP4_PROTOCOL.Configure() –** invokes **Configure()** to reconfigure the *AcceptBroadcast* before the instance has been stopped or reset.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Configure()** to reconfigure the *AcceptBroadcast*. The return status should be **EFI_ALREADY_STARTED**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.2.5	0xac4cf23e, 0x0c5e, 0x4299, 0xb4, 0x29, 0xc8, 0x83, 0xe7, 0xe6, 0x73, 0xb8	**EFI_UDP4_PROTOCOL.Configure() –** invokes **Configure()** to reconfigure the *AcceptAnyPort* before the instance has been stopped or reset.	**1. Call EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Configure()** to reconfigure the *AcceptAnyPort*. The return status should be **EFI_ALREADY_STARTED**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.2.6	0xc08bfbab, 0x0cde, 0x4332, 0x86, 0x86, 0x42, 0x52, 0xdc, 0x50, 0x48, 0xcc	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** to reconfigure the *AllowDuplicatePort* before the instance has been stopped or reset.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Configure()** to reconfigure the *AllowDuplicatePort*. The return status should be **EFI_ALREADY_STARTED**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.2.7	0x66544950, 0x16ff, 0x4854, 0x9c, 0x09, 0x45, 0x84, 0x29, 0x2d, 0x7c, 0x51	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** to reconfigure the *UseDefaultAddress* before the instance has been stopped or reset.	**1. Call EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Configure()** to reconfigure the *UseDefaultAddress*. The return status should be **EFI_ALREADY_STARTED**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.2.24	0xee87c393, 0xf728, 0x46b9, 0xb1, 0x31, 0x58, 0xc3, 0xdd, 0x5e, 0x18, 0x34	**EFI_UDP4_PROTOCOL. Configure()** – invokes **Configure()** when *UdpConfigData.AllowDuplicatePort is* **FALSE** and *UdpConfigData.StationPort* is already used by other instance.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** when *UdpConfigData.AllowDuplicatePort* is **FALSE** and **UdpConfigData.StationPort** is already used by other instance. The return status should be **EFI_ACCESS_DENIED**. 3. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.2.8	0xbe8ab604, 0x1c84, 0x4a80, 0xb6, 0x9a, 0x43, 0xfd, 0xf8, 0x94, 0x5e, 0xf2	**EFI_UDP4_PROTOCOL. Configure()** – invokes **Configure()** to test the function of transmitting a packet.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** to transmit a packet and verify if it is successful. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.2.9	0xddbba5ba, 0x678b, 0x426e, 0x87, 0xa8, 0x8c, 0x1b, 0xde, 0x5b, 0x36, 0x96	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** to test that function of receiving a packet.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Receive()** to receive a packet and verify if it is successful. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.2.10	0xefe91110, 0x4e6e, 0x4e07, 0xa7, 0xec, 0x09, 0x74, 0xb7, 0xe3, 0x03, 0x87	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** to reconfigure the *TypeOfService* before the instance has been stopped or reset.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Configure()** to reconfigure the *TypeOfService*. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.2.11	0xc6f4f65f, 0x9a98, 0x4d6e, 0xaf, 0xae, 0xe9, 0x87, 0xf9, 0xb4, 0xb4, 0x9c	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** to reconfigure the *TimeToLive* before the instance has been stopped or reset.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Configure()** to reconfigure the *TimeToLive*. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.2.12	0xe6313038, 0x43f2, 0x4cbe, 0xb8, 0x61, 0xa4, 0x1b, 0x6e, 0x3d, 0x58, 0x91	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** to reconfigure the *DoNotFragment* before the instance has been stopped or reset.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Configure()** to reconfigure the *DoNotFragment*. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.2.13	0x2c81abe0, 0xcf2a, 0x42d0, 0xb4, 0xe3, 0x59, 0x9e, 0x9e, 0x2f, 0x60, 0x6a	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** to reconfigure the *ReceiveTimeout* before the instance has been stopped or reset.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Configure()** to reconfigure the *ReceiveTimeout*. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.2.14	0x798d02e5, 0x0810, 0x462c, 0x8f, 0xba, 0xe9, 0x32, 0xfb, 0x9d, 0x84, 0x85	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** to reconfigure the *TransmitTimeout* before the instance has been stopped or reset.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Configure()** to reconfigure the *TransmitTimeout*. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.2.15	0xbe426d4c, 0x8242, 0x4a4e, 0x8d, 0x7d, 0x58, 0xe0, 0x93, 0x92, 0x77, 0x7c	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** with the parameter *AcceptBroadcast* set to **FALSE**. Check that it can not receive broadcast packet.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance with the parameter *AcceptBroadcast* set to **FALSE**. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Receive()** and check it can not receive broadcast packet. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.2.16	0xb50d8d35, 0xc0c9, 0x4955, 0x94, 0x13, 0xf7, 0x0a, 0x39, 0x2d, 0xa3, 0x0f	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** with the parameter *AcceptBroadcast* set to **TRUE**. Check that it can receive broadcast packet successfully.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance with the parameter *AcceptBroadcast* set to **TRUE**. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Receive()** and check that it can receive broadcast packet successfully. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.2.17	0x4881a297, 0x3afc, 0x4324, 0xa5, 0x8f, 0xcb, 0x02, 0x64, 0xe5, 0xbd, 0x5e	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** with the parameter *AcceptPromiscuous* set to **FALSE**. Check that it can not receive packet to other unicast MACs than its own.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance with the parameter *AcceptPromiscuous* set to **FALSE**. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Receive()** and check that it can not receive packet to other unicast MACs than its own. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.2.18	0x066131ca, 0xa6e4, 0x478b, 0x9a, 0xca, 0x05, 0x93, 0xfc, 0xc7, 0xfd, 0x4b	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** with the parameter *AcceptPromiscuous* set to **TRUE**. Check that it can receive packet to other unicast MACs than its own.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance with the parameter *AcceptPromiscuous* set to **TRUE**. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Receive()** and check that it can receive packet to other unicast MACs than its own. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.2.19	0x2867badf, 0x1696, 0x40a1, 0xb8, 0x40, 0x00, 0x4c, 0x79, 0xed, 0xc7, 0xf3	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** with the parameter *AcceptAnyPort* set to **FALSE**. Check that it can not receive packet to other port.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance with the parameter *AcceptAnyPort* set to **TRUE**. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Receive()** and check that it can not receive packet to other port. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.2.20	0x17d43b3d, 0x9187, 0x4515, 0x83, 0x94, 0x13, 0xdf, 0xf9, 0x35, 0xf4, 0x9e	**EFI_UDP4_PROTOCOL. Configure() –** invokes **Configure()** with the parameter *AcceptAnyPort* set to **TRUE**. Check that it can receive packet to other port.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance with the parameter *AcceptAnyPort* set to **TRUE**. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Receive()** and check that it can receive packet to other port. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.2.21	0x08c86675, 0x7018, 0x418d, 0xb4, 0x3d, 0x36, 0xdc, 0xc5, 0x8b, 0xdc, 0x88	**EFI_UDP4_PROTOCOL.Configure()** – invokes **Configure()** to check if the parameter *TypeOfService* can effect the sending out of the packet.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance with the parameter *TypeOfService* set to 1. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** and check that it can transmit the packet successfully. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.2.22	0x4fb07a34, 0xc2ab, 0x40c1, 0x8a, 0x26, 0x42, 0x6d, 0x54, 0x32, 0x3a, 0xa4	**EFI_UDP4_PROTOCOL.Configure()** – invokes **Configure()** to check if the parameter *TimeToLive* can effect the sending out of the packet .	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance with the parameter *TimeToLive* set to 111. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** and check that it can transmit the packet successfully. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

22.1.3 Groups()

Number	GUID	Assertion	Test Description
5.26.1.3.1	0x90ff05c9, 0xea78, 0x4359, 0x95, 0xc0, 0x4d, 0x09, 0x7b, 0xa2, 0xcf, 0x14	**EFI_UDP4_PROTOCOL.Groups() –** invokes **Groups()** with a *JoinFlag* value of **TRUE** and a *MulticastAddress* value of **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Groups()** with a *JoinFlag* value of **TRUE** and a *MulticastAddress* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.3.2	0x4e1cabfe, 0x2dda, 0x4e0c, 0xbd, 0xbc, 0x5f, 0xfc, 0x77, 0x42, 0xf8, 0x0f	**EFI_UDP4_PROTOCOL.Groups() –** invokes **Groups()** with a *JoinFlag* value of **TRUE** and a **MulticastAddress* value of an invalid multicast address.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Groups()** with a *JoinFlag* value of **TRUE** and a **MulticastAddress* value of an invalid multicast address. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.3.3	0xf1018cf8, 0xd8ba, 0x4fa1, 0x82, 0xec, 0x64, 0x52, 0x06, 0x9a, 0x4a, 0xa7	**EFI_UDP4_PROTOCOL.Groups() –** invokes **Groups()** when the group address is not in the group table ,while *JoinFlag* is **FALSE**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Groups()** to join a group address into the group table. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_PROTOCOL.Groups()** to leave an group address which is not in the group table The return status should be **EFI_NOT_FOUND**. 5. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.3.8	0x86b5bd38, 0x04ae, 0x4a44, 0xbe, 0x0d, 0x1d, 0x7f, 0x32, 0x0f, 0x46, 0xf8	**EFI_UDP4_PROTOCOL.Groups() –** invokes **Groups()** when the EFI UDPv4 protocol instance has not been started.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Groups()** when the EFI UDPv4 protocol instance has not been started. 3. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.3.9	0xde218295, 0x6dec, 0x4c7f, 0x8c, 0x02, 0xc9, 0x46, 0xea, 0x64, 0x59, 0xd6	**EFI_UDP4_PROTOCOL.Groups() –** invokes **Groups()** when the group address is already in the group table when *JoinFlag* is **FALSE.**	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Groups()** to join a group address into the group table. The return status should be **EFI_SUCCESS**. 3. Call **EFI_UDP4_PROTOCOL.Groups()** when the group address is already in the group table when *JoinFlag* is **FALSE.** The return status should be **EFI_ALREADY_STARTED**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.3.4	0x101a001f, 0x547e, 0x4e1b, 0xae, 0xf6, 0x7d, 0x35, 0x27, 0xb1, 0x23, 0x6f	**EFI_UDP4_PROTOCOL.Groups() –** invokes **Groups()** to join a group address and call **Receive()** to check that it can receive UDP packets to the group IP.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Groups()** to join a group address into the group table. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_PROTOCOL.Receive()** to receive the packets and check that it is successful. 5. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.3.5	0x22561bd0, 0x47ba, 0x4240, 0x96, 0x3a, 0x2a, 0xaf, 0x83, 0x5b, 0xda, 0x72	`EFI_UDP4_PROTOCOL.Groups()` – invokes `Groups()` to join two multicast group address and call `Receive()` to check if it can receive UDP packets to either of the groups.	1. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_UDP4_PROTOCOL` child. 2. Call `EFI_UDP4_PROTOCOL.Configure()` to configure the new `EFI_UDP4_PROTOCOL` instance. 3. Call `EFI_UDP4_PROTOCOL.Groups()` to join two group address into the group table. The return status should be `EFI_SUCCESS`. 4. Call `EFI_UDP4_PROTOCOL.Receive()` to check it can receive UDP packets to either of the groups. 5. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created `EFI_UDP4_PROTOCOL` child and clean up the environment.
5.26.1.3.6	0x7fcefed3, 0x6e40, 0x4ed8, 0xa4, 0x41, 0x83, 0x7f, 0x5e, 0x13, 0x06, 0x62	`EFI_UDP4_PROTOCOL.Groups()` – invokes `Groups()` to leave a specified group.	1. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_UDP4_PROTOCOL` child. 2. Call `EFI_UDP4_PROTOCOL.Configure()` to configure the new `EFI_UDP4_PROTOCOL` instance. 3. Call `EFI_UDP4_PROTOCOL.Groups()` to join a specified group address into the group table. The return status should be `EFI_SUCCESS`. 4. Call `EFI_UDP4_PROTOCOL.Groups()` to leave the group joined in step 3. The return status should be `EFI_SUCCESS`. 5. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created `EFI_UDP4_PROTOCOL` child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.3.7	0x06e97222, 0x1858, 0x469a, 0xa8, 0x19, 0x25, 0xd7, 0x1a, 0x15, 0xc3, 0x68	**`EFI_UDP4_PROTOCOL.Groups()`** **–** invokes **`Groups()`** to leave all multicast groups with a *MulticastAddress* value of **`NULL`**.	1. Call **`EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new **`EFI_UDP4_PROTOCOL`** child. 2. Call **`EFI_UDP4_PROTOCOL.Configure()`** to configure the new **`EFI_UDP4_PROTOCOL`** instance. 3. Call **`EFI_UDP4_PROTOCOL.Groups()`** to join two group address into the group table. The return status should be **`EFI_SUCCESS`**. 4. Call **`EFI_UDP4_PROTOCOL.Groups()`** to leave all multicast groups with a *`MulticastAddress`* value of **`NULL`**. The return status should be **`EFI_SUCCESS`**. 5. Call **`EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the created **`EFI_UDP4_PROTOCOL`** child and clean up the environment.

22.1.4 Routes()

Number	GUID	Assertion	Test Description
5.26.1.4.1	0xedcd02f7, 0x3b78, 0x4186, 0x9d, 0x14, 0x52, 0x92, 0x6b, 0x85, 0x73, 0x08	**EFI_UDP4_PROTOCOL.Routes()** – invokes **Routes()** with a *SubnetAddress* value of **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Routes()** with a *SubnetAddress* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.4.2	0xf0bedea5, 0x05bf, 0x4ab9, 0x89, 0xb3, 0xdf, 0xd9, 0x8e, 0x08, 0xe4, 0xdd	**EFI_UDP4_PROTOCOL.Routes()** – invokes **Routes()** with a *SubnetMask* value of **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Routes()** with a *SubnetMask* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.4.3	0x377694cc, 0x9254, 0x4197, 0x92, 0x6c, 0x26, 0x58, 0x5c, 0xde, 0xc9, 0x4c	**EFI_UDP4_PROTOCOL.Routes()** – invokes **Routes()** with a *GatewayAddress* value of **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Routes()** with a *GatewayAddress* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.4.4	0xc694ffe9, 0xef16, 0x47f4, 0x86, 0x89, 0x34, 0x6c, 0x80, 0xb1, 0x59, 0x54	**EFI_UDP4_PROTOCOL.Routes()** – invokes **Routes()** with a **SubnetMask* value of an invalid subnet mask.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Routes()** with a **SubnetMask* value of an invalid subnet mask. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.4.5	0x601c9a17, 0x1da6, 0x45bc, 0xbb, 0xdc, 0xf8, 0x92, 0xdc, 0xe3, 0x43, 0x04	**EFI_UDP4_PROTOCOL.Routes()** – invokes **Routes()** with a **GatewayAddress* value of an invalid unicast IP address.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Routes()** with a **GatewayAddress* value of an invalid unicast IP address. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.4.6	0xab7d87d5, 0x9761, 0x4877, 0x9f, 0x96, 0x42, 0xab, 0x99, 0x66, 0xd5, 0x3f	**EFI_UDP4_PROTOCOL.Routes()** – invokes **Routes()** to delete a route which is not in the routing table.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Routes()** to delete a route which is not in the routing table. The return status should be **EFI_NOT_FOUND**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.4.7	0x72569926, 0x4edb, 0x4d5b, 0xa2, 0xe5, 0x76, 0x31, 0x2f, 0xd2, 0x76, 0x74	**EFI_UDP4_PROTOCOL.Routes()** – invokes **Routes()** to add a route that has already defined in the routing table.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Routes()** to add a route into the routing table. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_PROTOCOL.Routes()** to add the route into the routing table which has already defined in step 3. The return status should be **EFI_ACCESS_DENIED**. 5. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.4.10	0xe9ff9948, 0x9168, 0x4698, 0xa1, 0x49, 0x44, 0xef, 0x57, 0x33, 0x77, 0x20	**EFI_UDP4_PROTOCOL.Routes()** – invokes **Routes()** when the EFI UDPv4 Protocol instance has not been started.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 3. Call **EFI_UDP4_PROTOCOL.Routes()** when the EFI UDPv4 Protocol instance has not been started. The return status should be **EFI_NOT_STARTED**. 3. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.4.8	0xae5c33be, 0x930e, 0x401b, 0x8f, 0x4d, 0x32, 0xc8, 0x95, 0xc4, 0x55, 0x48	`EFI_UDP4_PROTOCOL.Routes()` – invokes `Routes()` to add a route to destination IP and send a packet to it.	1. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_UDP4_PROTOCOL` child. 2. Call `EFI_UDP4_PROTOCOL.Configure()` to configure the new `EFI_UDP4_PROTOCOL` instance. 3. Call `EFI_UDP4_PROTOCOL.Routes()` to add a route into the routing table. The return status should be `EFI_SUCCESS`. 4. Call `EFI_UDP4_PROTOCOL.Transmit()` to send a packet to the destination IP and check that it is successful. 5. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created `EFI_UDP4_PROTOCOL` child and clean up the environment.
5.26.1.4.9	0xd39800b9, 0xe6e6, 0x4e29, 0xab, 0xd6, 0x17, 0x7a, 0x46, 0x10, 0x51, 0x3d	`EFI_UDP4_PROTOCOL.Routes()` – invokes `Routes()` to delete a route to destination IP and check that packet can not been sent to it.	1. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_UDP4_PROTOCOL` child. 2. Call `EFI_UDP4_PROTOCOL.Configure()` to configure the new `EFI_UDP4_PROTOCOL` instance. 3. Call `EFI_UDP4_PROTOCOL.Routes()` to add a route into the routing table. The return status should be `EFI_SUCCESS`. 4. Call `EFI_UDP4_PROTOCOL.Routes()` to delete the route added in the step 3. The return status should be `EFI_SUCCESS`. 5. Call `EFI_UDP4_PROTOCOL.Transmit()` to send a packet to the destination IP and check that it will be failed. 6. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created `EFI_UDP4_PROTOCOL` child and clean up the environment.

22.1.5 Transmit()

Number	GUID	Assertion	Test Description
5.26.1.5.1	0xd793cd46, 0x574d, 0x4f5d, 0x92, 0x8a, 0x2b, 0x84, 0x7a, 0xc0, 0x77, 0xd9	**EFI_UDP4_PROTOCOL.Transmit()** – invokes **Transmit()** with a *Token* value of **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** with a *Token* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.5.2	0xf8ffef65, 0x20fe, 0x4381, 0xa5, 0x46, 0x07, 0x7c, 0x5a, 0x89, 0x7b, 0x6d	**EFI_UDP4_PROTOCOL.Transmit()** – invokes **Transmit()** with a *Token.Event* value of **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** with a *Token.Event* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.5.3	0x157caa4e, 0xa260, 0x47a2, 0x97, 0x04, 0xd6, 0x62, 0x6c, 0xd9, 0x62, 0xf9	**EFI_UDP4_PROTOCOL.Transmit()** – invokes **Transmit()** with a *Token*.Packet.FragmentCount value of 0.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** with a *Token*.Packet.FragmentCount value of 0. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.5.4	0xceebb331, 0x26c1, 0x4c6b, 0x91, 0x74, 0xb2, 0xdd, 0xda, 0xb7, 0x3a, 0x7a	**EFI_UDP4_PROTOCOL.Transmit()** – invokes **Transmit()** with a *Token.Packet.TxData* value of **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** with a *Token.Packet.TxData* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.5.5	0xd381956d, 0x6b86, 0x48a4, 0x82, 0x56, 0x37, 0x5e, 0xa2, 0x46, 0xf6, 0xfa	**EFI_UDP4_PROTOCOL.Transmit()** – invokes **Transmit()** with a *Token*.**Packet.DataLength** value other than equal to the sum of fragment lengths.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** with a *Token*.**Packet.DataLength** value other than equal to the sum of fragment lengths. The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.5.7	0x7f9fc4ec, 0x756c, 0x4399, 0xa2, 0x7e, 0x2e, 0x38, 0x3a, 0xff, 0x4e, 0x7b	**EFI_UDP4_PROTOCOL.Transmit()** – invokes **Transmit()** with the parameter *Token.Packet.TxData.FragmentTable[].FragmentLenth/FragmetBuffer* fields being invalid.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** with the parameter *Token.Packet.TxData.FragmentTable[].FragmentLenth* fields being zero. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_PROTOCOL.Transmit()** with the parameter *Token.Packet.TxData.FragmentTable[].FragmentBuffer* fields being **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 5. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.5.8	0x5d755449, 0x3840, 0x4cc8, 0x9c, 0x7f, 0x3a, 0x1a, 0xf3, 0x42, 0xd2, 0x89	**EFI_UDP4_PROTOCOL.Transmit() –** invokes **Transmit()** with the parameter *Token.Packet.TxData.GatewayAddress* being not a valid unicast IPv4 address if it is not **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** with the parameter *Token.Packet.TxData.GatewayAddress* being 255.255.255.255. The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.5.9	0x411080da, 0x2db4, 0x415e, 0xa0, 0xf5, 0x72, 0xf4, 0x1e, 0x55, 0x38, 0xdb	**EFI_UDP4_PROTOCOL.Transmit() –** invokes **Transmit()** with the parameter *Token.Packet.TxData.GatewayAddress* being not a valid unicast IPv4 address if it is not **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** with the parameter *Token.Packet.TxData.GatewayAddress* being 172.16.220.255. The return status should be **EFI_INVALID_PARAMETER.** 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.5.10	0x154ee561, 0x041a, 0x4e4b, 0x96, 0x3a, 0xfd, 0xc6, 0x4c, 0x4e, 0x3f, 0x29	**EFI_UDP4_PROTOCOL.Transmit() –** invokes **Transmit()** with the parameter *Token.Packet.TxData.GatewayAddress* being not a valid unicast IPv4 address if it is not **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** with the parameter *Token.Packet.TxData.GatewayAddress* being 224.0.0.2. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.5.11	0x0161be6a, 0x75d4, 0x444b, 0xaf, 0x31, 0x78, 0xa4, 0xf0, 0x65, 0xed, 0x43	**EFI_UDP4_PROTOCOL.Transmit() –** invokes **Transmit()** with *Token.Packet.TxData.UdpSessionData* being not valid unicast IPv4 addresses if it is not **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** with the *Token.Packet.TxData.UdpSessionData* being 224.0.0.1. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.5.12	0x3315e964, 0xc1bb, 0x4984, 0xb7, 0xc3, 0xff, 0x1a, 0x94, 0xb0, 0xe9, 0xd3	**EFI_UDP4_PROTOCOL.Transmit()** – invokes **Transmit()** with *Token.Packet.TxData.UdpSessionData* being not valid unicast IPv4 addresses if it is not **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** with the *Token.Packet.TxData.UdpSessionData* being 172.16.220.0. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.5.13	0x4206d340, 0xe096, 0x4369, 0x96, 0x32, 0x9a, 0x35, 0x27, 0xcf, 0x64, 0xce	**EFI_UDP4_PROTOCOL.Transmit()** – invokes **Transmit()** when the EFI UDPv4 Protocol instance has not been started.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Transmit()** when the EFI UDPv4 Protocol instance has not been started. The return status should be **EFI_NOT_STARTED.** 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.5.14	0xbd543b46, 0xcb6a, 0x4cfb, 0x80, 0x68, 0xe1, 0xaa, 0x28, 0x32, 0x43, 0x75	**EFI_UDP4_PROTOCOL.Transmit() –** invokes **Transmit()** when there is no route to the destination network or address.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** when there is no route to the destination network or address. The return status should be **EFI_NOT_FOUND.** 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.5.15	0x0b3c198b, 0xfffd, 0x4dde, 0x9b, 0x1e, 0xbd, 0x5f, 0x8e, 0x70, 0xa0, 0xc2	**EFI_UDP4_PROTOCOL.Transmit() –** invokes **Transmit()** when the data length is greater than the maximum UDP packet size.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** when the data length is greater than the maximum UDP packet size. The return status should be **EFI_BAD_BUFFER_SIZE.** 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.5.16	0xae0d4495, 0xbcda, 0x4de3, 0xa4, 0xbc, 0xab, 0xed, 0xd4, 0x82, 0xdc, 0x92	**EFI_UDP4_PROTOCOL.Transmit()** – invokes **Transmit()** when the length of the IP header+UDP header+data length is greater than MTU if *DoNotFragment* is **TRUE**	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** when the length of the IP header+UDP header+data length is greater than MTU if *DoNotFragment* is **TRUE.** The return status should be **EFI_BAD_BUFFER_SIZE.** 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.5.17	0xd983be7a, 0x33fd, 0x4308, 0x80, 0x6c, 0x00, 0x58, 0xef, 0xff, 0xe8, 0x17	**EFI_UDP4_PROTOCOL.Transmit()** – to add a route to destination IP and send a packet to it.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Routes()** to add a route. 4. Call **EFI_UDP4_PROTOCOL.Transmit()** to transmit packet. 5. Captured packet and verify. 6. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.5.18	0x71158c72, 0xa476, 0x42a8, 0x94, 0x81, 0x6d, 0xa0, 0xb8, 0xb4, 0x2c, 0xef	**EFI_UDP4_PROTOCOL.Transmit()** – invokes **Transmit()** when the *TxData.GatewayAddress* has been set	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()**to transmit packet. 4. Captured packet and verify. 5. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.5.6	0xc0c68374, 0x0d85, 0x4bbb, 0x8b, 0x20, 0xbd, 0x88, 0xb1, 0xb0, 0x7b, 0xd7	**EFI_UDP4_PROTOCOL.Transmit()** – invokes **Transmit()** with the transmit completion token whose *Token.Event* was already in the transmit queue.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Transmit()** to transmit a packet. The return status should be **EFI_SUCCESS**. 4. Call **EFI_UDP4_PROTOCOL.Transmit()** with the same *Token.Event* in step 3. The return status should be **EFI_ACCESS_DENIED.** 5. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

22.1.6 Receive()

Number	GUID	Assertion	Test Description
5.26.1.6.1	0x95bf8134, 0x5277, 0x413c, 0xbe, 0x1f, 0xf5, 0x03, 0x2b, 0x08, 0x78, 0x92	**EFI_UDP4_PROTOCOL.Receive()** – invokes **Receive()** with a *Token* value of **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Receive()** with a *Token* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.6.2	0xa158c0cd, 0x496b, 0x4dfe, 0x9c, 0xe9, 0x93, 0xea, 0x76, 0x40, 0x77, 0x7a	**EFI_UDP4_PROTOCOL.Receive()** – invokes **Receive()** with a *Token.Event* value of **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Receive()** with a *Token.Event* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.6.3	0xdd8e13d5, 0x7a76, 0x4237, 0x82, 0x14, 0x79, 0x03, 0xda, 0x61, 0x92, 0x4d	**EFI_UDP4_PROTOCOL.Receive()** – invokes **Receive()** when the EFI UDPv4 Protocol instance has not been started.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Receive()** when the EFI UDPv4 Protocol instance has not been started. The return status should be **EFI_NOT_STARTED.** 3. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.6.4	0xe2a9f6b9, 0x7827, 0x474f, 0x97, 0x12, 0xc6, 0x9c, 0xad, 0xb0, 0x1c, 0x49	**EFI_UDP4_PROTOCOL.Receive()** – invokes **Receive()** when a receive completion token with the same *Token.Event* was already in the receive queue.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Receive()** at the first time the return status should be **EFI_SUCCESS.** 3. Call **EFI_UDP4_PROTOCOL.Receive()** again the return status should be EFI_ACCESS_DENIED. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.
5.26.1.6.5	0xa96aa0f5, 0x1c6b, 0x41cf, 0x98, 0x2f, 0xf8, 0x4f, 0x90, 0x43, 0x34, 0xb3	**EFI_UDP4_PROTOCOL.Receive()** – the receiving fails because an ICMP error packet is received.	1. Create a NETWORK unreachable packet. 2. Call **EFI_UDP4_PROTOCOL.Receive()** to receive the packet. The return status should be **EFI_SUCCESS.** 3. Verify the R_*Token*.Status it should be EFI_NETWORK_UNREACHABLE.

Number	GUID	Assertion	Test Description
5.26.1.6.6	0x3db8e8ee, 0x6c0b, 0x43d2, 0xa5, 0xfe, 0xb2, 0x34, 0x30, 0x5c, 0x12, 0xf8	**EFI_UDP4_PROTOCOL.Receive()** – the receiving fails because an ICMP error packet is received.	1. Create a HOST unreachable packet. 2. Call **EFI_UDP4_PROTOCOL.Receive()** to receive the packet. The return status should be **EFI_SUCCESS.** 3. Verify the R_*Token*.Status it should be **EFI_HOST_UNREACHABLE**.
5.26.1.6.7	0x26f533d1, 0xb63e, 0x4997, 0xbd, 0x2d, 0x68, 0x52, 0xc8, 0x0c, 0xe3, 0x71	**EFI_UDP4_PROTOCOL.Receive()** – the receiving fails because an ICMP error packet is received.	1. Create a PROTOCOL error packet. 2. Call **EFI_UDP4_PROTOCOL.Receive()** to receive the packet. The return status should be **EFI_SUCCESS.** 3. Verify the R_*Token*.Status it should be **EFI_PROTOCOL_UNREACHABLE**.
5.26.1.6.8	0xc982e2f7, 0xdf6f, 0x4a7b, 0x9d, 0x4a, 0x25, 0x87, 0x0c, 0x80, 0xb7, 0x9b	**EFI_UDP4_PROTOCOL.Receive()** – the receiving fails because an ICMP error packet is received.	1. Create a PORT unreachable packet. 2. Call **EFI_UDP4_PROTOCOL.Receive()** to receive the packet. The return status should be **EFI_SUCCESS.** 3. Verify the R_*Token*.Status it should be **EFI_PORT_UNREACHABLE**.
5.26.1.6.9	0x0685647b, 0xeee8, 0x4756, 0xbf, 0xea, 0x72, 0xc6, 0xb5, 0xff, 0x98, 0xb6	**EFI_UDP4_PROTOCOL.Receive()** – the receiving fails because an ICMP error packet is received.	1. Create a TCMP error packet. 2. Call **EFI_UDP4_PROTOCOL.Receive()** to receive the packet. The return status should be **EFI_SUCCESS.** 3. Verify the R_*Token*.Status it should be **EFI_ICMP_ERROR**.

22.1.7 Cancel()

Number	GUID	Assertion	Test Description
5.26.1.7.1	0xb4ca8ee0, 0x2b8b, 0x41b3, 0x97, 0x3c, 0x2f, 0x2b, 0x05, 0x07, 0x48, 0x17	`EFI_UDP4_PROTOCOL.Cancel()` – invokes `Cancel()` to cancel a receive request while it has been completed.	1. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_UDP4_PROTOCOL` child. 2. Call `EFI_UDP4_PROTOCOL.Configure()` to configure the new `EFI_UDP4_PROTOCOL` instance. 3. Call `EFI_UDP4_PROTOCOL.Receive()` to receive a packet and check that it is successful. 4. Call `EFI_UDP4_PROTOCOL.Cancel()` to cancel the receive request while the token has been completed. The return status should be `EFI_NOT_FOUND`. 5. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created `EFI_UDP4_PROTOCOL` child and clean up the environment.
5.26.1.7.2	0x46a1ec38, 0x0183, 0x485a, 0xa2, 0xa5, 0x50, 0x4e, 0x3b, 0xdb, 0x1b, 0x53	`EFI_UDP4_PROTOCOL.Cancel()` – invokes `Cancel()` to cancel a receive request.	1. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_UDP4_PROTOCOL` child. 2. Call `EFI_UDP4_PROTOCOL.Configure()` to configure the new `EFI_UDP4_PROTOCOL` instance. 3. Call `EFI_UDP4_PROTOCOL.Receive()` to receive a packet. 4. Call `EFI_UDP4_PROTOCOL.Cancel()` to cancel the receive request in step 3. Then check the packet sent to EUT will not be captured. 5. Call `EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created `EFI_UDP4_PROTOCOL` child and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.1.7.3	0x6fff20b8, 0x55cd, 0x4610, 0xb3, 0xbe, 0xaa, 0x19, 0x5f, 0x29, 0x10, 0x66	**EFI_UDP4_PROTOCOL.Cancel() –** invokes **Cancel()** to cancel all pending tokens with the parameter *Token* set to **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. 3. Call **EFI_UDP4_PROTOCOL.Receive()** to set two requests in the receive queue. 4. Call **EFI_UDP4_PROTOCOL.Cancel()** with the parameter *Token* set to **NULL**. Then check that no packet sent to EUT will be captured. 5. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

22.1.8 Poll()

Number	GUID	Assertion	Test Description
5.26.1.8.1	0x18e54eae, 0x4d67, 0x468c, 0xb6, 0x0d, 0x81, 0x83, 0xd4, 0x07, 0xfe, 0xe8	**EFI_UDP4_PROTOCOL.Poll()** – invokes **Poll()** when the instance has not been started.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_PROTOCOL.Configure()** to configure the new **EFI_UDP4_PROTOCOL** instance. Then call **EFI_IP4_PROTOCOL.Configure()** again with an *IpConfigData* value of **NULL**. 3. Call **EFI_IP4_PROTOCOL.Poll()** for incoming data packets and processing outgoing data packets. The return status should be **EFI_NOT_STARTED**. 4. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

22.1.9 CreateChild()

Number	GUID	Assertion	Test Description
5.26.1.9.1	0xf88eaa0c, 0x764e, 0x45e0, 0x95, 0x86, 0xa6, 0x7f, 0x7d, 0x6f, 0xb2, 0x82	**EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** – invokes CreateChild() with a *ChildHandle* value of **NULL**.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child with a *ChildHandle* value of **NULL**. The return status should be **EFI_INVALID_PARAMETER**.
5.26.1.9.2	0x4dedef14, 0xbcba, 0x4b26, 0xbc, 0xc8, 0xb4, 0x7f, 0x8c, 0x08, 0xc9, 0x9d	**EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** – invokes CreateChild() to create three instances.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create three **EFI_UDP4_PROTOCOL** instances and configure them. The return status should be **EFI_SUCCESS**. 2. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment.

22.1.10 DestroyChild()

Number	GUID	Assertion	Test Description
5.26.1.10.1	0x0ff5f5a1, 0x4d29, 0x40ae, 0xa4, 0xef, 0x02, 0x3b, 0xd3, 0xb8, 0x2e, 0x8c	**EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** – invokes **DestroyChild()** with an invalid *ChildHandle* value.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** with an invalid *ChildHandle* value. The return status should be **EFI_INVALID_PARAMETER**.
5.26.1.10.3	0x9d888685, 0xfde7, 0x4832, 0xbc, 0x95, 0x03, 0xd6, 0x44, 0xc6, 0x29, 0xc5	**EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** – invokes **DestroyChild()** to destroy an existed child twice.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_UDP4_PROTOCOL** child. 2. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL.** The return status should be **EFI_SUCCESS.** 2. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** again. The return status should be EFI_UNSUPPORTED.
5.26.1.10.2	0x1ff85dcf, 0x885e, 0x42bf, 0x80, 0xd8, 0xf8, 0x4a, 0xaf, 0x11, 0xeb, 0x77	**EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** – invokes **DestroyChild()** to destroy a child.	1. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a **EFI_UDP4_PROTOCOL** child. The return status should be **EFI_SUCCESS**. 2. Call **EFI_UDP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the created **EFI_UDP4_PROTOCOL** child and clean up the environment. The return status should be **EFI_SUCCESS**.

22.2 EFI_MTFTP4_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_MTFTP4_PROTOCOL Section.

22.2.1 CreateChild()

Number	GUID	Assertion	Test Description
5.26.2.1.1	0xf44c5295, 0x599e, 0x48bc, 0xbb, 0x67, 0xed, 0x9a, 0x21, 0x5b, 0xa9, 0xb1	**EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** - returns **EFI_INVALID_PARAMETER** when creating Child 1 again.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create the same child again. The return status must be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.1.5	0x5e30aa7c,0xd5f6, 0x4cac, 0xb2,0x54, 0xbf,0xdf, 0x16,0x3b, 0x34,0xfc	**EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** – invokes **CreateChild()** with *ChildHandle* being **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** with *ChildHandle* being **NULL.** The return status must be **EFI_INVALID_PARAMETER.**
5.26.2.1.2	0xca3fb64a, 0xd149, 0x4f76, 0x91, 0x45, 0xe4, 0xf6, 0xcc, 0xe6, 0x5b, 0x27	**EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** - returns **EFI_SUCCESS** when creating child1.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle: Handle1. The return status must be **FI_SUCCESS.**

Number	GUID	Assertion	Test Description
5.26.2.1.3	0xb07ae013, 0x0d83, 0x49c3, 0x99, 0x23, 0xef, 0x27, 0x67, 0xd5, 0x48, 0xfe	`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` - returns `EFI_SUCCESS` when creating child2.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle: Handle1. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters 3. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle: Handle2. The return status must be `EFI_SUCCESS.`
5.26.2.1.4	0xd4d966c4, 0xc05a, 0x4995, 0xbf, 0xfb, 0x2c, 0x86, 0x8b, 0x3c, 0x2c, 0x0b	`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` - returns `EFI_SUCCESS` when creating child3.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle: Handle1. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle: Handle2. 4. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters 5. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle: Handle3. The return status must be `EFI_SUCCESS.` 6. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters 7. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` three times to destroy the three newly created `EFI_MTFTP4_PROTOCOL` child handles and clean up the environment.

22.2.2 DestroyChild()

Number	GUID	Assertion	Test Description
5.26.2.2.1	0x3c312328, 0x313d, 0x47f6, 0x80, 0x7c, 0x5b, 0x1e, 0x10, 0xc2, 0xc0, 0x4d	**EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** - returns **EFI_INVALID_PARAMETER** when destroying a **NULL** child.	Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy a **NULL** child. The return status must be **EFI_INVALID_PARAMETER.**
5.26.2.2.2	0xe1c0ee52, 0xd5af, 0x4ec0, 0xa3, 0xf6, 0x31, 0xfb, 0xe0, 0xd4, 0xb7, 0x04	**EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** - returns **EFI_INVALID_PARAMETER** when destroying an un-existed child.	Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy an un-existed child. The return status must be **EFI_INVALID_PARAMETER.**
5.26.2.2.3	0x28f8e30c, 0xa5d9, 0x4327, 0x99, 0xfa, 0xac, 0xda, 0xc9, 0x5f, 0xa4, 0xff	**EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** - returns **EFI_UNSUPPORTED** when destroying the same child twice.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle: 2. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the new created child. 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created child again. The return status must be **EFI_UNSUPPORTED.**

Number	GUID	Assertion	Test Description
5.26.2.2.4	0xcb939b7a, 0x266a, 0x44f5, 0xa2, 0xe3, 0x57, 0xea, 0xde, 0x7f, 0x44, 0x08	`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` - returns `EFI_SUCCESS` with all valid invocations.	1 .Add an entry in ARP cache. 2. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle: Handle1. 3. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters 4. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 5. If having not captured the packet, OS side set assert fail and call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created child and clean up the environment. The return status must be `EFI_SUCCESS`.
5.26.2.2.5	0xc9d38d67, 0xadc1, 0x425d, 0xa4, 0xa1, 0x04, 0x18, 0xc6, 0x4b, 0x63, 0x0c	`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` - returns `EFI_SUCCESS` with all valid invocations.	1 .Add an entry in ARP cache. 2. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle: Handle1. 3. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters 4. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 5. If having captured the packet, configured OS side will send back a normal OACK packet with active flag set. 6. OS side captures ack packet sent from EUT side and call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the created child and clean up the environment. The return status must be `EFI_SUCCESS`.

22.2.3 GetModeData()

Number	GUID	Assertion	Test Description
5.26.2.3.1	0xdc9ac841, 0x8a0f, 0x4214, 0x91, 0x73, 0x60, 0x65, 0xee, 0x51, 0x8c, 0x52	**EFI_MTFTP4_PROTOCOL.GetModeData()** - returns **EFI_INVALID_PARAMETER** with a *ModeData* value of **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.GetModeData()** with a *ModeData* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.3.2	0x40eccfcd, 0xee1c, 0x405f, 0xb0, 0x64, 0x2d, 0xe5, 0x66, 0x7b, 0xfb, 0xee	**EFI_MTFTP4_PROTOCOL.GetModeData()** - returns **EFI_SUCCESS** with all valid invocations.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.GetModeData()** with all valid parameters. The return status must be **EFI_SUCCESS.** 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment..

22.2.4 Configure()

Number	GUID	Assertion	Test Description
5.26.2.4.1	0x2c5b72d9, 0x2c30, 0x4249, 0xa2, 0x3a, 0x92, 0x14, 0xfd, 0xea, 0x73, 0x12	**EFI_MTFTP4_PROTOCOL.Configure()** - returns **EFI_INVALID_PARAMETER** when *MtftpConfigData->UseDefaultSetting* is **FALSE** and *MtftpConfigData->StationIp* is an invalid IPv4 unicast address.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2.Call **EFI_MTFTP4_PROTOCOL.Configure()** with a *MtftpConfigData->UseDefaultSetting* value of **FALSE** and a *MtftpConfigData->StationIp* value of unicast address. The return status must be **EFI_INVALID_PARAMETER**. 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.4.2	0x01ef2cac, 0x1259, 0x41c9, 0xbd, 0x91, 0x49, 0x68, 0xa9, 0xfd, 0xd6, 0x42	**EFI_MTFTP4_PROTOCOL.Configure()** - returns **EFI_INVALID_PARAMETER** when *MtftpConfigData->UseDefaultSetting* is **FALSE** and *MtftpConfigData->SubnetMask* is invalid.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2.Call **EFI_MTFTP4_PROTOCOL.Configure()** when *MtftpConfigData->UseDefaultSetting* is **FALSE** and *MtftpConfigData->SubnetMask* is invalid. The return status must be **EFI_INVALID_PARAMETER.** 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.4.3	0xbe92bd2e, 0xd085, 0x4da2, 0xaf, 0xbf, 0xec, 0x7b, 0x0d, 0xc7, 0xec, 0xca	**EFI_MTFTP4_PROTOCOL.Configure()** - returns **EFI_INVALID_PARAMETER** when *MtftpConfigData->UseDefaultSetting* is **FALSE** and *MtftpConfigData-> ServerIp* is an invalid IPv4 unicast address.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with a *MtftpConfigData*-> *UseDefaultSetting* value of **FALSE** and a *MtftpConfigData*-> *ServerIp* value of an invalid IPv4 unicast address. The return status must be **EFI_INVALID_PARAMETER.** 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.4.4	0x5891d15c, 0x7f5d, 0x4c0d, 0xb0, 0x90, 0x88, 0xcd, 0x44, 0xe1, 0xea, 0x68	**EFI_MTFTP4_PROTOCOL.Configure()** - returns **EFI_INVALID_PARAMETER** when *MtftpConfigData->UseDefaultSetting* is **FALSE** and **MtftpConfigData-> GatewayIp** is an invalid IPv4 unicast address.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call *EFI_MTFTP4_PROTOCOL*.**Configure()** with a *MtftpConfigData*-> *UseDefaultSetting* value of **FALSE** and a *MtftpConfigData*-> *GatewayIp* value of an invalid IPv4 unicast address. The return status must be **EFI_INVALID_PARAMETER.** 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.4.5	0xd01d26be, 0x35fb, 0x4a08, 0xb0, 0x22, 0x7b, 0xe2, 0x53, 0xcf, 0x99, 0x02	**EFI_MTFTP4_PROTOCOL.Configure()** - returns **EFI_INVALID_PARAMETER** when *MtftpConfigData->UseDefaultSetting* is **FALSE** and *MtftpConfigData-> GatewayIp* is not in the same subnet with station address.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2.Call **EFI_MTFTP4_PROTOCOL.Configure()** when *MtftpConfigData*-> *UseDefaultSetting* is **FALSE** and *MtftpConfigData*-> *GatewayIp* is not in the same subnet with station address. The return status must be **EFI_INVALID_PARAMETER.** 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.4.6	0x37ccae28, 0x4b81, 0x4ba5, 0x8d, 0xe6, 0x79, 0xe7, 0xda, 0xb9, 0x03, 0x04	**EFI_MTFTP4_PROTOCOL.Configure ()** - returns **EFI_ACCESS_DENIED** when some operation of this EFI MTFTPv4 Protocol driver instance has not finished yet and the configuration data cannot be changed at this time.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2.Call **EFI_MTFTP4_PROTOCOL.Configure()** when some operation of this EFI MTFTPv4 Protocol driver instance has not finished yet and the configuration data cannot be changed at this time. The return status must be **EFI_ACCESS_DENIED.** 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.4.7	0xd31e47ea, 0x5a76, 0x49aa, 0xbd, 0x40, 0x6f, 0xd9, 0x49, 0x88, 0x5f, 0x84	**EFI_MTFTP4_PROTOCOL.Configure ()** - returns **EFI_SUCCESS** when it is reset by calling Configure () with a *MtftpConfigData* value of **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with *MtftpConfigData* set to **NULL**. The return status must be **EFI_SUCCESS.** 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

22.2.5 GetInfo()

Number	GUID	Assertion	Test Description
5.26.2.5.1	0x794b1aae, 0x92b4, 0x40de, 0xad, 0xed, 0x43, 0xb3, 0x55, 0x37, 0xd8, 0xa3	**EFI_MTFTP4_PROTOCOL.GetInfo()** - returns **EFI_INVALID_PARAMETER** with a *FileName* value of **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with a *MtftpConfigData->UseDefaultSetting* value of **FALSE**. 3. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** with a *FileName* value of **NULL**. The return stats must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.5.2	0x0733cdb5, 0x4072, 0x4129, 0xa2, 0x06, 0xce, 0x56, 0x6e, 0xf6, 0xd8, 0x61	**EFI_MTFTP4_PROTOCOL.GetInfo()** - returns **EFI_INVALID_PARAMETER** with an *OverrideData*.GatewayIp value of invalid.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with a *MtftpConfigData->UseDefaultSetting* value of **FALSE**. 3. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** with an *OverrideData*.GatewayIp value of invalid. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.5.3	0xa04d3e7c, 0x5e50, 0x4472, 0xa7, 0x70, 0xc1, 0xa9, 0x48, 0xcb, 0xd9, 0x1e	**EFI_MTFTP4_PROTOCOL.GetInfo()** - returns **EFI_INVALID_PARAMETER** with an invalid *OverrideData*.ServerIp value.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with a *MtftpConfigData->UseDefaultSetting* value of **FALSE**. 2. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** with an *OverrideData*.ServerIp value of invalid. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.5.4	0x10d2101c, 0x0aa3, 0x4713, 0xb8, 0x2b, 0xe1, 0x43, 0xed, 0xf4, 0x11, 0x26	**EFI_MTFTP4_PROTOCOL.GetInfo()** - returns **EFI_INVALID_PARAMETER** when *OverrideData*.GatewayIp and *OverrideData*.ServerIp are not in the same subnet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with a *MtftpConfigData*-> *UseDefaultSetting* value of **FALSE**. 3. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** with *OverrideData*.GatewayIp and *OverrideData*.ServerIp are not in the same subnet. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.5.5	0xf85b07f6, 0x9f89, 0x41ad, 0x8d, 0x53, 0x47, 0x53, 0x97, 0xac, 0x98, 0x1a	**EFI_MTFTP4_PROTOCOL.GetInfo()** - returns **EFI_INVALID_PARAMETER** when *OptionCount* is not 0 and *OptionList* is **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with a *MtftpConfigData*-> *UseDefaultSetting* value of **FALSE**. 3. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** when *OptionCount* is not 0 and *OptionList* is **NULL**. The return status must be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.5.6	0xb9caedcf, 0xf071, 0x421a, 0x9f, 0xb9, 0x7e, 0x24, 0x9d, 0xf4, 0xe3, 0xb2	**EFI_MTFTP4_PROTOCOL.GetInfo()** - returns **EFI_INVALID_PARAMETER** when *PacketLength* is **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with a *MtftpConfigData*-> *UseDefaultSetting* value of **FALSE**. 3. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** with a *PacketLength* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.5.7	0x5cb9e305, 0xb4e2, 0x4416, 0xa7, 0x35, 0xe2, 0x72, 0xb6, 0x98, 0xf8, 0x23	**EFI_MTFTP4_PROTOCOL.GetInfo()** - returns **EFI_TFTP_ERROR** with a MTFTPv4 ERROR packet having received in the Buffer.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with *MtftpConfigData*-> *UseDefaultSetting* is **FALSE**. 3. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** when *OverrideData* is **NULL** and *ModeStr* is **NULL**. OS side must capture the packet sent from EUT side. 4. If have captured the packet, configured OS side to send back a MTFTPv4 ERROR packet and OS side should capture another packet sent from EUT side. The return status must be **EFI_TFTP_ERROR**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.5.8	0x30e6a222, 0x2bbc, 0x4ff6, 0xa8, 0xf2, 0xd6, 0x8a, 0xc2, 0x91, 0x98, 0x29	**EFI_MTFTP4_PROTOCOL.GetInfo()** - returns **EFI_TIMEOUT** when no packets were received from the MTFTPv4 server.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** when *OverrideData* is **NULL** and ModeStr is **NULL**. In addition, the OS side doesn't send any packets back. The return status must be **EFI_TIMEOUT.** 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.5.9	0xc4c5ced1, 0x30a5, 0x4c54, 0xa3, 0xc0, 0x80, 0x2b, 0x35, 0x83, 0xbf, 0x70	**EFI_MTFTP4_PROTOCOL.GetInfo()** - returns **EFI_NOT_STARTED** with the EFI MTFTPv4 Protocol driver having not been started.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** when both *OverrideData* and *ModeStr* are **NULL**. The return status must be **EFI_NOT_STARTED**. 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.5.10	0x32db978c, 0x9d9b, 0x4144, 0x97, 0x9c, 0x27, 0x14, 0x42, 0x9f, 0xe3, 0x47	EFI_MTFTP4_PROTOCOL.GetInfo() - returns EFI_ACCESS_DENIED when invoking GetInfo() interface while the previous operation has not been completed yet.	1. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new EFI_MTFTP4_PROTOCOL child handle. 2. Call EFI_MTFTP4_PROTOCOL.Configure() with all valid parameters. 3. Call EFI_MTFTP4_PROTOCOL.ReadFile() to change the EFI_MTFTP4_PROTOCOL State. 4. Call EFI_MTFTP4_PROTOCOL.GetInfo() when the previous operation has not been completed yet. The return status must be EFI_ACCESS_DENIED. 5. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created EFI_MTFTP4_PROTOCOL child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.5.11	0xbf72714a, 0x113f, 0x487e, 0xab, 0x10, 0x08, 0xa7, 0x98, 0xf3, 0x4f, 0xc4	`EFI_MTFTP4_PROTOCOL.GetInfo()` - returns `EFI_SUCCESS` when the server responding a normal OACK packet.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.GetInfo()` with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure OS side to send back a normal OACK packet and OS side should capture another packet sent from EUT side. 5. The return status of the `EFI_MTFTP4_PROTOCOL.GetInfo()` must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.5.12	0x77dbe1e4, 0x6219, 0x4531, 0xae, 0xbe, 0x58, 0x26, 0x4b, 0x53, 0x7e, 0xd1	EFI_MTFTP4_PROTOCOL.GetInfo() - test the EFI_ICMP_ERROR conformance of GetInfo() when an ICMP ERROR packet was received and in the buffer.	1. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new EFI_MTFTP4_PROTOCOL child handle. 2. Call EFI_MTFTP4_PROTOCOL.Configure() with all valid parameters. 3. Call EFI_MTFTP4_PROTOCOL.GetInfo() with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure OS side to send back a ICMP error packet. The return status must be EFI_ICMP_ERROR. 5. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created EFI_MTFTP4_PROTOCOL child handle and clean up the environment.
5.26.2.5.13	0x70e8d3e9, 0x75a9, 0x4652, 0x82, 0x68, 0xa4, 0x0d, 0xdd, 0x1a, 0x81, 0x5f	EFI_MTFTP4_PROTOCOL.GetInfo() - test the EFI_UNSUPPORTED conformance of GetInfo() when one or more options in the optionlist are in the unsupported list of structure EFI_MTFTP4_MODE_DATA.	1. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new EFI_MTFTP4_PROTOCOL child handle. 2. Call EFI_MTFTP4_PROTOCOL.Configure() with all valid parameters. 3. Call EFI_MTFTP4_PROTOCOL.GetInfo() with one or more options in the optionlist are in the unsupported list of structure EFI_MTFTP4_MODE_DATA. The return status should be EFI_UNSUPPORTED. 4. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created EFI_MTFTP4_PROTOCOL child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.5.14	0xfaa23d30, 0x1d66, 0x4d8e, 0xbe, 0x21, 0x2d, 0xa7, 0xbc, 0x1c, 0x9d, 0xfd	**EFI_MTFTP4_PROTOCOL.GetInfo()** - test the **EFI_PROTOCOL_ERROR** conformance of **GetInfo()**. The client received an unexpected MTFTPv4 packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure OS side to send back an unexpected packet and the return status should be **EFI_PROTOCOL_ERROR**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment
5.26.2.5.15	0xd2c1e819, 0x610b, 0x4cfc, 0x94, 0xf1, 0x33, 0xcd, 0x13, 0xaf, 0x4b, 0xc9	**EFI_MTFTP4_PROTOCOL.GetInfo – GetInfo()** must return **EFI_NETWORK_UNREACHABLE** when receive an ICMP net unreachable packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROT OCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure Host side to send back an ICMP net unreachable packet and the return status should be **EFI_NETWORK_UNREACHABLE**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.5.16	0x290076e3, 0xdaf2, 0x453d, 0xb2,0x21,0xcd,0x27, 0xce,,0xe7,0x3d,0xbe	EFI_MTFTP4_PROTOCOL.GetInfo – GetInfo() must return EFI_HOST_UNREACHABLE when receiving an ICMP host unreachable packet.	1. Call EFI_MTFTP4_SERVICE_BINDING_PROT OCOL.CreateChild() to create a new EFI_MTFTP4_PROTOCOL child handle. 2. Call EFI_MTFTP4_PROTOCOL.Configure() with all valid parameters. 3. Call EFI_MTFTP4_PROTOCOL.GetInfo() with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure Host side to send back an ICMP host unreachable packet and the return status should be EFI_HOST_UNREACHABLE. 5. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created EFI_MTFTP4_PROTOCOL child handle and clean up the environment
5.26.2.5.17	0x706bc816, 0x6353, 0x40ae, 0xa9,0x47,0x9a,0xf0, 0x01,0xa9,0x82,0x8c	EFI_MTFTP4_PROTOCOL.GetInfo – GetInfo() must return EFI_PROTOCOL_UNREACHABLE when receive an ICMP protocol unreachable packet.	1. Call EFI_MTFTP4_SERVICE_BINDING_PROT OCOL.CreateChild() to create a new EFI_MTFTP4_PROTOCOL child handle. 2. Call EFI_MTFTP4_PROTOCOL.Configure() with all valid parameters. 3. Call EFI_MTFTP4_PROTOCOL.GetInfo() with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure Host side to send back an ICMP protocol unreachable packet and the return status should be EFI_PROTOCOL_UNREACHABLE. 5. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created EFI_MTFTP4_PROTOCOL child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.5.18	0xa165bd19, 0x951a, 0x4486, 0x88,0x4d,0x1d,0x94,0x30,0xa7,0xbe,0x3c	**EFI_MTFTP4_PROTOCOL.GetInfo – GetInfo()** must return **EFI_PORT_UNREACHABLE** when receive an ICMP port unreachable packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROT OCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.GetInfo()** with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure Host side to send back an ICMP port unreachable packet and the return status should be **EFI_PORT_UNREACHABLE**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

22.2.6 ParseOptions()

Number	GUID	Assertion	Test Description
5.26.2.6.1	0x9bea2f3f, 0x9f02, 0x4eb2, 0x8b, 0x1f, 0x99, 0xd5, 0xcf, 0xc3, 0x57, 0x29	**EFI_MTFTP4_PROTOCOL.ParseOptions()** - returns **EFI_INVALID_parameter** with a *PacketLength* value of 0.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ParseOptions()** with a **PacketLength** value of 0. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.6.2	0x0bc09196, 0xb38a, 0x4fa8, 0xb0, 0x38, 0x4c, 0x4c, 0x8b, 0x3c, 0x69, 0xfa	**EFI_MTFTP4_PROTOCOL.ParseOptions()** - returns **EFI_INVALID_parameter** with a *Packet* value of **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ParseOptions()** with a *Packet* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.6.3	0x72723929, 0x60bd, 0x49c1, 0x99, 0xbd, 0xd1, 0x48, 0x60, 0x33, 0x7a, 0xdc	**EFI_MTFTP4_PROTOCOL.ParseOptions()** - returns **EFI_INVALID_parameter** with a *Packet* value of an invalid MTFTPv4 Packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ParseOptions()** with a *Packet* value of an invalid MTFTPv4 Packet - **Packet.OpCode** is 0x11. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.6.4	0xb7ed01b9, 0x7e1b, 0x40ba, 0x8b, 0x6a, 0x52, 0x34, 0xdf, 0x13, 0x53, 0xf0	**EFI_MTFTP4_PROTOCOL.ParseOptions()** - returns **EFI_INVALID_parameter** with a *Packet* value of an invalid MTFTPv4 Packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ParseOptions()** with a *Packet* value of an invalid MTFTPv4 Packet - **Packet.OpCode** is 0x01. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.6.5	0x350c473e, 0x9901, 0x4125, 0xbc, 0xc9, 0x65, 0xbf, 0xa9, 0xf3, 0x16, 0x30	**EFI_MTFTP4_PROTOCOL.ParseOptions()** - returns **EFI_INVALID_parameter** with a *Packet* value of an invalid MTFTPv4 Packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ParseOptions()** with a *Packet* value of an invalid MTFTPv4 Packet - **Packet.OpCode** is 0x06 and *PacketLength* is 1. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.6.6	0xcf909489, 0xace2, 0x4fec, 0x8d, 0xc9, 0x66, 0xa0, 0xd9, 0x33, 0xa6, 0x4a	**EFI_MTFTP4_PROTOCOL.ParseOptions()** - returns **EFI_INVALID_parameter** with an *OptionCount* value of **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ParseOptions()** with an *OptionCount* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.6.7	0x0131da11, 0x62a1, 0x494f, 0xb1, 0x0a, 0xaf, 0x5d, 0xe2, 0x12, 0xe9, 0x88	EFI_MTFTP4_PROTOCOL.ParseOptions() - returns EFI_INVALID_PARAMETER when parsing a non-OACK packet.	1. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new EFI_MTFTP4_PROTOCOL child handle. 2. Call EFI_MTFTP4_PROTOCOL.Configure() with all valid parameters. 3. Call BS-> CopyMem() to fill the packet needed to be parsed. Set Packet.OpCode to be 0x100. 4. Call EFI_MTFTP4_PROTOCOL.ParseOptions() with the configured non-OACK packet. The return status must be EFI_INVALID_PARAMETER. 5. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created EFI_MTFTP4_PROTOCOL child handle and clean up the environment.
5.26.2.6.9	0x5b7bbe95, 0xdba3, 0x4e9c, 0x89, 0xde, 0x37, 0xf1, 0xf6, 0x42, 0x04, 0x24	EFI_MTFTP4_PROTOCOL.ParseOptions() - test the EFI_NOT_FOUND conformance of ParseOptions() with no options were found in the OACK packet.	1. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new EFI_MTFTP4_PROTOCOL child handle. 2. Call EFI_MTFTP4_PROTOCOL.Configure() with all valid parameters. 4. Call EFI_MTFTP4_PROTOCOL.ParseOptions() with no options were found in the OACK packet..The return status must be EFI_NOT_FOUND. 5. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created EFI_MTFTP4_PROTOCOL child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.6.8	0x973e370a, 0x5936, 0x4377, 0xb0, 0x6c, 0x82, 0xe6, 0x11, 0x4d, 0xda, 0x6f	`EFI_MTFTP4_PROTOCOL.ParseOptions()` - returns `EFI_SUCCESS` when parsing a OACK packet.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `BS-> CopyMem()` to fill the packet needed to be parsed. Set `Packet.OpCode` to be 0x600. 4. Call `EFI_MTFTP4_PROTOCOL.ParseOptions()` with the configured OACK packet. The return status must be `EFI_SUCCESS`. 5. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

22.2.7 ReadFile()

Number	GUID	Assertion	Test Description
5.26.2.7.1	0x38728e11, 0x6f6f, 0x409a, 0x84, 0x31, 0xf5, 0x1e, 0x60, 0x0f, 0x7d, 0x6f	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_TIMEOUT` with no packets sent back from the MTFTPv4 server.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. 4. If OS side has captured the packet, don’t send back any packets, stall and wait until client timeout. The return status must be `EFI_TIMEOUT`. 5. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.2	0xcb0105ab, 0x7f16, 0x46a1, 0x87, 0xf2, 0x18, 0x6b, 0x86, 0x74, 0x6a, 0xba	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_TIMEOUT** when the passive Client having not received any data packets from the server.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. Configure OS side to send back a normal OACK packet with flag set to be passive. 5. Then OS side doesn't send any data packets back, then stall and wait until client timeout. The return status must be **EFI_TIMEOUT**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.3	0x95384167, 0xa706, 0x4f2c, 0x82, 0x8c, 0x8e, 0x3f, 0x15, 0xee, 0x82, 0x0a	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_TFTP_ERROR` when TFTPv4 ERROR packet was received.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If having captured the packet, configure OS side to send back a `EFI_MTFTP4_PROTOCOL` Error packet. 5. OS side should capture another packet sent from EUT side. The return status must be `EFI_TFTP_ERROR.` 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.4	0xf5ac75d7, 0xa32e, 0x4b1f, 0xa8, 0x19, 0x2e, 0xfc, 0x73, 0x24, 0xcc, 0xba	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_TFTP_ERROR** when the active client receives an MTFTPv4 ERROR packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If having captured the packet, Configure OS side to respond a normal OACK with flag set to be active. 5. If having captured ack, OS side sends back a **EFI_MTFTP4_PROTOCOL** Error packet. The return status must be **EFI_TFTP_ERROR**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.5	0x49f424ed, 0xfdbc, 0x4c82, 0x8d, 0xb8, 0xd5, 0xa2, 0xa4, 0x9b, 0x7e, 0xff	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_TFTP_ERROR` when the passive client has received a MTFTPv4 ERROR packet.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If having captured the packet, configure OS side to respond a normal OACK packet with flag set to be passive, and then send back a `EFI_MTFTP4_PROTOCOL` Error packet. The return status must be `EFI_TFTP_ERROR`. 5. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.6	0x1392cef9, 0x74e0, 0x4f89, 0xa5, 0x26, 0xa7, 0xa7, 0x77, 0x56, 0x33, 0xd4	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_TFTP_ERROR** when the server responds with an error OACK packet – active/passive flag error.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to respond with an error OACK packet – active/ passive flag error. 5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be **EFI_TFTP_ERROR**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.7	0x1f4fd053, 0x9e4b, 0x49c4, 0x9a, 0xea, 0x58, 0x75, 0x60, 0xf1, 0xec, 0x7d	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_TFTP_ERROR** when timeout value in OACK packet is invalid.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to respond with an error OACK packet – timeout value is invalid. 5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be **EFI_TFTP_ERROR**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.8	0x9bbcc0bb, 0x5386, 0x4e5c, 0xa3, 0xac, 0x65, 0xc7, 0x62, 0xf6, 0x93, 0xaa	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_TFTP_ERROR** when blocksize option value in OACK packet is invalid.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to respond with an error OACK packet – blocksize option value is invalid. 5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be **EFI_TFTP_ERROR**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.9	0x329ae187, 0x6758, 0x42b9, 0x84, 0xae, 0x92, 0x32, 0x42, 0x15, 0xa5, 0xef	**`EFI_MTFTP4_PROTOCOL.ReadFile()`** - returns **`EFI_TFTP_ERROR`** when multicast IP address in OACK packet is invalid.	1. Call **`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new **`EFI_MTFTP4_PROTOCOL`** child handle. 2. Call **`EFI_MTFTP4_PROTOCOL.Configure()`** with all valid parameters. 3. Call **`EFI_MTFTP4_PROTOCOL.ReadFile()`** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to respond with an error OACK packet – multicast IP address is invalid. 5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be **`EFI_TFTP_ERROR`**. 6. Call **`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created **`EFI_MTFTP4_PROTOCOL`** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.10	0xe491fc10, 0x0c0f, 0x4d45, 0xb5, 0xc3, 0x3c, 0x29, 0x10, 0xdb, 0xe4, 0x70	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_TFTP_ERROR** when client's listening port in OACK packet is invalid.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to respond with an error OACK packet – client's listening port is 65536. 5. If having captured an ack, OS check whether it is a packet with an error code. The return status must be **EFI_TFTP_ERROR**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.11	0xcff83e43, 0x5d33, 0x4cc0, 0x80, 0xc4, 0x55, 0x96, 0x0e, 0x5f, 0x58, 0xae	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_TFTP_ERROR` when the format of multicast IP address in OACK packet is invalid.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to respond with an error OACK packet – the format of multicast IP address is invalid. 5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be `EFI_TFTP_ERROR`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.12	0x28754983, 0xac7d, 0x4e7f, 0x9f, 0xad, 0xbf, 0x55, 0x59, 0xff, 0xa7, 0x62	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_TFTP_ERROR` when the format of multicast option in OACK packet is invalid.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to respond with an error OACK packet – the format of multicast option is invalid. 5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be `EFI_TFTP_ERROR`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.13	0x64fd965d, 0x2acc, 0x4540, 0xbc, 0x57, 0x50, 0xe8, 0xab, 0x02, 0xe8, 0x8a	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_TFTP_ERROR` when the format of multicast option in OACK packet is invalid.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to respond with error OACK packet – the format of multicast option is invalid. 5. If having captured ack, OS check whether it is a packet with error code. The return status must be `EFI_TFTP_ERROR`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.14	0xd09c7076, 0x316f, 0x4245, 0xac, 0x31, 0x95, 0x82, 0x22, 0xa4, 0x67, 0xd7	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_TFTP_ERROR` when the format of multicast option in OACK packet is invalid.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to respond with an error OACK packet – the format of multicast option is invalid. 5. If having captured an ack, OS check whether it is a packet with an error code. The return status must be `EFI_TFTP_ERROR`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.15	0x1322cb38, 0x8f90, 0x4fa8, 0xbe, 0xa9, 0x5b, 0x31, 0x8e, 0xb8, 0x24, 0xad	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_TFTP_ERROR` when the passive client tries to change to be active, but the server responds with an error OACK packet - active/ passive flag is error.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to respond OACK with flag set to be passive and then send the file missing several packets. 5. The OS side should capture the ack sent from the passive client to ask for the missing packets. 7. If having captured it, OS sends back OACK with error active/ passive flag. The return status must be `EFI_TFTP_ERROR`. 8. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.16	0xff2d0e80, 0xdecd, 0x4a1c, 0xb6, 0x7c, 0xe4, 0xcd, 0x99, 0x9d, 0x69, 0x09	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_TFTP_ERROR` when the server adds more other options in the OACK packet.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active while adding more other options in the OACK packet; then OS should capture the ack packet. 5. If having captured ack, OS sends back the only data packet and then receives another ack. The return status must be `EFI_TFTP_ERROR`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.17	0xa7fcbfff, 0x8367, 0x466e, 0x9d, 0x25, 0x5b, 0x80, 0xb8, 0x4f, 0xb5, 0x8f	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_TFTP_ERROR** when active client receives OACK, while *Token->OptionCount* is 0.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** without any option requested. OS side must capture the packet sent from EUT side 4. If OS side has captured the packet, configure server to send back OACK with some options and flag set to be active. 5. Then OS should capture ack and sends back the only one data packet. The return status must be **EFI_TFTP_ERROR**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.18	0x00450815, 0x41f5, 0x4da8, 0x90, 0x66, 0x78, 0x80, 0x94, 0x07, 0x34, 0xea	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_TFTP_ERROR` – When the passive client downloads, it misses the first and the last data packet. Then server set the client to be active while changing the transfer channel.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive. 5. Then server sends back the second and the third data packet to the multicast IP address while missing the first and the last data packet. 6. After passive client is timeout, it'll send ack0 to ask for the missing packets and the server should capture it. 7. If having captured the request, the server sends back OACK with flag set to be active and the client's listening port also changed. Then the server should capture an error packet. The return status must be `EFI_TFTP_ERROR`. 8. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.19	0x9017244c, 0x127a, 0x486e, 0x81, 0x5b, 0x20, 0xe8, 0xa6, 0x55, 0xd4, 0x6f	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_NOT_STARTED** with the EFI MTFTPv4 Protocol driver having not been started.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. The return status must be **EFI_NOT_STARTED.** 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.7.20	0x84b13fab, 0x04f5, 0x474b, 0x89, 0x4c, 0x63, 0xef, 0x9d, 0xcf, 0x78, 0x58	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_INVALID_PARAMETER** when *Token* is **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** when *Token* is **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.21	0xd25ff5a4, 0x71e7, 0x4e38, 0xb4, 0x3e, 0x4a, 0xcc, 0xe7, 0x83, 0xfa, 0x77	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_INVALID_PARAMETER** when *Token*->*Filename* is **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** when *Token*->*Filename* is **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment
5.26.2.7.22	0xf370c329, 0xe20b, 0x45a0, 0x9a, 0xb3, 0xd4, 0x13, 0x70, 0x98, 0x00, 0x03	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_INVALID_PARAMETER** when *Token*->*OptionCount* is not 0 and *Token*->*OptionList* is **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** when *Token*->*OptionCount* is not 0 and *Token*->*OptionList* is **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.23	0x2357c86f, 0xf9ba, 0x4f25, 0x9c, 0x77, 0x75, 0x10, 0xab, 0xb5, 0x10, 0x7e	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_INVALID_PARAMETER** when *Token*->*Buffer* and *Token*->*CheckPacket* are both **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** when *Token*->*Buffer* and *Token*->*CheckPacket* are both **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment
5.26.2.7.24	0x66019567, 0x321d, 0x41a8, 0xaa, 0xff, 0x60, 0x7f, 0x75, 0xa4, 0x08, 0x42	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_INVALID_PARAMETER** when *OverrideData*.GatewayIp is invalid.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** when *OverrideData*.GatewayIp is invalid. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.25	0x5f64495c, 0xad06, 0x4185, 0x87, 0x55, 0x86, 0xd9, 0x44, 0xf6, 0x39, 0x81	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_INVALID_PARAMETER** when *OverrideData*.ServerIp is invalid.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** when *OverrideData*.ServerIp is invalid. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment
5.26.2.7.26	0x17fa0734, 0x38f6, 0x4fe5, 0x9f, 0x6a, 0x5d, 0xae, 0x9e, 0xf2, 0xf3, 0xac	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_INVALID_PARAMETER** when *OverrideData*.GatewayIp is not in the same subnet with *StationIp*.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** when *OverrideData*.*GatewayIp* is not in the same subnet with *StationIp*. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.27	0xa5d93fc4, 0x9b20, 0x45cc, 0xbe, 0x45, 0xcc, 0x60, 0x5e, 0x51, 0xae, 0xf4	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_UNSUPPORTED** when options of “restart” and “session” in the *Token*->*OptionList* are in the unsupported list of this implementation.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** when options of “restart” and “session” in the *Token*->*OptionList* are in the unsupported list of this implementation. The return status must be **EFI_UNSUPPORTED**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment
5.26.2.7.28	0x40f05e07, 0x3a7b, 0x4244, 0x97, 0x4f, 0x96, 0x9a, 0x89, 0x5c, 0xa4, 0x83	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_UNSUPPORTED** when option of “pktdelay” in the *Token*->*OptionList* are in the unsupported list of this implementation.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** when option of “pktdelay” in the *Token*->*OptionList* are in the unsupported list of this implementation.The return status must be **EFI_UNSUPPORTED**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.29	0xa8d5abdf, 0x3e19, 0x462e, 0x9f, 0x6d, 0x9f, 0xa6, 0x13, 0xd2, 0x96, 0xd3	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_ACCESS_DENIED** for calling **EFI_MTFTP4_PROTOCOL.ReadFile()** again before the first call ends.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** for the first time with all valid parameters. 4. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** for the second time with the same *Token* before the first call ends. The return status must be **EFI_ACCESS_DENIED**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.30	0xab02a8d2, 0x2086, 0x4372, 0xb5, 0xc7, 0x06, 0x0e, 0x28, 0x65, 0x1e, 0x8f	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_BUFFER_TOO_SMALL` when client is active and the *BufferSize* is not larger enough to hold the downloaded data in downloading process.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. Configure OS side to send back a normal OACK packet with multicast option and flag set to be active. In addition, OS side should capture Ack packet sent from EUT side and then responds with data packet whose size is larger than the set *BufferSize*. 5. Then OS side should capture another packet. The return status must be `EFI_BUFFER_TOO_SMALL`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.31	0xf135f02b, 0x51ca, 0x47b9, 0xab, 0xf4, 0x4b, 0xd9, 0x78, 0x86, 0x68, 0xf8	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_BUFFER_TOO_SMALL** when client is passive.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. Configure OS side to send back a normal OACK packet with multicasts option and flag set to be passive and wait for the client's processing. 5. OS side sends a data packet whose size is larger than the set *BufferSize*. The return status must be **EFI_BUFFER_TOO_SMALL**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.32	0xb8363dd2, 0xedca, 0x49a6, 0xbe, 0x32, 0x90, 0x87, 0xb9, 0x57, 0x6a, 0x1f	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_BUFFER_TOO_SMALL** when calling *ReadFile* asynchronously and Client is passive.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** asynchronously with all valid parameters. OS side must capture the packet sent from EUT side. 4. Configure OS side to send back a normal OACK packet with option of multicast and flag set to be passive and wait for the client's processing. 5. OS side sends a data packet whose size is larger than the set *BufferSize*. The return status must be **EFI_BUFFER_TOO_SMALL**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment

Number	GUID	Assertion	Test Description
5.26.2.7.33	0x5ae24123, 0xbb88, 0x42a5, 0xa1, 0xd0, 0xb3, 0x49, 0xfa, 0x20, 0x04, 0x6f	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_BUFFER_TOO_SMALL** when the client is an active client and the *BufferSize* is not larger enough to hold the downloaded data in downloading process - return this status until having received the last data block.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. Configure OS side to send back a normal OACK packet with multicast option and flag set to be active. In addition, OS side should capture Ack packet sent from EUT side and then responds with serious data packets whose size are larger than the set *BufferSize*. 5. Then OS side should capture the ack for the data blocks except the last block. 6. Then OS side should capture the error packet. The return status must be **EFI_BUFFER_TOO_SMALL**. 7. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.34	0xcfdaf47b, 0x8a46, 0x498c, 0x92, 0x0e, 0x96, 0x15, 0xc1, 0x23, 0xbe, 0x57	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_ABORTED** when the user aborts the active download process in *CheckPacket* callback routine in the case of receiving data packets.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with *CheckPacket* callback set. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture an ack packet sent from client. 5. If having captured it, server sends the only data packet back to the client. 6. Then server should capture another packet and check that if it is an error packet. The return status must be **EFI_ABORTED**. 7. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.35	0x731fb0ec, 0xb6b1, 0x4424, 0xb0, 0x61, 0x1b, 0xaa, 0xb3, 0x3f, 0xc0, 0x88	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_ABORTED** when the user aborts the active download process in *CheckPacket* callback routine in the case of receiving OACK packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with *CheckPacket* callback set. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture another packet and check whether it is an error packet. The return status must be **EFI_ABORTED**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.36	0xbd75e9f5, 0x76b3, 0x4e67, 0xb9, 0xbf, 0xcd, 0xfb, 0xed, 0x5c, 0x34, 0xa6	**`EFI_MTFTP4_PROTOCOL.ReadFile()`** - returns **`EFI_ABORTED`** when the user aborts the passive download process in *`CheckPacket`* callback routine in the case of receiving data packets.	1. Call **`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new **`EFI_MTFTP4_PROTOCOL`** child handle. 2. Call **`EFI_MTFTP4_PROTOCOL.Configure()`** with all valid parameters. 3. Call **`EFI_MTFTP4_PROTOCOL.ReadFile()`** with *`CheckPacket`* callback set. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive. 5. If having captured it, server sends the only data packet back to the client. 6. Then server should capture another packet and check that if it is an error packet. The return status must be **`EFI_ABORTED`**. 7. Call **`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created **`EFI_MTFTP4_PROTOCOL`** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.37	0xc9f2cdc8, 0x38eb, 0x4446, 0x9d, 0xc4, 0x5c, 0x78, 0x4a, 0x69, 0x0b, 0xd1	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_ABORTED** when the user aborts the passive download process in *CheckPacket* callback routine in the case of receiving OACK packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with *CheckPacket* callback set. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive and then OS should capture another packet and check that if it is an error packet. The return status must be **EFI_ABORTED**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.38	0xc911f1f0, 0x385b, 0x4de3, 0xb3, 0x86, 0xe3, 0x20, 0xec, 0x3c, 0xa8, 0xc2	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_ABORTED` when the user aborts the active download process in *`TimeoutCallback`* routine in the case of receiving Ack.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with Timeout callback set. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack. The return status must be `EFI_ABORTED`. 5. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.
5.26.2.7.39	0xfd55be46, 0xb941, 0x4708, 0xbe, 0x69, 0x24, 0x82, 0xca, 0x2c, 0x29, 0x34	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_ABORTED` when the user aborts the passive download process in Timeout Callback routine in the case of receiving Ack.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with Timeout callback set. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive and then OS should capture ack. The return status must be `EFI_ABORTED`. 5. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.40	0x8ad083d8, 0x9757, 0x40ef, 0x99, 0x86, 0x21, 0xee, 0x90, 0x4a, 0xa0, 0x2d	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the server sends back with normal OACK packet whose active flag is set.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet. 5. If having captured ack, OS sends back the only data packet and then receives another ack. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.41	0xc3640c29, 0xbfcd, 0x4f0c, 0xae, 0x7e, 0xcc, 0x44, 0x8a, 0xc1, 0x8e, 0x16	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when server send backs with normal OACK packet whose passive flag is set.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive and stall to wait for the client to join in the multicast group. 5. OS sends back the only data packet. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.42	0x5e294d5a, 0xf09e, 0x4fdc, 0xa2, 0x2e, 0x9d, 0xcb, 0xfa, 0x44, 0x3d, 0x2b	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the server sends back normal OACK packet after the client resends RRQ several times.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters, client should retry 5 times to send RRQ then OS side should capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet. 5. If having captured ack, OS sends back the only data packet and then receives another ack. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.43	0x162e4457, 0x63d9, 0x4402, 0xad, 0xac, 0xaa, 0xdf, 0x3a, 0x61, 0xaf, 0xdc	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the server doesn't copy the client's option strings verbatim from the RRQ packet to the OACK packet.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active while not coping the client's option strings verbatim from the RRQ packet to the OACK packet; then OS should capture ack packet. 5. If having captured ack, OS sends back the only data packet and then receives another ack. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.44	0xe0d3922c, 0x017d, 0x44a2, 0x90, 0x88, 0xad, 0xb6, 0xeb, 0x9f, 0x4c, 0xed	**`EFI_MTFTP4_PROTOCOL.ReadFile()`** - returns **`EFI_SUCCESS`** when client receives an error server source port data packet, it just ignores the packet and continues the data processing.	1. Call **`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new **`EFI_MTFTP4_PROTOCOL`** child handle. 2. Call **`EFI_MTFTP4_PROTOCOL.Configure()`** with all valid parameters. 3. Call **`EFI_MTFTP4_PROTOCOL.ReadFile()`** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active; then OS should capture ack packet. 5. If having captured ack, OS sends back an error server source data packet. In addition, client just ignores it. 6. The server sends back another correct source data packet then. The return status must be **`EFI_SUCCESS`**. 7. Call **`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created **`EFI_MTFTP4_PROTOCOL`** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.45	0xcc4f141c, 0x9df1, 0x404e, 0x90, 0x27, 0x60, 0xea, 0xbd, 0xa8, 0x08, 0xd8	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` After passive client having received some packets, the server sets it to be active and sends out remaining packet.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back a normal OACK to set the client passive. 5. If having captured ack0, OS sends back the first and the last data packet. 6. Server resends an empty multicast OACK to set the client active. 7. If having captured ack, OS sends out remain packet. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.46	0x2d4d9962, 0x24ac, 0x4f62, 0x9b, 0x66, 0x3c, 0xa5, 0xf3, 0x67, 0xb3, 0xa0	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_SUCCESS** when server doesn't support option extension and just sends back the data packet.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If having captured the packet, OS side sends back the only one data packet then receives another ack. The return status must be **EFI_SUCCESS**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.47	0x9e9e85f5, 0x669d, 0x4de3, 0x82, 0xa4, 0xff, 0x96, 0xb9, 0x69, 0x79, 0x05	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the server doesn't support multicast option and just doesn't support multicast.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back an OACK without multicast option only and then OS should capture ack packet. 5. If having captured ack, OS sends back the only one data packet then receives another ack. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.48	0x0bdc47fc, 0x659e, 0x497f, 0x8d, 0x10, 0x10, 0x52, 0xd3, 0x95, 0x7d, 0x19	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the client continuously joins the group to download file, while the Active flag is set.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Do the step of 4,5,6 for 5 times: 4. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 5. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet. 6. If having captured ack, OS sends back the only data packet and then receives another ack. The return status must be `EFI_SUCCESS`. 7. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.49	0xc965cbdf, 0x1539, 0x4507, 0xb0, 0xd1, 0x4f, 0xcd, 0x17, 0xc4, 0xbb, 0x54	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the server sends back the Data with incorrect sequence of the block numbers.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet. 5. If having captured ack, OS sends back data packets with incorrect sequence of block numbers and then receives another ack. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.50	0xcf00a8ae, 0x8676, 0x4ee3, 0xb5, 0xcc, 0x82, 0x22, 0xf9, 0x46, 0x94, 0x03	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_SUCCESS** when the server sends back the Data after some packets' retransmission.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet. 5. If having captured ack, server send backs the Number1 data packet and then receive another ack. 6. Then server doesn't do anything until having received the fourth ack. Then it sends the rest data packets back. The return status must be **EFI_SUCCESS**. 7. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.51	0x21a80b34, 0x73b3, 0x47ba, 0x82, 0x0c, 0x37, 0x34, 0x43, 0x7e, 0xd5, 0xd4	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_SUCCESS** when client downloads a file with length equal to 1 byte.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet. 5. If having captured ack, OS sends back the only data packet with length equal to 1 byte and then receives another ack. The return status must be **EFI_SUCCESS**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.52	0x9e8004a9, 0xc28c, 0x461b, 0x84, 0xa1, 0x31, 0xca, 0xc6, 0x48, 0x31, 0x28	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when using *`OverrideData`* to replace the configuration data and retry counter is set to 0 in override data.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with *`OverrideData`* replacing the configuration data and retry counter set to 0 in override data. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet. 5. If having captured ack, OS sends back the only data packet with length equal to 1 byte and then receives another ack. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.53	0x9bd82567, 0x6249, 0x4635, 0xb0, 0x2d, 0xf8, 0x06, 0x0d, 0x26, 0x68, 0xa6	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when active client receives data packets after server sends back OACK packet twice.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active; then OS should capture ack packet. 5. If having captured ack, OS send backs another OACK with the same option. Then sends back the only data packet and then receives another ack. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.54	0xcf35445d, 0x0aa1, 0x4485, 0x8e, 0xb6, 0x5f, 0xd8, 0xb4, 0x65, 0x55, 0x84	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the client is passive and it receives unexpected packets (BlockNo is a former number).	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive. 5. Then OS doesn't sends back all the data packets in sequence. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.55	0x62908d19, 0xc308, 0x4f16, 0xa1, 0x70, 0xb6, 0x9a, 0xdf, 0x47, 0xb4, 0x72	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_SUCCESS** when the client is passive and it receives unexpected packets (BlockNo is a further number).	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive. 5. Then OS doesn't sends back all the data packets in sequence. The return status must be **EFI_SUCCESS**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.56	0x58c614fb, 0x51d9, 0x4043, 0xb1, 0x24, 0x95, 0xa3, 0x7c, 0xcd, 0x3d, 0x70	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_SUCCESS** when the server responds data packet with data length larger than blocksize.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back OACK with flag set to be active. In addition, OS should capture an ack. 5. If having captured it, OS sends back the first data packet with length larger than blocksize, then the rest data packet. 6. OS should capture ack. The return status must be **EFI_SUCCESS**. 7. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.57	0x23a7aebe, 0x0117, 0x44fc, 0x9d, 0xcc, 0x68, 0x4c, 0xa6, 0x31, 0x2a, 0x20	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the client receives an unexpected ACK packet in the case of downloading file.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back an unexpected ACK and a normal OACK with flag set to be active. 5. Then if OS side has captured the ack, OS side sends back the only data packet. 6. OS should capture another ack. The return status must be `EFI_SUCCESS`. 7. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.58	0x9df88b27, 0x0a20, 0x4d91, 0x98, 0x2b, 0x32, 0x26, 0x41, 0x62, 0x39, 0x44	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_SUCCESS** when the active client receives an unexpected OACK packet in the case of downloading file.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be active. 5. If OS side has captured the ack, OS side send backs OACK again. 6. The server should capture another ack. Then the server sends the only data packet back to the client. The return status must be **EFI_SUCCESS**. 7. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.59	0xad60cb28, 0x6451, 0x400a, 0xa5, 0x74, 0xf6, 0x35, 0x9f, 0x01, 0x92, 0xd3	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the passive client receives an unexpected OACK packet in the case of downloading file.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive. 5. Then server sends back the first data packet to the multicast IP address and another OACK to the client again. 6. Then the server sends the last data packet back to the multicast IP address. The return status must be `EFI_SUCCESS`. 7. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.60	0x2309b8ea, 0x5593, 0x4835, 0xb6, 0x24, 0x65, 0xda, 0xc5, 0x51, 0x04, 0x5d	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the passive client downloads, it misses the last data packet. After client is timeout, server sets client to be passive again and sends out the lost packet.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive. 5. Then server sends back the first three data packets to the multicast IP address while missing the last data packet. 6. After passive client is timeout, it'll send ack0 to ask for missing packets and the server should capture it. 7. If having captured the request, the server sends OACK back again and then the last data packet. The return status must be `EFI_SUCCESS`. 8. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.61	0xf6c81b41, 0x8edd, 0x46df, 0x8a, 0x82, 0x46, 0x40, 0xd9, 0x8b, 0xda, 0xa5	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the passive client downloads, it misses the last data packet. After client is timeout, server sets client to be passive again and sends out all the data packets.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive. 5. Then server sends back the first three data packets to the multicast IP address while missing the last data packet. 6. After passive client is timeout, it'll send ack0 and the server should capture it. 7. If having captured the request, the server sends OACK back again and then all the data packets. The return status must be `EFI_SUCCESS`. 8. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.62	0x7156d37f, 0xd7ef, 0x47ea, 0xa2, 0xf3, 0x64, 0x3e, 0x7c, 0x44, 0x9f, 0x65	**EFI_MTFTP4_PROTOCOL.ReadFile()** - returns **EFI_SUCCESS** when the passive client downloads, it misses the first and the last packet. After client is timeout, server sets client to be passive again and sends out the lost packet	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive. 5. Then server sends back the second and the third data packet to the multicast IP address while missing the first and the last data packet. 6. After passive client is timeout, it'll send ack0 and the server should capture it. 7. If having captured the request, the server sends OACK back again and then all the lost packets. The return status must be **EFI_SUCCESS**. 8. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.63	0x34753378, 0xb423, 0x40b1, 0x93, 0x7c, 0x4d, 0xaa, 0x5c, 0xa6, 0x63, 0x43	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the passive client downloads, it misses the first and the last packet. After client is timeout, server sets client to be passive again and sends out all the data packets.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive. 5. Then server sends back the second and the third data packet to the multicast IP address while missing the first and the last data packet. 6. After passive client is timeout, it'll send ack0 and the server should capture it. 7. If having captured the request, the server sends OACK back again and then all the packets. The return status must be `EFI_SUCCESS`. 8. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.64	0xd756be67, 0xd667, 0x432f, 0xbb, 0xd6, 0x3a, 0xe1, 0xf5, 0xe6, 0x61, 0xd1	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the passive client downloads, it misses the first and the last packet. After client is timeout, server sets client to be active again and sends out the missing packets.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive. 5. Then server sends back the second and the third data packet to the multicast IP address while missing the first and the last data packet. 6. After passive client is timeout, it'll send ack0 to ask for the missing packets. 7. If having captured the request, the server sends back OACK with flag set to be active and then the first data packet. 8. The server expects the ack packet to request the last packet. If having captured it, server will send the last data packet. The return status must be `EFI_SUCCESS`. 9. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.65	0xc0fc889f, 0xc91f, 0x4a41, 0x80, 0x59, 0x0e, 0x22, 0x56, 0x79, 0x0b, 0x53	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the passive client downloads, it misses two blocks of packets. After client is timeout, server sets client to be passive again and sends out the lost packets randomly.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive. 5. Then server sends back the first and the seventh data packet to the multicast IP address while missing the Number2, 3, 4, 5, 6, 8 data packets. 6. After passive client is timeout, it'll send ack0 and the server should capture it. 7. If having captured the ack0 packet, the server sends back OACK with flag set to be passive. Then it sends out the data packets randomly in the order Number4, 2, 6, 3, 5, 8. 8. The server expects the ack packet. If having captured, server will send the second data packet. The return status must be `EFI_SUCCESS`. 9. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.66	0x5a4ed7d1, 0x0e36, 0x4f9c, 0xa7, 0x9c, 0xf2, 0x35, 0x2e, 0xf7, 0x3b, 0x2d	`EFI_MTFTP4_PROTOCOL.ReadFile()` - returns `EFI_SUCCESS` when the passive client downloads, it misses the first and the last packets. Then server changes the client to be active and retrieves its unicast transfer model.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.ReadFile()` with all valid parameters. OS side must capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive. 5. Then server sends back the second and the third data packets to the multicast IP address while missing the first and the last data packets. 6. After passive client is timeout, it'll send ack0 and the server should capture it. 7. If having captured the request, the server sends back OACK with transfer mode changed to be unicast and flag set to be active. Then it expects the ack sent from the client and sends out the first data packet. 8. The server should capture the ack and then sends back the second packet. 9. As above, server sends the third and the last data packets. The return status must be `EFI_SUCCESS`. 10. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.67	0xb441ee5b, 0xbf7f, 0x446f, 0xa2, 0x5c, 0x77, 0x7a, 0x0b, 0xdd, 0xde, 0x78	**EFI_MTFTP4_PROTOCOL.ReadFile()** - to test the **EFI_ICMP_ERROR** conformance of **ReadFile()** with an ICMP ERROR packet being received.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** and OS side capture the packet sent from EUT side. 4. If OS side has captured the packet, configure server to send back an ICMP error packet. The return status must be **EFI_ICMP_ERROR**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.7.68	0x6eaabf78, 0x3914, 0x4d08, 0x85, 0x0c, 0xbf, 0x63, 0x6d, 0xe9, 0xf3, 0x55	**EFI_MTFTP4_PROTOCOL.ReadFile()** - to test the **EFI_INVALID_PARAMETER** conformance of **ReadFile()** when one or more options in *Token.OptionList* have wrong format.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadFile()** with invalid muticast option value. The return status must be **EFI_INVALID_PARAMETER.** 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.69	0xd5e062fc, 0x5c0f, 0x470c, 0x8b,0x7a,0x44,0xf7, 0xbc,0xad, 0xc6,0x9c	EFI_MTFTP4_PROTOCOL.ReadFile()- ReadFile() must return EFI_NETWORK_UNREACHABLE when receive an ICMP network unreachable packet.	1. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new EFI_MTFTP4_PROTOCOL child handle. 2. Call EFI_MTFTP4_PROTOCOL.Configure() with all valid parameters. 3. Call EFI_MTFTP4_PROTOCOL.ReadFile() with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure Host side to send back an ICMP network unreachable packet and the return status should be EFI_NETWORK_UNREACHABLE. 5. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created EFI_MTFTP4_PROTOCOL child handle and clean up the environment.
5.26.2.7.70	0x6d8a5555, 0xe632, 0x470e, 0x98,0xe5,0x61,0xd2,0x2e,0xc9, 0x0d,0x0d	EFI_MTFTP4_PROTOCOL.ReadFile() - ReadFile() must return EFI_HOST_UNREACHABLE when receive an ICMP host unreachable packet.	1. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new EFI_MTFTP4_PROTOCOL child handle. 2. Call EFI_MTFTP4_PROTOCOL.Configure() with all valid parameters. 3. Call EFI_MTFTP4_PROTOCOL.ReadFile() with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure Host side to send back an ICMP host unreachable packet and the return status should be EFI_HOST_UNREACHABLE. 5. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created EFI_MTFTP4_PROTOCOL child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.7.71	0x732738e8, 0x1ff1, 0x4f3a, 0xa0,0xc8, 0x38,0x81,0x1d,0x15,0x92,0x83	EFI_MTFTP4_PROTOCOL.ReadFile() - ReadFile() must return EFI_PROTOCOL_UNREACHABLE when receive an ICMP protocol unreachable packet.	1. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new EFI_MTFTP4_PROTOCOL child handle. 2. Call EFI_MTFTP4_PROTOCOL.Configure() with all valid parameters. 3. Call EFI_MTFTP4_PROTOCOL.ReadFile() with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure Host side to send back an ICMP protocol unreachable packet and the return status should be EFI_PROTOCOL_UNREACHABLE. 5. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created EFI_MTFTP4_PROTOCOL child handle and clean up the environment.
5.26.2.7.72	0xd1c4e1e8, 0x1099, 0x4646, 0xb7,0xc9, 0x64,0x7e, 0x65,0xc3, 0x82,0x30	EFI_MTFTP4_PROTOCOL.ReadFile()- ReadFile() must return EFI_PORT_UNREACHABLE when receive an ICMP port unreachable packet.	1. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild() to create a new EFI_MTFTP4_PROTOCOL child handle. 2. Call EFI_MTFTP4_PROTOCOL.Configure() with all valid parameters. 3. Call EFI_MTFTP4_PROTOCOL.ReadFile() with all valid parameters. OS side should capture the packet sent from EUT side. 4. Configure Host side to send back an ICMP port unreachable packet and the return status should be EFI_PORT_UNREACHABLE. 5. Call EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild() to destroy the newly created EFI_MTFTP4_PROTOCOL child handle and clean up the environment.

22.2.8 WriteFile()

Number	GUID	Assertion	Test Description
5.26.2.8.1	0x4b00df17, 0xc244, 0x413d, 0x8e, 0xbf, 0xe8, 0x7e, 0x10, 0x9a, 0xa8, 0xd4	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_INVALID_PARAMETER** with a *Token* value of **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** with a *Token* value of **NULL**. The return status must be **EFI_NVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.8.2	0xddc80d3b, 0x448d, 0x4ef9, 0xab, 0x74, 0x88, 0x47, 0xa7, 0xc9, 0x7c, 0xa8	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_INVALID_PARAMETER** with a *Token*->*Filename* value of **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** with *Token*->*Filename* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.3	0x97304d43, 0x1101, 0x4b76, 0x90, 0x70, 0x66, 0x85, 0x62, 0x9e, 0xb3, 0xa3	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_INVALID_PARAMETER** when *Token->OptionCount* is not 0 and *Token->OptionList* is **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** when *Token*-> *OptionCount* is not 0 and *Token*->*OptionList* is **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.8.4	0xf061683f, 0xb39e, 0x42af, 0x92, 0x86, 0x9f, 0x18, 0xcc, 0xc7, 0xc0, 0x8d	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_INVALID_PARAMETER** when both *Token*->*Buffer* and *Token*-> *PacketNeeded* are **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** when both *Token*->*Buffer* and *Token*-> *PacketNeeded* are **NULL**. The return status must be **EFI_NVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.5	0xa2d02347, 0x9410, 0x49b3, 0xa9, 0xd2, 0xd7, 0x1a, 0xf4, 0xc5, 0xa7, 0x34	`EFI_MTFTP4_PROTOCOL.WriteFile()` - returns `EFI_INVALID_PARAMETER` with an `OverrideData.GatewayIp` value of invalid.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.WriteFile()` with an `OverrideData.GatewayIp` value of invalid. The return status must be `EFI_INVALID_PARAMETER`. 4. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment..
5.26.2.8.6	0xe8f09c7b, 0x2cf3, 0x482e, 0x93, 0xc6, 0x4f, 0x45, 0x85, 0x3a, 0x43, 0x0c	`EFI_MTFTP4_PROTOCOL.WriteFile()` - returns `EFI_INVALID_PARAMETER` with an `OverrideData.ServerIp` value of invalid.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.WriteFile()` with an *`OverrideData.ServerIp`* value of invalid. The return status must be `EFI_INVALID_PARAMETER`. 4. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.7	0x069921c9, 0x8f37, 0x45b6, 0xa4, 0x98, 0xa3, 0x2f, 0xc9, 0xb5, 0x8d, 0x50	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_INVALID_PARAMETER** when **OverrideData.GatewayIp** is not in the same subnet with *StationIp*.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** when **OverrideData.GatewayIp** is not in the same subnet with *StationIp*. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.8.8	0xb95d36a6, 0x091e, 0x444b, 0x9d, 0xd7, 0x30, 0x4c, 0x9e, 0x59, 0xab, 0x81	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_NOT_STARTED** when the EFI MTFTPv4 Protocol driver having not been started.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** with all valid parameters. The return status must be **EFI_NOT_STARTED**. 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.9	0x67021dd5, 0xf97d, 0x4783, 0x8d, 0xe2, 0x93, 0x6e, 0x6c, 0x5a, 0xe5, 0xeb	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_ACCESS_DENIED** when calling **EFI_MTFTP4_PROTOCOL.WriteFile** again before the first call ends.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** for the first time with all valid parameters. 4. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** for the second time with the same *Token* before the first call ends. The return status must be **EFI_ACCESS_DENIED**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.8.10	0x4a445105, 0xf332, 0x4251, 0xb1, 0x5c, 0x10, 0x5c, 0x27, 0xeb, 0x67, 0x09	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_UNSUPPORTED** when one or more options in the *Token*->*OptionList* are in the unsupported list for this implementation.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** when one or more options in the *Token*->*OptionList* are in the unsupported list for this implementation. The return status must be **EFI_UNSUPPORTED**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.11	0x00ee8108, 0xb8ce, 0x4428, 0x9a, 0x58, 0x3c, 0xf3, 0x33, 0x3e, 0xf4, 0x9a	`EFI_MTFTP4_PROTOCOL.WriteFile()` - returns `EFI_TFTP_ERROR` when the client receives an MTFTPv4 ERROR packet during uploading.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.WriteFile()` with all valid parameters. 4. The server should capture the write request. If having captured the packet, send a normal OACK to the client. 5. Then OS side should capture the data packets. If having captured, OS side sends an error packet back. The return status must be `EFI_TFTP_ERROR`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.
5.26.2.8.12	0x0b05148f, 0x4f07, 0x413d, 0x8e, 0x47, 0x99, 0xbe, 0xac, 0x25, 0xc3, 0x4d	`EFI_MTFTP4_PROTOCOL.WriteFile()` - returns `EFI_ICMP_ERROR,` when server sends back an ICMP error packet, client should terminate the session.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.WriteFile()` with all valid parameters. 4. The server should capture the write request. If having captured the packet, server responds an ICMP error packet. The return status must be `EFI_ICMP_ERROR`. 5. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.13	0x26ac0f66, 0x2fa1, 0x4e91, 0x93, 0x14, 0xfe, 0x0f, 0x86, 0x93, 0x47, 0x4d	`EFI_MTFTP4_PROTOCOL.WriteFile()` - returns `EFI_ABORTED` when the user aborts the upload process in *CheckPacket* callback routine	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.WriteFile()` with *CheckPacket* callback set. 4. The server should capture the write request. If having captured the packet, server responds a normal OACK. The return status must be `EFI_ABORTED`. 5. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.
5.26.2.8.14	0x105a5b0c, 0x72cb, 0x4854, 0x95, 0xdd, 0x86, 0xd7, 0x28, 0x0d, 0xa6, 0x12	`EFI_MTFTP4_PROTOCOL.WriteFile()` - returns `EFI_ABORTED` when the user aborts the upload process in TimeoutCallback callback routine	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.WriteFile()` with TimeoutCallback callback set. 4. The server should capture the write request. If having captured the packet, server responds a normal OACK. The return status must be `EFI_ABORTED`. 5. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.15	0xcaeef509, 0x3240, 0x4675, 0xa2, 0x50, 0x0b, 0xaf, 0xb5, 0x5a, 0xcb, 0x16	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_ABORTED** when the user aborts the upload process in PacketNeeded callback routine	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** with PacketNeeded callback set. 4. The server should capture the write request. If having captured the packet, server responds a normal OACK. The return status must be **EFI_ABORTED**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.8.25	0xb76d5034, 0xbee6, 0x468a, 0xa1, 0xf2, 0xc6, 0x9f, 0x20, 0x0d, 0xa6, 0xae	**EFI_MTFTP4_PROTOCOL.WriteFile()** - to test the **EFI_INVALID_PARAMETER** conformance of **WriteFile** when one or more options in *Token.OptionList* have wrong format.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** with invalid timeout option value. The return status must be **EFI_INVALID_PARAMETER** 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.16	0xcc7a5aad, 0xe6ec, 0x4fa7, 0x97, 0x0a, 0xac, 0x30, 0xd6, 0x39, 0x20, 0x16	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_SUCCESS** when the user uploads a packet with data less than one block.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** with *BufferSize* set to 100. 4. The server should capture the write request. If having captured the packet, server responds a normal OACK. 5. The server should capture the only data packet sent from the client and respond ACK. The return status must be **EFI_SUCCESS**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.8.17	0x2649936f, 0x161c, 0x40c2, 0xa8, 0x53, 0xc0, 0xa4, 0xa3, 0x2e, 0xf2, 0x62	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_SUCCESS** when the user uploads a packet with data length equal to 1 byte.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** with *BufferSize* set to 1. 4. The server should capture the write request. If having captured the packet, server responds a normal OACK. 5. The server should capture the only data packet sent from the client and respond with an ACK packet. The return status must be **EFI_SUCCESS**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.18	0xbcbec9fd, 0x00d8, 0x494d, 0xa4, 0xff, 0x86, 0x98, 0xc4, 0xb0, 0x6a, 0x5a	`EFI_MTFTP4_PROTOCOL.WriteFile()` - returns `EFI_SUCCESS` when the user uploads a packet with override configuration data.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.WriteFile()` with override configuration data. 4. The server should capture the write request. If having captured the packet, server responds a normal OACK. 5. The server should capture the only data packet sent from the client and respond with an ACK packet. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.19	0x741101e7, 0x7888, 0x4bd8, 0xa2, 0xcb, 0x1d, 0xec, 0xb1, 0x34, 0x66, 0x31	`EFI_MTFTP4_PROTOCOL.WriteFile()` - returns `EFI_SUCCESS` when the server responds with an incorrect ack packet.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.WriteFile()` with all valid parameters. 4. The server should capture the write request. If having captured the packet, server responds an OACK. 5. The server should capture the only data packet sent from the client and responds with an incorrent ACK to the incorrect packet number followed by a correct ACK. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.20	0xa3c22b82, 0x5f14, 0x4419, 0x8f, 0xc6, 0xd7, 0x89, 0x88, 0xa9, 0x88, 0xe9	`EFI_MTFTP4_PROTOCOL.WriteFile()` - returns `EFI_SUCCESS` when the server responds WRQ with an ACK instead of OACK, so client sends data packet to server.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.WriteFile()` with all valid parameters. 4. The server should capture the write request. If having captured the packet, server responds with an ACK instead of an OACK. 5. The server should capture the only data packet sent from the client and respond with an ACK to this packet. The return status must be `EFI_SUCCESS.` 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.21	0x20787b06, 0x8766, 0x4ced, 0xb0, 0x25, 0x65, 0xfa, 0xf1, 0xd3, 0x6c, 0x7c	`EFI_MTFTP4_PROTOCOL.WriteFile()` - returns `EFI_SUCCESS` when the server replies WRQ with an invalid BlockNo ACK instead of OACK, client should ignore this packet and continue the normal process.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.WriteFile()` with all valid parameters. 4. The server should capture the write request. If having captured the packet, server responds with an invalid BlockNo ACK instead of an OACK. 5. The server should capture the only data packet sent from the client and responds with an ACK to this packet. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.22	0xf549a91c, 0x9d15, 0x45c7, 0xb2, 0xed, 0xa6, 0x7e, 0xff, 0x08, 0xc0, 0xf4	**`EFI_MTFTP4_PROTOCOL.WriteFile()`** - returns **`EFI_SUCCESS`** when the server replies DATA packet with an error ACK (*`BufferLen < sizeof(UINT16))`* instead of OACK, client should ignore this packet and continue the normal process.	1. Call **`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()`** to create a new **`EFI_MTFTP4_PROTOCOL`** child handle. 2. Call **`EFI_MTFTP4_PROTOCOL.Configure()`** with all valid parameters. 3. Call **`EFI_MTFTP4_PROTOCOL.WriteFile()`** with all valid parameters. 4. The server should capture the write request. If having captured the packet, server responds with a normal OACK. 5. The server should capture the only data packet sent from the client and replies with an error ACK (*`BufferLen < sizeof(UINT16))`* and a correct ACK , client should ignore this error ACK and continue the normal process. The return status must be **`EFI_SUCCESS`**. 6. Call **`EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()`** to destroy the newly created **`EFI_MTFTP4_PROTOCOL`** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.23	0x9ee2172f, 0xb96e, 0x4d13, 0x9e, 0x6c, 0xbd, 0x27, 0x44, 0x95, 0xee, 0xc6	**EFI_MTFTP4_PROTOCOL.WriteFile()** - returns **EFI_SUCCESS** when the client receives an unexpected OACK when waiting for ACK packet during uploading file.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.WriteFile()** with all valid parameters. 4. The server should capture the write request. If having captured the packet, server responds with ACK. 5. The server should capture the only data packet sent from the client and respond with an unexpected OACK and an ACK to this packet. The return status must be **EFI_SUCCESS**. 6. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.8.24	0x434974c8, 0x5f8c, 0x46d8, 0x89, 0x57, 0x4e, 0x03, 0xff, 0xfa, 0xa3, 0xc5	`EFI_MTFTP4_PROTOCOL.WriteFile()` - returns `EFI_SUCCESS` when the client receives an error server source port ACK in the case of waiting for ACK packet.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Configure()` with all valid parameters. 3. Call `EFI_MTFTP4_PROTOCOL.WriteFile()` with all valid parameters. 4. The server should capture the write request. If having captured the packet, server responds with a normal OACK. 5. The server should capture the only data packet sent from the client and replies with an error server source port ACK and a correct ACK for the packet; client should ignore this error ACK and continue the normal process. The return status must be `EFI_SUCCESS`. 6. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

22.2.9 ReadDirectory()

Number	GUID	Assertion	Test Description
5.26.2.9.1	0xc9e02ded, 0x0e98, 0x4162, 0x8d, 0x4c, 0x14, 0x58, 0xd0, 0x6a, 0xc7, 0xab	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - returns **EFI_INVALID_PARAMETER** with a *Token* value of **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** with a *Token* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.9.2	0x120fa0f3, 0xad22, 0x4d39, 0xb9, 0x00, 0xe5, 0x60, 0xdd, 0x8f, 0xe3, 0xb2	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - returns **EFI_INVALID_PARAMETER** with a *Token*->*Filename* value of **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** with a *Token*->*Filename* value of **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.9.3	0xe6339187, 0x07d0, 0x467f, 0x9b, 0x89, 0x5b, 0xf5, 0x6c, 0x2d, 0xf8, 0xe0	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - returns **EFI_INVALID_PARAMETER** when *Token->OptionCount* is not 0 and *Token->OptionList* is **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** when *Token->OptionCount* is not 0 and *Token->OptionList* is **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.9.4	0xc39cb583, 0x3fa4, 0x4c7f,0x9a, 0x93, 0xa5, 0xf9, 0x30, 0xf0, 0x42, 0x6c	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - returns **EFI_INVALID_PARAMETER** when both *Token->Buffer* and *Token->CheckPacket* are **NULL**.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** when both *Token->Buffer* and *Token->CheckPacket* are **NULL**. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.9.5	0xaf58aaf5, 0x3cd0, 0x47aa, 0x8b, 0x93, 0x4f, 0x7b, 0x8b, 0xe8, 0x4d, 0xf1	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - returns **EFI_INVALID_PARAMETER** with an *OverrideData*.GatewayIp value of invalid.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** with an *OverrideData*.**GatewayIp** value of invalid. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.9.6	0x7044e68a, 0x6ca9, 0x4b23, 0x9a, 0x50, 0x91, 0x85, 0x34, 0xa3, 0xca, 0xfb	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - returns **EFI_INVALID_PARAMETER** with an *OverrideData*.ServerIp value of invalid.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** with an *OverrideData*.**ServerIp** value of invalid. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.9.7	0x8bd21805, 0xec3c, 0x4041, 0xa4, 0xe4, 0x75, 0xf1, 0xa4, 0xec, 0xae, 0x4d	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - returns **EFI_INVALID_PARAMETER** when *OverrideData*.GatewayIp is not in the same subnet with *StationIp*.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** for *OverrideData*.GatewayIp is not in the same subnet with*StationIp* . The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.9.8	0x7ecf38c4, 0x4fc5, 0x4663, 0xa4, 0xc4, 0xc0, 0x48, 0x45, 0xfe, 0x59, 0x6b	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - returns **EFI_NOT_STARTED** while the EFI MTFTPv4 Protocol driver having not been started.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** with all valid parameters. The return status must be **EFI_NOT_STARTED**. 3. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.9.9	0x31599521, 0xb38b, 0x47c8, 0xa6, 0x39, 0xaf, 0x50, 0xe3, 0x30, 0xbe, 0x87	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - returns **EFI_UNSUPPORTED** when one or more options in the a *Token->OptionList* value of in the unsupported list of this implementation.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** with one or more options in the a *Token->OptionList* value of in the unsupported list of this implementation. The return status must be **EFI_UNSUPPORTED**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.9.10	0xefc6d249, 0x179f, 0x49a2, 0x96, 0x1c, 0x0d, 0x90, 0xe7, 0x79, 0x4c, 0xcb	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - returns **EFI_ICMP_ERROR** when the server responds with an ICMP error packet, client should terminate the session.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** with all valid parameters. 4. If OS side has captured the request, it sends out an ICMP error packet. The return status must be **EFI_ICMP_ERROR**. 5. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

Number	GUID	Assertion	Test Description
5.26.2.9.12	0xab9bacfb, 0x79ee, 0x41e5, 0xb9, 0xe9, 0x40, 0x31, 0x7a, 0xf1, 0xcc, 0x64	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - test the **EFI_INVALID_PARAMETER** conformance of **ReadDirectory()** when one or more options in *Token.OptionList* have wrong format.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** with invalid timeout option value. The return status must be **EFI_INVALID_PARAMETER**. 4. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.
5.26.2.9.11	0x968731a3, 0x01e8, 0x44d7, 0xad, 0xba, 0x70, 0x88, 0x80, 0x8c, 0x99, 0xe1	**EFI_MTFTP4_PROTOCOL.ReadDirectory()** - returns **EFI_SUCCESS** - read a list of files on the MTFTPv4 server that are logically (or operationally) related to *Token*->*FileName*.	1. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()** to create a new **EFI_MTFTP4_PROTOCOL** child handle. 2. Call **EFI_MTFTP4_PROTOCOL.Configure()** with all valid parameters. 3. Call **EFI_MTFTP4_PROTOCOL.ReadDirectory()** with all valid parameters. 4. If OS side has captured the request, it sends out a normal OACK. 5. Then OS side should capture the ack from the client and send back the only data packet. 6. Then OS side expects another ack. The return status must be **EFI_ICMP_ERROR**. 7. Call **EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()** to destroy the newly created **EFI_MTFTP4_PROTOCOL** child handle and clean up the environment.

22.2.10 Poll()

Number	GUID	Assertion	Test Description
5.26.2.10.1	0x57e97972, 0xa7a3, 0x4647, 0x95, 0x9a, 0x23, 0x29, 0x5b, 0x81, 0x2c, 0xfe	`EFI_MTFTP4_PROTOCOL.Poll()` - returns `EFI_NOT_STARTED` when the EFI MTFTPv4 Protocol driver having not been started.	1. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.CreateChild()` to create a new `EFI_MTFTP4_PROTOCOL` child handle. 2. Call `EFI_MTFTP4_PROTOCOL.Poll()` with all valid parameters. The return status must be `EFI_NOT_STARTED`. 3. Call `EFI_MTFTP4_SERVICE_BINDING_PROTOCOL.DestroyChild()` to destroy the newly created `EFI_MTFTP4_PROTOCOL` child handle and clean up the environment.

22.3 EFI_UDP6_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_UDP6_PROTOCOL Section.

22.3.1 CreateChild()

Number	GUID	Assertion	Test Description
5.26.3.1.1	0x1d3e7323, 0x5a46, 0x4fe3, 0xbf, 0x9d, 0x0a, 0xb8, 0xb1, 0xfd, 0xe7, 0x92	**EFI_UDP6_SERVICE_BINDING_PROTOCOL.CreateChild() - CreateChild()** returns **EFI_INVALID_PARAMETER** with a **NULL** *ChildHandle*.	Call **CreateChild()** with a **NULL** *ChildHandle*, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.1.2	0x8872614e, 0x51d5, 0x434d, 0xb8, 0x71, 0x20, 0x30, 0x4f, 0xbe, 0x04, 0x92	**EFI_UDP6_SERVICE_BINDING_PROTOCOL.CreateChild() - CreateChild()** returns **EFI_SUCCESS** with a valid *ChildHandle*.	Call **CreateChild()** with a valid *ChildHandle*, the return status should be **EFI_SUCCESS.**

22.3.2 DestoryChild()

Number	GUID	Assertion	Test Description
5.26.3.2.1	0x25c39b09, 0xba61, 0x49f3, 0xa3, 0x58, 0x98, 0x11, 0x17, 0xd8, 0x14, 0x0e	**EFI_UDP6_SERVICE_BINDING_PROTOCOL.DestoryChild() - DestoryChild()** returns **EFI_INVALID_PARAMETER** with a **NULL** *ChildHandle*.	Call **DestoryChild()** with a **NULL** *ChildHandle*, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.2.2	0x1e938ebd, 0x425a, 0x4eb6, 0xbd, 0x12, 0x9c, 0xa2, 0xdc, 0xc4, 0x0b, 0x4c	**EFI_UDP6_SERVICE_BINDING_PROTOCOL.DestoryChild() - DestoryChild()** returns **EFI_SUCCESS** with a valid *ChildHandle*.	Call **DestoryChild()** with a valid *ChildHandle*, the return status should be **EFI_SUCCESS.**

22.3.3 GetModeData()

Number	GUID	Assertion	Test Description
5.26.3.3.1	0x920b75d9, 0xba94, 0x4e72, 0xb0, 0x4d, 0x77, 0xe5, 0x81, 0xe7, 0xcf, 0x91	`EFI_UDP6 PROTOCOL.GetModeData() - GetModeData()` returns `EFI_NOT_STARTED` with a not configured *`ChildHandle`*	Call `GetModeData()` with a not configured *`ChildHandle`*, the return status should be `EFI_NOT_STARTED.`
5.26.3.3.2	0x1a823790, 0xcaec, 0x413d, 0xbc, 0xf3, 0xe7, 0xfa, 0x70, 0xdf, 0x87, 0x6d	`EFI_UDP6 PROTOCOL.GetModeData() - GetModeData()` returns `EFI_SUCCESS` with valid parameters	5.26.3.3.2 to 5.26.3.3.4 belong to one case 1. Call `GetModeData()` with valid parameters, the return status should be `EFI_ SUCCESS.`
5.26.3.3.3	0xdb72ffca, 0xd3d9, 0x4837, 0x8f, 0x39, 0xf9, 0x67, 0x2e, 0x9d, 0x93, 0xab	Validate the *`IP6ModeData.IsConfigured`*	2. The value of *`IP6ModeData.IsConfigured`* should be `TRUE`.
5.26.3.3.4	0x923aecf2, 0xcfc6, 0x4497, 0x8c, 0x49, 0xe6, 0x74, 0x1c, 0x60, 0xc7, 0x66	Validate the *`Udp6ConfigData`*	3. The value of *`Udp6ConfigData`* should be same with the assigned configure data.

22.3.4 Configure()

Number	GUID	Assertion	Test Description
5.26.3.4.1	0x1c36e6e8, 0xf453, 0x41bb, 0x84, 0x6f, 0x0a, 0x67, 0x91, 0xa6, 0xe5, 0xe7	**EFI_UDP6 PROTOCOL.Configure() - Configure()** returns **EFI_INVALID_PARAMETER** with a *UdpConfigData.StationAddress* being neither zero nor one of the configured IP addresses in the underlying IPv6 driver	Call **Configure()** with a *UdpConfigData.StationAddress* being neither zero nor one of the configured IP addresses in the underlying IPv6 driver, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.4.2	0xef302465, 0x7ec6, 0x4652, 0xbb, 0xf0, 0x62, 0x73, 0xa5, 0x5a, 0xd5, 0x52	**EFI_UDP6 PROTOCOL.Configure() - Configure()** returns **EFI_INVALID_PARAMETER** with a *UdpConfigData.RemoteAddress* being an invalid unicast IPv6 address if it is not zero.	Call **Configure()** with a *UdpConfigData RemoteAddress* being an invalid unicast IPv6 address if it is not zero, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.4.3	0xe146a746, 0x2985, 0x4a7b, 0x92, 0xa5, 0x08, 0x44, 0x8d, 0x41, 0x69, 0x03	**EFI_UDP6 PROTOCOL.Configure() - Configure()** returns **EFI_ALREADY_STARTED** with a *ChildHandle* instance has already been started/configured.	Call **Configure()** with a *ChildHandle* instance has already been started/configured, the return status should be **EFI_ALREADY_STARTED.**
5.26.3.4.4	0x3522ad76, 0xe7aa, 0x4477, 0x9a, 0x41, 0xb7, 0xdc, 0xd6, 0xff, 0x7f, 0xf2	**EFI_UDP6 PROTOCOL.Configure() - Configure()** returns **EFI_ACCESS_DENIED** with *UdpConfigData. AllowDuplicatePort* is **FALSE** and *UdpConfigData.StationPort* is already used by other instance.	Call **Configure()** with *UdpConfigData. AllowDuplicatePort* is **FALSE** and *UdpConfigData.StationPort* is already used by other instance, the return status should be **EFI_ACCESS_DENIED.**
5.26.3.4.5	0x370fcb11, 0x68de, 0x4c01, 0xb0, 0xce, 0x64, 0x53, 0xb0, 0x94, 0x8f, 0xb5	**EFI_UDP6 PROTOCOL.Configure() - Configure()** returns **EFI_SUCCESS** with valid parameters	5.26.3.4.5 to 5.26.3.4.9 belong to one case. 1. Call **Configure()** with valid parameters, the return status should be **EFI_SUCCESS.**

Number	GUID	Assertion	Test Description
5.26.3.4.6	0xd6c84689, 0x0df8, 0x4f69, 0xa6, 0xd0, 0x76, 0x92, 0x89, 0xd0, 0x7d, 0x20	`EFI_UDP6 PROTOCOL.GetModeData() - GetModeData()` returns `EFI_SUCCESS` with valid parameters	2. Call `GetModeData()` with valid parameters, the return status should be `EFI_SUCCESS.`
5.26.3.4.7	0x7c2f3112, 0x80e9, 0x4b59, 0x98, 0xd5, 0x06, 0x25, 0x8e, 0x3e, 0x5f, 0x9f	Validate the *IP6ModeData.IsConfigured* and *Udp6ConfigData*	3. The value of *IP6ModeData.IsConfigured* should be `TRUE`. The value of *Udp6ConfigData* should be same with the assigned configure data.
5.26.3.4.8	0xc3fbe729, 0x3f1d, 0x41df, 0x83, 0x66, 0x6f, 0x50, 0x45, 0xf7, 0xce, 0x74	`EFI_UDP6 PROTOCOL.Configure() - Configure()` returns `EFI_SUCCESS` with a `NULL` *Udp6ConfigData*	4. Call `Configure()` with a `NULL` *Udp6ConfigData*, the return status should be `EFI_SUCCESS.`
5.26.3.4.9	0xd5a2273d, 0x33f4, 0x4f98, 0xb0, 0x8e, 0x9a, 0xd4, 0xec, 0x49, 0x9c, 0x76	`EFI_UDP6 PROTOCOL.GetModeData() - GetModeData()` returns `EFI_NOT_STARTED` with valid parameters	5. Call `GetModeData()` with valid parameters, the return status should be `EFI_NOT_STARTED.`

22.3.5 Groups()

Number	GUID	Assertion	Test Description
5.26.3.5.1	0x1f290403, 0xaa9e, 0x4e3b, 0x94, 0xfb, 0x2d, 0x2b, 0xa0, 0x56, 0x6b, 0x22	**EFI_UDP6 PROTOCOL.Groups() – Groups()** returns **EFI_NOT_STARTED** with a not configured *ChildHandle*	Call **Groups()** with a not configured *ChildHandle*, the return status should be **EFI_NOT_STARTED.**
5.26.3.5.2	0xb1fd2421, 0x6e59, 0x4987, 0xb8, 0x28, 0x1c, 0x13, 0xb1, 0xe3, 0x60, 0x37	**EFI_UDP6 PROTOCOL.Groups() – Groups()** returns **EFI_INVALID_PARAMETER** with **TRUE** *JoinFlag* and an invalid *MulticaseAddress*	Call **Groups()** with **TRUE** *JoinFlag* and an invalid *MulticaseAddress*, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.5.3	0xd2d32833, 0x51b6, 0x4c1b, 0x9a, 0x1c, 0x08, 0x11, 0xe6, 0xc1, 0xef, 0x4a	**EFI_UDP6 PROTOCOL.Groups() – Groups()** returns **EFI_ALREADY_STARTED** with **TRUE** *JoinFlag* and an *MulticaseAddress* which has already been in the group table.	Call **Groups()** with **TRUE** *JoinFlag* and an *MulticaseAddress* which has already been in the group table, the return status should be **EFI_ALREADY_STARTED.**
5.26.3.5.4	0x68c084c2, 0x55ef, 0x488a, 0x93, 0x24, 0xf9, 0x7b, 0x64, 0xbc, 0xbf, 0x03	**EFI_UDP6 PROTOCOL.Groups() – Groups()** returns **EFI_NOT_FOUND** with **FALSE** *JoinFlag* and an *MulticaseAddress* which is not in the group table.	5.26.3.5.4 to 5.26.3.5.7 belong to one case. 1. Call **Groups()** with **FALSE** *JoinFlag* and an *MulticaseAddress* which is not in the group table, the return status should be **EFI_NOT_FOUND.**
5.26.3.5.5	0xf16ff0fc, 0x074a, 0x460e, 0xa1, 0x11, 0x5f, 0x9e, 0xd3, 0x35, 0x9c, 0xac	**EFI_UDP6 PROTOCOL.Groups() – Groups()** returns **EFI_SUCCESS** with **TRUE** *JoinFlag* and an *MulticaseAddress* which is not in the group table.	2. Call **Groups()** with **TRUE** *JoinFlag* and an *MulticaseAddress* which is not in the group table, the return status should be **EFI_SUCCESS.**
5.26.3.5.6	0x60253644, 0x6c0e, 0x4662, 0xbd, 0x4c, 0x63, 0xc8, 0xde, 0xb1, 0x0c, 0x21	**EFI_UDP6 PROTOCOL.Groups() – Groups()** returns **EFI_SUCCESS** with **FALSE** *JoinFlag* and an *MulticaseAddress* which has been inserted in the group table.	3. Call **Groups()** with **FALSE** *JoinFlag* and an *MulticaseAddress* which has been inserted in the group table, the return status should be **EFI_SUCCESS.**

Number	GUID	Assertion	Test Description
5.26.3.5.7	0x5200ac0c, 0x0adb, 0x4a14, 0xa8, 0xbf, 0xbd, 0x42, 0xeb, 0x68, 0x2d, 0x8e	**EFI_UDP6 PROTOCOL.Groups() - Groups()** returns **EFI_NOT_FOUND** with **FALSE** *JoinFlag* and an *MulticaseAddress* which has been removed from the group table.	4. Call **Groups()** with **FALSE** *JoinFlag* and an *MulticaseAddress* which has been removed from the group table, the return status should be **EFI_NOT_FOUND.**
5.26.3.5.8	0x05df343c, 0xaff4, 0x4dc5, 0x8b, 0xa5, 0xd7, 0x76, 0x63, 0x12, 0x89, 0x25	**EFI_UDP6 PROTOCOL.Groups() - Groups()** returns **EFI_SUCCESS** with **TRUE** *JoinFlag* and an *MulticaseAddress* which is not in the group table.	5.26.3.5.8 to 5.26.3.5.11 belong to one case. 1. Call **Groups()** with **TRUE** *JoinFlag* and an *MulticaseAddress* which is not in the group table, the return status should be **EFI_SUCCESS.**
5.26.3.5.9	0x24602ea3, 0x6bb2, 0x49cf, 0xac, 0x38, 0xb0, 0x13, 0x85, 0x5c, 0xc8, 0xb9	**EFI_UDP6 PROTOCOL.GetModeData() - GetModeData()** returns **EFI_SUCCESS** with valid parameters. Check the *Ip6ModeData.GroupCount* and *Ip6ModeData.GroupTable*	2. Call **GetModeData()** with valid parameters, the return status should be **EFI_SUCCESS.** *Ip6ModeData.GroupCount* and *Ip6ModeData.GroupTable* should be reasonable.
5.26.3.5.10	0x6aabe731, 0x0de1, 0x4643, 0x82, 0x4e, 0x18, 0x0c, 0x65, 0x4a, 0xac, 0x0c	**EFI_UDP6 PROTOCOL.Groups() - Groups()** returns **EFI_SUCCESS** with **FALSE** *JoinFlag* and an *MulticaseAddress* which has been inserted in the group table.	3. Call **Groups()** with **FALSE** *JoinFlag* and an *MulticaseAddress* which has been inserted in the group table, the return status should be **EFI_SUCCESS.**
5.26.3.5.11	0xe9d7c7e6, 0xfc75, 0x48ef, 0xb9, 0x46, 0x00, 0xda, 0x5d, 0xe4, 0xcd, 0xea	**EFI_UDP6 PROTOCOL.GetModeData() - GetModeData()** returns **EFI_SUCCESS** with valid parameters. Check the *Ip6ModeData.GroupCount*	4. Call **GetModeData()** with valid parameters, the return status should be **EFI_SUCCESS.** *Ip6ModeData.GroupCount* should be reasonable.

22.3.6 Transmit()

Number	GUID	Assertion	Test Description
5.26.3.6.1	0x845b6a05, 0x23f3, 0x4c4f, 0x8d, 0xbc, 0xc0, 0xd3, 0x69, 0x9b, 0x76, 0x46	**EFI_UDP6 PROTOCOL.Transmit() - Transmit()** returns **EFI_NOT_STARTED** with a not configured *ChildHandle*	Call **Transmit()** with a not configured *ChildHandle*, the return status should be **EFI_NOT_STARTED.**
5.26.3.6.2	0x71c15402, 0x7d5c, 0x4b8c, 0xb9, 0xa5, 0xfd, 0xe5, 0x3e, 0x68, 0xed, 0x22	**EFI_UDP6 PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with a **NULL** *Token*	Call **Transmit()** with a **NULL** *Token*, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.6.3	0x12795cad, 0xdbbe, 0x41cd, 0x84, 0x57, 0x5f, 0xae, 0x7d, 0x72, 0x07, 0x2a	**EFI_UDP6 PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with a **NULL** *Token->Event*	Call **Transmit()** with a **NULL** *Token->Event*, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.6.4	0xbfcd7c31, 0xcb6f, 0x4cfd, 0xb9, 0xe2, 0x01, 0xd7, 0x5c, 0x6b, 0x44, 0xfa	**EFI_UDP6 PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with a **NULL** *Token->Packet.TxData*	Call **Transmit()** with a **NULL** *Token->Packet.TxData*, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.6.5	0x4c71fbec, 0x6cc6, 0x4cac, 0x89, 0x74, 0x67, 0xb5, 0x27, 0xbe, 0xef, 0xa3	**EFI_UDP6 PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with *Token->Packet.TxData-*>FragmentCount is Zero	Call **Transmit()** with *Token->Packet.TxData*->FragmentCount is Zero, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.6.6	0xe0e3d058, 0xbdc3, 0x4ed2, 0x9c, 0x39, 0xea, 0x10, 0x6b, 0xe5, 0xea, 0x7a	**EFI_UDP6 PROTOCOL.Transmit() - Transmit()** returns **EFI_INVALID_PARAMETER** with *Token->Packet.TxData->FragmentTable[0].FragmentLength* is Zero	Call **Transmit()** with *Token->Packet.TxData->FragmentTable[0].FragmentLength* is Zero, the return status should be **EFI_INVALID_PARAMETER.**

Number	GUID	Assertion	Test Description
5.26.3.6.7	0xbacc7fd3, 0x9a5c, 0x4ae6, 0xb6, 0xb3, 0x7f, 0x95, 0xc7, 0xda, 0xc4, 0xa2	**EFI_UDP6 PROTOCOL.Transmit() – Transmit()** returns **EFI_INVALID_PARAMETER** with a **NULL** *Token->Packet.TxData->FragmentTable[0].FragmentBuffer*	Call **Transmit()** with a **NULL** *Token->Packet.TxData->FragmentTable[0].FragmentBuffer*, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.6.8	0xf062269b, 0x66bb, 0x426a, 0x8e, 0xeb, 0x06, 0xd3, 0x0c, 0xd3, 0x30, 0x16	**EFI_UDP6 PROTOCOL.Transmit() – Transmit()** returns **EFI_INVALID_PARAMETER** with an invalid *Token->Packet.TxData->DataLength*	Call **Transmit()** with an invalid *Token->Packet.TxData->DataLength* which is not equal to the sum of the fragments length, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.6.9	0x5a3af347, 0xdf8a, 0x4a67, 0x80, 0x32, 0xa7, 0xd0, 0xa8, 0xcc, 0x2f, 0x97	**EFI_UDP6 PROTOCOL.Transmit() – Transmit()** returns **EFI_INVALID_PARAMETER** with a non-zero *Token->Packet.TxData->Udp6sessionData->DestinationAddress* which is not specified in Configure process	Call **Transmit()** with a non-zero *Token->Packet.TxData->Udp6sessionData->DestinationAddress* which is not specified in Configure process, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.6.10	0x52218200, 0xfffd, 0x4b78, 0x8b, 0x2b, 0xec, 0x17, 0x56, 0x2c, 0x3f, 0xd7	**EFI_UDP6 PROTOCOL.Transmit() – Transmit()** returns **EFI_INVALID_PARAMETER** with a zero *Token->Packet.TxData->Udp6sessionData->DestinationAddress* when DestinationAddress is unspecified when doing Configure process	Call **Transmit()** with a zero *Token->Packet.TxData->Udp6sessionData->DestinationAddress* when DestinationAddress is unspecified when doing Configure process, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.6.11	0x97434d51, 0x8e06, 0x49e9, 0x95, 0xd0, 0xfc, 0x3a, 0x03, 0xf9, 0x9c, 0xee	**EFI_UDP6 PROTOCOL.Transmit() – Transmit()** returns **EFI_INVALID_PARAMETER** with a **NULL** *Token->Packet.TxData->Udp6sessionData* and the instance's *UdpConfigData.RemoteAddress* is unspecified.	Call **Transmit()** with a **NULL** *Token->Packet.TxData->Udp6sessionData* and the instance's *UdpConfigData.RemoteAddress* is unspecified, the return status should be **EFI_INVALID_PARAMETER.**

Number	GUID	Assertion	Test Description
5.26.3.6.12	0x31b5da9f, 0xd866, 0x43c7, 0x8c, 0x2b, 0xf8, 0xd9, 0x7c, 0x5b, 0xdb, 0x12	**EFI_UDP6 PROTOCOL.Transmit() - Transmit()** returns **EFI_ACCESS_DENIED** with a *Token->Event* which has already been in the transmit queue.	Call **Transmit()** with a *Token->Event* which has already been in the transmit queue, the return status should be **EFI_ACCESS_DENIED.**
5.26.3.6.13	0x99e6bfb0, 0x903b, 0x4c6c, 0xa4, 0x6c, 0x9e, 0x51, 0x23, 0xdb, 0xdd, 0x4b	**EFI_UDP6 PROTOCOL.Transmit() - Transmit()** returns **EFI_BAD_BUFFER_SIZE** with a *Token->Packet.TxData->DataLength* which beyond the maximum udp6 packet size.	Call **Transmit()** with a *Token->Packet.TxData->DataLength* which beyond the maximum udp6 packet size, the return status should be **EFI_BAD_BUFFER_SIZE.**
5.26.3.6.14	0xaf040d05, 0xf0e3, 0x4348, 0x8f, 0x1d, 0xd9, 0x99, 0x90, 0xc7, 0x3d, 0x06	**EFI_UDP6 PROTOCOL.Transmit() - Transmit()** returns **EFI_SUCCESS** with valid parameters.	5.26.3.6.14 to 5.26.3.6.17 belong to one case. 1. Call **Transmit()** with valid parameters, the return status should be **EFI_SUCCESS.**
5.26.3.6.15	0x930f3d18, 0x3261, 0x4d17, 0xa3, 0xc0, 0x0d, 0xd1, 0xa6, 0x5d, 0x10, 0xe1	*Token->Event* should be signnaled	*Token->Event* should be signaled.
5.26.3.6.16	0x93873bee, 0x2136, 0x432e, 0xb0, 0x8f, 0xd7, 0x9d, 0xd9, 0xf9, 0xcf, 0x04	*Token->Status* should be **EFI_SUCCESS**	*Token->Status* should be **EFI_SUCCESS**.
5.26.3.6.17	0x30ca402a, 0xed8a, 0x4c69, 0x94, 0x7f, 0xa0, 0x4c, 0xd1, 0xbb, 0xaa, 0x58	The received packet content should be reasonable.	The received packet content should be reasonable.

22.3.7 Receive()

Number	GUID	Assertion	Test Description
5.26.3.7.1	0xb5c83b2c, 0x66c1, 0x4ea5, 0xba, 0x41, 0x6c, 0xc4, 0x85, 0xb2, 0x58, 0xaf	**EFI_UDP6 PROTOCOL.Receive() – Receive()** returns **EFI_NOT_STARTED** with a not configured *ChildHandle*	Call **Receive()** with a not configured *ChildHandle*, the return status should be **EFI_NOT_STARTED.**
5.26.3.7.2	0xc5c9fd31, 0xf095, 0x473f, 0xaf, 0x53, 0x87, 0x16, 0xc8, 0x51, 0x58, 0x9d	**EFI_UDP6 PROTOCOL.Receive() – Receive()** returns **EFI_INVALID_PARAMETER** with a **NULL** *Token*	Call **Receive()** with a **NULL** *Token*, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.7.3	0xa8916a19, 0xecf7, 0x4392, 0xa1, 0x65, 0xc0, 0x6e, 0x1b, 0xff, 0xc1, 0xe6	**EFI_UDP6 PROTOCOL.Receive() – Receive()** returns **EFI_INVALID_PARAMETER** with a **NULL** *Token->Event*	Call **Receive()** with a **NULL** *Token->Event*, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.7.14	0x17a43441, 0x0701, 0x446b, 0xab, 0x37, 0x4c, 0xd9, 0x23, 0xcf, 0xc1, 0x43	**EFI_UDP6 PROTOCOL.Receive() – Receive()** returns **EFI_ACCESS_DENIED** with a *Token->Event* which has already been in the transmit queue.	Call **Receive()** with a *Token->Event* which has already been in the transmit queue, the return status should be **EFI_ACCESS_DENIED.**
5.26.3.7.5	0x3166ca55, 0x6f3f, 0x4748, 0xbc, 0x48, 0xf7, 0xb6, 0x86, 0x35, 0x9d, 0xcc	**EFI_UDP6 PROTOCOL.Receive() – Receive()** returns **EFI_SUCCESS** with valid parameters.	5.26.3.7.5 to 5.26.3.7.8 belong to one case. 1. Call **Receive()** with valid parameters, the return status should be **EFI_SUCCESS.**
5.26.3.7.6	0xb5e37f49, 0xc13a, 0x4c80, 0x9d, 0x37, 0x9b, 0xb6, 0x96, 0xb8, 0x14, 0xe7	*Token->Event* should be signaled	*Token->Event* should be signaled.
5.26.3.7.7	0x96a78bb2, 0x8d5d, 0x4ed1, 0x9e, 0xc5, 0xc7, 0x34, 0x28, 0x61, 0x1e, 0x7d	The received packet content should be reasonable.	The received packet content should be reasonable.

Number	GUID	Assertion	Test Description
5.26.3.7.8	0x90b87634, 0x1da5, 0x4f26, 0x8c, 0x78, 0x82, 0xba, 0xd5, 0x4a, 0xc8, 0xfe	*Token->Status* should be **EFI_SUCCESS**	*Token->Status* should be **EFI_SUCCESS**.

22.3.8 Cancel()

Number	GUID	Assertion	Test Description
5.26.3.8.1	0xd0aafd24, 0xa340, 0x40f4, 0xba, 0x48, 0xe1, 0x59, 0x86, 0xc7, 0x78, 0x79	**EFI_UDP6 PROTOCOL.Cancel() – Cancel()** returns **EFI_NOT_STARTED** with a not configured *ChildHandle*	Call **Cancel()** with a not configured *ChildHandle*, the return status should be **EFI_NOT_STARTED.**
5.26.3.8.2	0x063478c3, 0x207d, 0x4b82, 0x96, 0xf5, 0x0f, 0xbf, 0xee, 0x2f, 0xac, 0x5f	**EFI_UDP6 PROTOCOL.Cancel() – Cancel()** returns **EFI_NOT_FOUND** with a *Token* which hasn't been inserted into both transmit and receive queue.	Call **Cancel()** with a *Token* which hasn't been inserted into both transmit and receive queue, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.8.3	0xed1466df, 0xccc6, 0x412e, 0xbe, 0xda, 0xb9, 0x87, 0xc8, 0x37, 0x2b, 0x6f	**EFI_UDP6 PROTOCOL.Cancel() – Cancel()** returns **EFI_NOT_FOUND** with a *Token* which has been removed into both transmit and receive queue.	Call **Cancel()** with a *Token* which has been removed into both transmit and receive queue, the return status should be **EFI_INVALID_PARAMETER.**
5.26.3.8.4	0xebe8e81e, 0x632c, 0x4aa9, 0xa8, 0x50, 0x27, 0xb7, 0x32, 0x63, 0xd0, 0x62	**EFI_UDP6 PROTOCOL.Cancel() – Cancel()** returns **EFI_SUCCESS** with valid parameters.	5.26.3.8.4 to 5.26.3.8.6 belong to one case. 1. Call **Cancel()** with valid parameters, the return status should be **EFI_SUCCESS.**
5.26.3.8.5	0x616b87c1, 0xa5f9, 0x4195, 0x81, 0x38, 0x9c, 0xb8, 0xcd, 0x3c, 0x64, 0x50	*Token->Event* should be signaled	*Token->Event* should be signaled.
5.26.3.8.6	0x1280bba6, 0x5d60, 0x43ae, 0xba, 0x36, 0x2d, 0xce, 0x08, 0x79, 0x5e, 0x57	*Token->Status* should be **EFI_SUCCESS**	*Token->Status* should be **EFI_SUCCESS**.

22.4 EFI_MTFTP6_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_MTFTP6_PROTOCOL Section.

22.4.1 CreateChild()

Number	GUID	Assertion	Test Description
5.26.4.1.1	0xed279b2f, 0x0fb1, 0x4f84, 0x8c, 0x11, 0x69, 0x36, 0x88, 0x0f, 0x94, 0x48	`EFI_MTFTP6_SERVICE_BINDING_PROTOCOL.CreateChild() - CreateChild()` returns `EFI_INVALID_PARAMETER` with a `NULL` *`ChildHandle`*.	Call `CreateChild()` with a `NULL` *`ChildHandle`*, the return status should be `EFI_INVALID_PARAMETER.`
5.26.4.1.2	0x758b358d, 0x4bf0, 0x4bcc, 0x82, 0x6c, 0xe4, 0xad, 0x40, 0xe8, 0x29, 0x6e	`EFI_MTFTP6_SERVICE_BINDING_PROTOCOL.CreateChild() - CreateChild()` returns `EFI_SUCCESS` with the 1st valid *`ChildHandle`*.	5.26.4.1.2 to 5.26.4.1.5 belong to one case. 1. Call `CreateChild()` with the 1st valid *`ChildHandle`*, the return status should be `EFI_SUCCESS`.
5.26.4.1.3	0x5446dbb2, 0xbf0b, 0x4685, 0x88, 0xf4, 0x3b, 0x14, 0x3e, 0x2b, 0xdd, 0x1b	`EFI_MTFTP6_SERVICE_BINDING_PROTOCOL.CreateChild() - CreateChild()` returns `EFI_SUCCESS` with the 2nd valid *`ChildHandle`*.	2. Call `CreateChild()` with the 2nd valid *`ChildHandle`*, the return status should be `EFI_SUCCESS`.
5.26.4.1.4	0x6a61e0bd, 0xd760, 0x4788, 0x85, 0xc9, 0x4b, 0x45, 0xe2, 0x9e, 0x7e, 0x02	`EFI_MTFTP6_SERVICE_BINDING_PROTOCOL.DestroyChild() - DestroyChild()` returns `EFI_SUCCESS` with the 2nd valid *`ChildHandle`*.	3. Call `DestroyChild()` with the 2nd valid *`ChildHandle`*, the return status should be `EFI_SUCCESS`.
5.26.4.1.5	0x0403eeee, 0x34d6, 0x47f4, 0x80, 0xcf, 0x28, 0x44, 0xa1, 0x7e, 0xfb, 0x7a	`EFI_MTFTP6_SERVICE_BINDING_PROTOCOL.DestroyChild() - DestroyChild()` returns `EFI_SUCCESS` with the 1st valid *`ChildHandle`*.	4. Call `DestroyChild()` with the 1st valid *`ChildHandle`*, the return status should be `EFI_SUCCESS`.

22.4.2 DestroyChild ()

Number	GUID	Assertion	Test Description
5.26.4.2.1	0xc4bdecde, 0xc89f, 0x4402, 0x9e, 0x9b, 0x2e, 0xac, 0xdd, 0xd6, 0xf7, 0xa6	**EFI_MTFTP6_SERVICE_BINDING_PROTOCOL.DestroyChild() - DestroyChild()** returns **EFI_INVALID_PARAMETER** with a **NULL** *ChildHandle*.	Call **DestroyChild()** with a **NULL** *ChildHandle*, the return status should be **EFI_INVALID_PARAMETER**.

22.4.3 GetModeData()

Number	GUID	Assertion	Test Description
5.26.4.3.1	0x2d5eae25, 0x9fda, 0x47c9, 0x80, 0x14, 0xf3, 0x34, 0xf0, 0x1e, 0x67, 0x12	**EFI_MTFTP6_PROTOCOL.GetModeData() - GetModeData()** returns **EFI_INVALID_PARAMETER** with **NULL ModeData**	Call **GetModeData()** with **NULL ModeData,** the return status should be **EFI_INVALID_PARAMETER**.
5.26.4.3.2	0x2a17e0f5, 0x6eab, 0x4528, 0xb1, 0xef, 0x4e, 0x99, 0x77, 0xd0, 0xc4, 0xa7	**EFI_MTFTP6_PROTOCOL.GetModeData() - GetModeData()** returns **EFI_SUCCESS** with the valid parameters.	5.26.4.3.2 to 5.26.4.3.3 belong to one case. 1. Call **CreateChild()** to create an MTFTP6 instance. 2. Call **Configure()** to initialize the MTFTP6 instance. 3. Call **GetModeData()** with the valid parameters, the return status should be **EFI_SUCCESS**.
5.26.4.3.3	0x51c6056a, 0x9582, 0x444a, 0xba, 0x84, 0x0d, 0xd5, 0xe4, 0x93, 0xb3, 0xa0	**Mtftp6ModeData.ConfigData** should be the same as previous set **ConfigData**.	4. **Mtftp6ModeData.ConfigData** should be the same as previous set **ConfigData**. 5. Call **DestroyChild()** to destroy the MTFTP6 instance.

22.4.4 Configure()

Number	GUID	Assertion	Test Description
5.26.4.4.1	0x2a946231, 0xa817, 0x45ed, 0x88, 0x59, 0x44, 0x42, 0xd5, 0x6d, 0x53, 0x45	`EFI_MTFTP6_PROTOCOL.Configure()` - `Configure()` returns `EFI_INVALID_PARAMETER` when StationIP is neither zero nor a configured IP address.	Call `Configure()` when StationIP is neither zero nor a configured IP address, the return status should be `EFI_INVALID_PARAMETER`.
5.26.4.4.2	0x02caf586, 0xff1c, 0x41e6, 0xb6, 0x5b, 0x6f, 0xd0, 0x22, 0xa4, 0x60, 0x14	`EFI_MTFTP6_PROTOCOL.Configure()` - `Configure()` returns `EFI_INVALID_PARAMETER` when ServerIp is an invalid unicast IPv6 address.	Call `Configure()` when ServerIp is an invalid unicast IPv6 address, such as ff02::1, the return status should be `EFI_INVALID_PARAMETER`.
5.26.4.4.3	0x9e06f1d5, 0xb888, 0x4976, 0x9c, 0x39, 0x6a, 0xcf, 0x26, 0x94, 0x2d, 0x76	`EFI_MTFTP6_PROTOCOL.Configure()` - `Configure()` returns `EFI_INVALID_PARAMETER` when ServerIp is an invalid unicast IPv6 address.	Call `Configure()` when ServerIp is an invalid unicast IPv6 address, such as ::, the return status should be `EFI_INVALID_PARAMETER`.
5.26.4.4.4	0xe5efe42a, 0x6539, 0x487d, 0x89, 0xe3, 0xb2, 0x88, 0x2a, 0xb1, 0xd7, 0xd4	`EFI_MTFTP6_PROTOCOL.Configure()` - `Configure()` returns `EFI_ACCESS_DENIED` when StationIp and LocalPort have already been used.	Call `Configure()` when StationIp and LocalPort have already been used, the return status should be `EFI_ACCESS_DENIED`.
5.26.4.4.5	0xcde4ae63, 0x74f6, 0x46fc, 0xa2, 0xae, 0x23, 0x2b, 0x39, 0x3a, 0x02, 0xd3	`EFI_MTFTP6_PROTOCOL.Configure()` - `Configure()` returns `EFI_ACCESS_DENIED` when call `Configure()` again to update the Configure Data without call `Configure()` with `NULL`.	Call `Configure()` again to update the Configure Data without call `Configure()` with `NULL`, the return status should be `EFI_ACCESS_DENIED`.
5.26.4.4.6	0x90337601, 0x85ca, 0x4152, 0x9f, 0x54, 0xdc, 0xac, 0x13, 0x87, 0x28, 0xdb	`EFI_MTFTP6_PROTOCOL.Configure()` - `Configure()` returns `EFI_SUCCESS` with valid `Mtftp6ConfigData`.	5.26.4.4.6 to 5.26.4.4.9 belong to one case. 1. Call `CreateChild()` to create an MTFTP6 instance. 2. Call `Configure()` with valid `Mtftp6ConfigData`, the return status should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.26.4.4.7	0x85bceaa3, 0x377a, 0x4847, 0x93, 0x4c, 0xa7, 0xea, 0x95, 0x1a, 0x4e, 0x57	EFI_MTFTP6_PROTOCOL.Configure() - Configure() returns EFI_SUCCESS when Mtftp6ConfigData is NULL.	3. Call Configure() when Mtftp6ConfigData is NULL, the return status should be EFI_SUCCESS.
5.26.4.4.8	0x62c85a93, 0x029d, 0x4bb2, 0xb5, 0x82, 0x25, 0x63, 0xae, 0x63, 0xba, 0xd7	EFI_MTFTP6_PROTOCOL.Configure() - Configure() returns EFI_SUCCESS with valid Mtftp6ConfigData in the second time.	4. Call Configure() with the valid Mtftp6ConfigData in the second time, the return status should be EFI_SUCCESS.
5.26.4.4.9	0xef42aa6a, 0x1c66, 0x4768, 0x8c, 0x4f, 0xc1, 0x18, 0x8a, 0xdc, 0x69, 0xfb	Call GetModeData() with the valid parameters, the Mtftp6ModeData.ConfigData should be the same as previous set ConfigData.	5. Call GetModeData() with the valid parameters, the Mtftp6ModeData.ConfigData should be the same as previous set ConfigData. 6. Call DestroyChild() to destroy the MTFTP6 instance.

22.4.5 GetInfo()

Number	GUID	Assertion	Test Description
5.26.4.5.1	0xed2fb03d, 0x8422, 0x46dc, 0xa4, 0xda, 0x31, 0xbe, 0x84, 0xa2, 0xf5, 0x0d	EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo() returns EFI_NOT_STARTED when the instance hasn't been configured.	Call GetInfo() when the instance hasn't been configured, the return status should be EFI_NOT_STARTED.
5.26.4.5.2	0xae921a1d, 0x1d87, 0x40a0, 0x90, 0x09, 0xe1, 0xc3, 0x30, 0x45, 0xd7, 0x7d	EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo() returns EFI_INVALID_PARAMETER when filename is NULL.	Call GetInfo() when filename is NULL, the return status should be EFI_INVALID_PARAMETER.
5.26.4.5.3	0x99321cf6, 0x6591, 0x4b71, 0xbd, 0xbe, 0x5a, 0xcb, 0xbe, 0x97, 0x32, 0x51	EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo() returns EFI_INVALID_PARAMETER when OptionCount isn't zero and OptionList is NULL.	Call GetInfo() when OptionCount isn't zero and OptionList is NULL, the return status should be EFI_INVALID_PARAMETER.
5.26.4.5.4	0x807e6ac5, 0x5ff8, 0x4e9c, 0x9f, 0xb0, 0x21, 0x9d, 0x3c, 0xf6, 0xae, 0x1c	EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo() returns EFI_INVALID_PARAMETER when one or more options in OptionList is wrong format.	Call GetInfo() when one or more options in OptionList is wrong format, the return status should be EFI_INVALID_PARAMETER.
5.26.4.5.5	0xdddb451a, 0x2d08, 0x45f2, 0xb4, 0x3c, 0x63, 0xde, 0xfc, 0xcc, 0x29, 0x42	EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo() returns EFI_INVALID_PARAMETER when PacketLength is NULL.	Call GetInfo() when PacketLength is NULL, the return status should be EFI_INVALID_PARAMETER.
5.26.4.5.6	0x1b915cd6, 0x34eb, 0x4a87, 0x8f, 0x18, 0x63, 0x25, 0x91, 0x1b, 0x80, 0x85	EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo() returns EFI_INVALID_PARAMETER when OverrideData.ServerIp is invalid unicast address.	Call GetInfo() when OverrideData.ServerIp is invalid unicast address, the return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.26.4.5.7	0x890ecac1, 0xd029, 0x4a8f, 0x99, 0x10, 0x57, 0x73, 0xa0, 0x70, 0xba, 0xff	EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo() returns EFI_INVALID_PARAMETER when one or more options in OptionList is unsupported.	Call GetInfo() when one or more options in OptionList is unsupported, the return status should be EFI_UNSUPPORTED.
5.26.4.5.8	0xa807dd98, 0x8d94, 0x42cd, 0x9b, 0x38, 0x2c, 0x4d, 0xa1, 0x43, 0xc1, 0xbc	EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo() returns EFI_TFTP_ERROR when a mtftp6 error packet received from the other side.	Call GetInfo() when a mtftp6 error packet received from the other side, the return status should be EFI_TFTP_ERROR.
5.26.4.5.9	0x8ea63309, 0x2824, 0x4186, 0x93, 0x39, 0xd4, 0x10, 0x44, 0xef, 0xf2, 0x36	EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo() returns EFI_TIMEOUT when there is no response from the other side.	Call GetInfo() when no response is sent from the other side, the return status should be EFI_TIMEOUT.
5.26.4.5.10	0x29b90725, 0x6662, 0x43f5, 0xa4, 0xe5, 0xb0, 0xb5, 0xfa, 0x26, 0x55, 0x38	EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo() returns EFI_PORT_UNREACHABLE when an ICMP port unreachable error packet was received.	Call GetInfo() when an ICMP port unreachable error packet was received, the return status should be EFI_PORT_UNREACHABLE.
5.26.4.5.11	0x01b2ee0f, 0xb879, 0x4475, 0x9e, 0x58, 0x7d, 0xff, 0x51, 0x13, 0x88, 0x87	EFI_MTFTP6_PROTOCOL. GetInfo () - GetInfo () returns EFI_SUCCESS with valid parameters.	5.26.4.5.11 to 5.26.4.5.12 belong to one case. 1. Call CreateChild() to create an MTFTP6 instance. 2. Call Configure() with valid Mtftp6ConfigData, the return status should be EFI_SUCCESS. 3. Call GetInfo() with valid parameters. 4. Host send MTFTP6 OACK packet. 5. Host receive the Ack for OACK 6. The return status of GetInfo() should be EFI_SUCCESS.
5.26.4.5.12	0x9ddd227a, 0x0734, 0x4d6b, 0xaf, 0xa9, 0xdb, 0xc1, 0xad, 0x10, 0xb7, 0xd3	Call ParseOptions() to parse the Packet, the content of EFI_MTFTP6_OPTION should be right.	7. Call ParseOptions() to parse the Packet, the content of EFI_MTFTP6_OPTION should be right.

Number	GUID	Assertion	Test Description
5.26.4.5.13	0x1257a949, 0xb84d, 0x43f6, 0x89,0x2b, 0x48,0x5f, 0x33,0x65, 0x82,0x12	**EFI_MTFTP6_PROTOCOL.GetInfo()** - **GetInfo()** returns **EFI_PORT_UNREACHABLE** when an ICMP port unreachable error packet was received.	Call **GetInfo()** when an ICMP port unreachable error packet was received, the return status should be **EFI_PORT_UNREACHABLE**.
5.26.4.5.14	0xd3688340, 0x7b29, 0x46cb, 0x98,0x05, 0x76,0xf0, 0xab,0xef, 0x78,0xc0	**EFI_MTFTP6_PROTOCOL.GetInfo()** - **GetInfo()** returns **EFI_NETWORK_UNREACHABLE** when an ICMP net unreachable error packet was received.	Call **GetInfo()** when an ICMP net unreachable error packet was received, the return status should be **EFI_NETWORK_UNREACHABLE**.
5.26.4.5.15	0x8cffd8f0,0xf8e7, 0x4e6c, 0x8e,0x2f, 0xbe,0xf5, 0xff,0xd8, 0xd4,0x8c	**EFI_MTFTP6_PROTOCOL.GetInfo()** - **GetInfo()** returns **EFI_HOST_UNREACHABLE** when an ICMP host unreachable error packet was received.	Call **GetInfo()** when an ICMP host unreachable error packet was received, the return status should be **EFI_HOST_UNREACHABLE**.

22.4.6 ParseOptions()

Number	GUID	Assertion	Test Description
5.26.4.6.1	0x165bba38, 0x2cc8, 0x4c86, 0xb5, 0x9a, 0x82, 0xd5, 0x33, 0xe0, 0x3d, 0x12	EFI_MTFTP6_PROTOCOL .ParseOptions() - ParseOptions() returns EFI_INVALID_PARAMETER when PacketLen is zero.	Call ParseOptions() when PacketLen is zero, the return status should be EFI_INVALID_PARAMETER.
5.26.4.6.2	0x46feb505, 0x82fd, 0x4d84, 0x98, 0x9f, 0x2a, 0x24, 0x70, 0xff, 0xf9, 0x1f	EFI_MTFTP6_PROTOCOL . ParseOptions() - ParseOptions() returns EFI_INVALID_PARAMETER when Packet is NULL.	Call ParseOptions() when Packet is NULL, the return status should be EFI_INVALID_PARAMETER.
5.26.4.6.3	0x2c5276ba, 0x0fed, 0x474f, 0x91, 0x79, 0x9d, 0xa1, 0xdb, 0x8e, 0x32, 0x19	EFI_MTFTP6_PROTOCOL . ParseOptions() - ParseOptions() returns EFI_INVALID_PARAMETER when Packet isn't a valid Mtftp6 packet.	Call ParseOptions() when Packet isn't a valid Mtftp6 packet, the return status should be EFI_INVALID_PARAMETER.
5.26.4.6.4	0x3bd37c27, 0xeaea, 0x474c, 0x92, 0xf7, 0xd4, 0x90, 0x68, 0x50, 0xb5, 0x54	EFI_MTFTP6_PROTOCOL . ParseOptions() - ParseOptions() returns EFI_INVALID_PARAMETER when when OptionCount is NULL.	Call ParseOptions() when OptionCount is NULL, the return status should be EFI_INVALID_PARAMETER.
5.26.4.6.5	0xd5918b06, 0x88cd, 0x4321, 0x90, 0x18, 0x3e, 0x3e, 0x1a, 0xd2, 0xcd, 0xa1	EFI_MTFTP6_PROTOCOL . ParseOptions() - ParseOptions() returns EFI_NOT_FOUND when no Options is found.	Call ParseOptions() when no Options is found, the return status should be EFI_NOT_FOUND.
5.26.4.6.6	0xad87d495, 0x9738, 0x4c86, 0x97, 0x2c, 0x63, 0xdb, 0xcd, 0x2b, 0xda, 0x84	EFI_MTFTP6_PROTOCOL . ParseOptions() - ParseOptions() returns EFI_PROTOCOL_ERROR when one or more of the option fields are not valid.	Call ParseOptions() when one or more of the option fields are not valid, the return status should be EFI_PROTOCOL_ERROR.

22.4.7 ReadFile()

Number	GUID	Assertion	Test Description
5.26.4.7.1	0x33346d27, 0x213b, 0x4137, 0xa0, 0x4e, 0xff, 0x79, 0xc3, 0x40, 0x82, 0x2a	EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile() returns EFI_NOT_STARTED when the instance hasn't been configured.	Call ReadFile() when the instance hasn't been configured. The return status should be EFI_NOT_STARTED.
5.26.4.7.2	0xfa4a5e44, 0x3823, 0x4273, 0xa8, 0x86, 0x7d, 0x95, 0xb4, 0xd9, 0x0d, 0xa1	EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile() returns EFI_INVALID_PARAMETER when Token is NULL.	Call ReadFile() when Token is NULL. The return status should be EFI_INVALID_PARAMETER.
5.26.4.7.3	0x2e09fd86, 0xfe91, 0x4490, 0x9f, 0x33, 0xa9, 0xdf, 0x38, 0x65, 0xf0, 0xdf	EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile() returns EFI_INVALID_PARAMETER when Token.Filename is NULL.	Call ReadFile() when Token.Filename is NULL. The return status should be EFI_INVALID_PARAMETER.
5.26.4.7.4	0x197e3225, 0xc6ba, 0x43ee, 0x8d, 0xf7, 0x31, 0x31, 0xbd, 0x71, 0x5e, 0x2e	EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile() returns EFI_INVALID_PARAMETER when OptionCount isn't zero and OptionList is NULL.	Call ReadFile() when OptionCount isn't zero and OptionList is NULL. The return status should be EFI_INVALID_PARAMETER.
5.26.4.7.5	0x983411c5, 0x040b, 0x4995, 0xbb, 0x0e, 0x80, 0xb8, 0x69, 0x3b, 0x6b, 0x9e	EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile() returns EFI_INVALID_PARAMETER when one or more options in Token.OptionList is wrong format.	Call ReadFile() when one or more options in Token.OptionList is wrong format. The return status should be EFI_INVALID_PARAMETER.
5.26.4.7.6	0x7fff6983, 0x39e5, 0x421f, 0x93, 0xb8, 0x3a, 0x16, 0x4d, 0x3a, 0x95, 0x34	EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile() returns EFI_INVALID_PARAMETER when one or more options in Token.Buffer and Token.CheckPacket are both NULL.	Call ReadFile() when one or more options in Token.Buffer and Token.CheckPacket are both NULL. The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.26.4.7.7	0xbdb9aaa3, 0x4efa, 0x41dc, 0x91, 0x22, 0xf9, 0x15, 0x04, 0x4f, 0x34, 0xac	`EFI_MTFTP6_PROTOCOL.ReadFile()` - `ReadFile()` returns `EFI_INVALID_PARAMETER` when one or more options in `Token.OverrideData.ServerIp` is not valid unicast IPv6 address.	Call `ReadFile()` when one or more options in `Token.OverrideData.ServerIp` is not valid unicast IPv6 address. The return status should be `EFI_INVALID_PARAMETER`.
5.26.4.7.8	0xf410b1c3, 0x5e50, 0x4389, 0x99, 0xac, 0x5f, 0x9f, 0xe7, 0xed, 0x47, 0x9d	`EFI_MTFTP6_PROTOCOL.ReadFile()` - `ReadFile()` returns `EFI_INVALID_PARAMETER` when one or more options in `Token.OptionList` is not supported.	Call `ReadFile()` when one or more options in `Token.OptionList` is not supported. The return status should be `EFI_UNSUPPORTED`.
5.26.4.7.9	0xb5b845cf, 0x1ac2, 0x4ba6, 0x88, 0x13, 0x35, 0xdc, 0xe6, 0x07, 0xec, 0x82	`EFI_MTFTP6_PROTOCOL.ReadFile()` - `ReadFile()` returns `EFI_BUFFER_TOO_SMALL` when `Token.BufferSize` isn't large enough to hold the download data in download process.	Call `ReadFile()` when `Token.BufferSize` isn't large enough to hold the download data in download process. The return status should be `EFI_BUFFER_TOO_SMALL`.
5.26.4.7.10	0x79f11d98, 0x4a0c, 0x4c2a, 0x8f, 0x48, 0x58, 0xa6, 0x04, 0x6c, 0x11, 0x94	`EFI_MTFTP6_PROTOCOL.ReadFile()` - `ReadFile()` returns `EFI_ABORTED` when current operation is aborted by user.	Call `ReadFile()` when current operation is aborted by user. The return status should be `EFI_ABORTED`.
5.26.4.7.11	0x99d1d01e, 0x23f4, 0x4877, 0x98, 0xe9, 0x6e, 0xa3, 0xb9, 0x98, 0x8e, 0x9d	`EFI_MTFTP6_PROTOCOL.ReadFile()` - `ReadFile()` returns `EFI_TFTP_ERROR` when a mtftp6 error packet received.	Call `ReadFile()` when a mtftp6 error packet was received. The return status should be `EFI_TFTP_ERROR`.
5.26.4.7.12	0x2e222488, 0xcab8, 0x40d5, 0xa6, 0x71, 0xac, 0xa6, 0x6c, 0x76, 0x58, 0x59	`EFI_MTFTP6_PROTOCOL.ReadFile()` - `ReadFile()` returns `EFI_PORT_UNREACHABLE` when a icmp6 port unreachable error packet was received.	Call `ReadFile()` when a icmp6 port unreachable error packet was received. The return status should be `EFI_PORT_UNREACHABLE`.

Number	GUID	Assertion	Test Description
5.26.4.7.13	0x36a6ebe2,0xdb79, 0x423f, 0xad,0x53, 0x9b,0xf1, 0x7d,0x1b, 0x4c,0x20	`EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo()` returns `EFI_NETWORK_UNREACHABLE` when an ICMP net unreachable error packet was received.	Call `GetInfo()` when an ICMP net unreachable error packet was received. The return status should be `EFI_NETWORK_UNREACHABLE`.
5.26.4.7.14	0x3215f20a, 0xec4f, 0x4666, 0x8d,0x6b, 0xe7,0x09, 0x21,0x65, 0x7a,0xa2	`EFI_MTFTP6_PROTOCOL.GetInfo() - GetInfo()` returns `EFI_HOST_UNREACHABLE` when an ICMP host unreachable error packet was received.	Call `GetInfo()` when an ICMP host unreachable error packet was received. The return status should be `EFI_HOST_UNREACHABLE`.
5.26.4.7.15	0x0d5a4c2a, 0xc87e, 0x41e4, 0xa8, 0x6b, 0xce, 0x62, 0x30, 0x7c, 0x84, 0x06	`EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile()` returns `EFI_TIMEOUT` when no response was received.	Call `ReadFile()` when no response was received. The return status should be `EFI_TIMEOUT`.
5.26.4.7.16	0xa29fb61f, 0x4f6c, 0x4e15, 0xaf, 0x96, 0xb7, 0x0c, 0xf2, 0x1c, 0xd7, 0x71	`EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile()` returns `EFI_SUCCESS` with valid parameters.	5.26.4.7.16 to 5.26.4.7.17 belong to one case. 1. Call `ReadFile()` with valid parameters. The return status should be `EFI_SUCCESS`.
5.26.4.7.17	0xea84cd69, 0x5550, 0x44a0, 0xbb, 0xe7, 0x0f, 0xc1, 0x5d, 0x08, 0xb8, 0x35	The `Token.Status` should be `EFI_SUCCESS`.	2. The `Token.Status` should be `EFI_SUCCESS`.
5.26.4.7.18	0x789c0d97, 0x68d8, 0x4a72, 0xa8, 0x7f, 0x66, 0x37, 0xcb, 0x6b, 0xb8, 0xe0	`EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile()` returns `EFI_SUCCESS` with valid parameters.	5.26.4.7.18 to 5.26.4.7.20 belong to one case. 1. Call `ReadFile()` with valid parameters, the return status should be `EFI_SUCCESS`.
5.26.4.7.19	0x9a991ff0, 0x84af, 0x4290, 0x85, 0x3b, 0x02, 0xe0, 0xb7, 0xe4, 0xe0, 0x28	The `Token.Event` should be signaled.	2. The `Token.Event` should be signaled.

Number	GUID	Assertion	Test Description
5.26.4.7.20	0xd90350a1, 0x7e65, 0x435f, 0xa3, 0x8f, 0x24, 0x27, 0xf1, 0x08, 0x8f, 0x5f	The `Token.Status` should be `EFI_SUCCESS`.	3. The `Token.Status` should be `EFI_SUCCESS`.
5.26.4.7.21	0xdf7f3d8e, 0x492e, 0x46ef, 0xb9, 0x8e, 0x26, 0x06, 0x9e, 0x85, 0x7a, 0x73	`EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile()` returns `EFI_SUCCESS` with valid parameters, `Token.Event` is not `NULL` and `Token.BufferSize` is not large enough.	5.26.4.7.21 to 5.26.4.7.23 belong to one case. 1. Call `ReadFile()` with valid parameters, `Token.Event` is not `NULL` and `Token.BufferSize` is not large enough, the return status should be `EFI_SUCCESS`.
5.26.4.7.22	0x5ff92824, 0x75a9, 0x4e39, 0xa3, 0xa0, 0xc6, 0x2d, 0x42, 0x09, 0x48, 0x78	The `Token.Event` should be signaled.	2. The `Token.Event` should be signaled.
5.26.4.7.23	0x5bd23489, 0xc9df, 0x4ce5, 0x84, 0xe9, 0x51, 0xa1, 0x88, 0xe2, 0x96, 0x3f	The `Token.Status` should be `EFI_BUFFER_TOO_SMALL`.	3. The `Token.Status` should be `EFI_BUFFER_TOO_SMALL`.
5.26.4.7.24	0x4f23a070, 0xd01c, 0x441c, 0x88, 0x36, 0x26, 0xc4, 0x00, 0x05, 0xda, 0x0b	`EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile()` returns `EFI_SUCCESS` with valid parameters, `Token.Event` is not `NULL` and current operation is aborted by user.	5.26.4.7.24 to 5.26.4.7.26 belong to one case. 1. Call `ReadFile()` with valid parameters, `Token.Event` is not `NULL` and current operation is aborted by user, the return status should be `EFI_SUCCESS`.
5.26.4.7.25	0xb22cb194, 0xd7db, 0x4141, 0x87, 0x8d, 0xab, 0xb7, 0x76, 0x9a, 0x12, 0xf6	The `Token.Event` should be signaled.	2. The `Token.Event` should be signaled.

Number	GUID	Assertion	Test Description
5.26.4.7.26	0x8a946d5c, 0xa820, 0x47c1, 0x83, 0xdf, 0x4b, 0x73, 0x9f, 0x52, 0x89, 0x53	The `Token.Status` should be `EFI_ABORTED`.	3. The `Token.Status` should be `EFI_ABORTED`.
5.26.4.7.27	0x11b9ec6c, 0xff52, 0x4279, 0x9a, 0x07, 0x64, 0x1b, 0xcb, 0xe5, 0x37, 0x73	`EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile()` returns `EFI_SUCCESS` with valid parameters, `Token.Event` is not `NULL` and a mtftp6 error packet was received.	5.26.4.7.27 to 5.26.4.7.29 belong to one case. 1. Call `ReadFile()` with valid parameters, `Token.Event` is not `NULL` and a mtftp6 error packet was received, the return status should be `EFI_SUCCESS`.
5.26.4.7.28	0x70e67e7f, 0x0a67, 0x4402, 0xa2, 0x63, 0x9a, 0xe9, 0x75, 0xcb, 0x7c, 0x71	The `Token.Event` should be signaled.	2. The `Token.Event` should be signaled.
5.26.4.7.29	0x35b45761, 0x9657, 0x4211, 0xb4, 0xfb, 0xed, 0x68, 0xe1, 0x98, 0xf4, 0x02	The `Token.Status` should be `EFI_TFTP_ERROR`.	3. The `Token.Status` should be `EFI_TFTP_ERROR`.
5.26.4.7.30	0x6aa2ecf0, 0xb01e, 0x4a8e, 0xb3, 0xdc, 0xd0, 0x54, 0xce, 0xb6, 0xa0, 0x83	`EFI_MTFTP6_PROTOCOL.ReadFile() - ReadFile()` returns `EFI_SUCCESS` with valid parameters, `Token.Event` is not `NULL` and a icmp6 error packet was received.	5.26.4.7.30 to 5.26.4.7.32 belong to one case. 1. Call `ReadFile()` with valid parameters, `Token.Event` is not `NULL` and a icmp6 error packet was received, the return status should be `EFI_SUCCESS`.
5.26.4.7.31	0x6794533c, 0xf4f6, 0x4972, 0x8c, 0xf4, 0x3a, 0xc6, 0x20, 0x20, 0x19, 0x4e	The `Token.Event` should be signaled.	2. The `Token.Event` should be signaled.
5.26.4.7.32	0xda706911, 0x98fd, 0x49a3, 0xa6, 0x55, 0x74, 0x75, 0xb5, 0x5a, 0x0d, 0x4f	The `Token.Status` should be `EFI_PORT_UNREACHABLE`.	3. The `Token.Status` should be `EFI_PORT_UNREACHABLE`.

Number	GUID	Assertion	Test Description
5.26.4.7.33	0xc80090b9, 0x0876, 0x4959, 0xbd, 0x80, 0x6c, 0x41, 0x40, 0x92, 0xac, 0x48	EFI_MTFTP6_PROTOCOL .ReadFile() - ReadFile() returns EFI_SUCCESS with valid parameters, Token.Event is not NULL and a icmp6 error packet was received.	5.26.4.7.33 to 5.26.4.7.35 belong to one case. 1. Call ReadFile() with valid parameters, Token.Event is not NULL and no response was received, the return status should be EFI_SUCCESS.
5.26.4.7.34	0xa8ce4013, 0x648f, 0x46d5, 0xa4, 0x89, 0xd3, 0x33, 0x1a, 0xee, 0x1d, 0x57	The Token.Event should be signaled.	2. The Token.Event should be signaled.
5.26.4.7.35	0xbb028e8c, 0xf45a, 0x4052, 0x83, 0x3b, 0x45, 0x81, 0xec, 0x1b, 0x62, 0x0f	The Token.Status should be EFI_TIMEOUT.	3. The Token.Status should be EFI_TIMEOUT.

22.4.8 WriteFile()

Number	GUID	Assertion	Test Description
5.26.4.8.1	0x3123cc65, 0x7cea, 0x4b5e, 0x92, 0xd9, 0x7d, 0x8c, 0xe4, 0x95, 0x3f, 0x4f	`EFI_MTFTP6_PROTOCOL.WriteFile() - WriteFile()` returns `EFI_NOT_STARTED` when the instance hasn't been configured.	Call `WriteFile()` when the instance hasn't been configured, the return status should be `EFI_NOT_STARTED`.
5.26.4.8.2	0x6738f74e, 0x3f6f, 0x48db, 0xaa, 0x1b, 0x8d, 0x38, 0xf8, 0x19, 0x4a, 0x61	`EFI_MTFTP6_PROTOCOL.WriteFile() - Writeile()` returns `EFI_INVALID_PARAMETER` when `Token` is `NULL`.	Call `WriteFile()` when `Token` is `NULL`, the return status should be `EFI_INVALID_PARAMETER`.
5.26.4.8.3	0x637d7d38, 0x102d, 0x4382, 0x9f, 0x95, 0x0f, 0xc3, 0x97, 0x84, 0x29, 0xcf	`EFI_MTFTP6_PROTOCOL.WriteFile() - WriteFile()` returns `EFI_INVALID_PARAMETER` when `Token.Filename` is `NULL`.	Call `WriteFile()` when `Token.Filename` is `NULL`, the return status should be `EFI_INVALID_PARAMETER`.
5.26.4.8.4	0xf39cdb05, 0xd139, 0x4dd7, 0x8c, 0x15, 0x41, 0x2f, 0x8b, 0xde, 0x04, 0xb3	`EFI_MTFTP6_PROTOCOL.WriteFile() - WriteFile()` returns `EFI_INVALID_PARAMETER` when `Token.OptionCount` isn't zero and `Token.OptionList` is `NULL`.	Call `WriteFile()` when `Token.OptionCount` isn't zero and `Token.OptionList` is `NULL`, the return status should be `EFI_INVALID_PARAMETER`.
5.26.4.8.5	0x9d3fcbac, 0xbc54, 0x46d9, 0x85, 0x41, 0x64, 0xe3, 0xaa, 0x41, 0x25, 0x58	`EFI_MTFTP6_PROTOCOL.WriteFile() - WriteFile()` returns `EFI_INVALID_PARAMETER` when one or more options in `Token.OptionList` is wrong format.	Call `WriteFile()` when one or more options in `Token.OptionList` is wrong format, the return status should be `EFI_INVALID_PARAMETER`.
5.26.4.8.6	0x181a05aa, 0xcd53, 0x4ba1, 0xb8, 0xf3, 0x98, 0x09, 0x9b, 0xd4, 0xa7, 0xe1	`EFI_MTFTP6_PROTOCOL.WriteFile() - WriteFile()` returns `EFI_INVALID_PARAMETER` when one or more options in `Token.Buffer` and `Token.PacketNeeded` are both `NULL`.	Call `WriteFile()` when one or more options in `Token.Buffer` and `Token.PacketNeeded` are both `NULL`, the return status should be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.26.4.8.7	0xb820e6cb, 0x5290, 0x4748, 0x94, 0x66, 0x8d, 0xc1, 0x99, 0x2e, 0x12, 0xc1	`EFI_MTFTP6_PROTOCOL.WriteFile() – WriteFile()` returns `EFI_INVALID_PARAMETER` when one or more options in `Token.OverrideData.ServerIp` is not valid unicast IPv6 address.	Call `WriteFile()` when one or more options in `Token.OverrideData.ServerIp` is not valid unicast IPv6 address, the return status should be `EFI_INVALID_PARAMETER`.
5.26.4.8.8	0x3854186d, 0x550e, 0x4006, 0xbe, 0xff, 0x1c, 0x52, 0x5b, 0xa4, 0x3e, 0xcf	`EFI_MTFTP6_PROTOCOL.WriteFile() – WriteFile()` returns `EFI_INVALID_PARAMETER` when one or more options in `Token.OptionList` is not supported.	Call `WriteFile()` when one or more options in `Token.OptionList` is not supported, the return status should be `EFI_UNSUPPORTED`.
5.26.4.8.9	0x54ae8e18, 0xd428, 0x48c1, 0xad, 0x32, 0xb5, 0x51, 0xda, 0x1e, 0x90, 0x7f	`EFI_MTFTP6_PROTOCOL.WriteFile() – WriteFile()` returns `EFI_SUCCESS` with valid parameters.	5.26.4.8.9 to 5.26.4.8.10 belong to one case. 1. Call `WriteFile()` with valid parameters, the return status should be `EFI_SUCCESS`.
5.26.4.8.10	0xc2e70601, 0xb8d5, 0x4aa6, 0xbb, 0x5c, 0x6e, 0xda, 0x2a, 0xd0, 0xa6, 0x6a	The `Token.Status` should be `EFI_SUCCESS`.	2. The `Token.Status` should be `EFI_SUCCESS`.
5.26.4.8.11	0x9e572894, 0x38da, 0x4039, 0x96, 0xc9, 0xaa, 0xfe, 0xa6, 0x48, 0x60, 0x74	`EFI_MTFTP6_PROTOCOL.WriteFile() – WriteFile()` returns `EFI_SUCCESS` with valid parameters.	5.26.4.8.11 to 5.26.4.8.13 belong to one case. 1. Call `WriteFile()` with valid parameters, the return status should be `EFI_SUCCESS`.
5.26.4.8.12	0xeba41d25, 0x03d7, 0x41d7, 0xa0, 0x58, 0xa8, 0x90, 0xad, 0x68, 0xa7, 0x0b	The `Token.Event` should be signaled.	2. The `Token.Event` should be signaled.

Number	GUID	Assertion	Test Description
5.26.4.8.13	0x39afac2f, 0xb620, 0x45e9, 0x8d, 0x82, 0x7a, 0xec, 0x36, 0x9d, 0x19, 0xfb	The `Token.Status` should be `EFI_SUCCESS`.	3. The `Token.Status` should be `EFI_SUCCESS`.
5.26.4.8.14	0x98410f1a, 0x6f26, 0x45f4, 0x8c, 0x5d, 0x9e, 0x11, 0x19, 0x53, 0xd3, 0xf8	`EFI_MTFTP6_PROTOCOL.WriteFile() - WriteFile()` returns `EFI_SUCCESS` with valid parameters, `Token.Event` is not `NULL` and current operation is aborted by user.	5.26.4.8.14 to 5.26.4.8.16 belong to one case. 1. Call `WriteFile()` with valid parameters, `Token.Event` is not `NULL` and current operation is aborted by user, the return status should be `EFI_SUCCESS`.
5.26.4.8.15	0xbb6d10b9, 0x4466, 0x4f97, 0x9a, 0x3f, 0xa9, 0xa0, 0x7a, 0x49, 0x88, 0x5d	The `Token.Event` should be signaled.	2. The `Token.Event` should be signaled.
5.26.4.8.16	0x87eff284, 0x80a4, 0x48ae, 0xa1, 0x87, 0x54, 0xa5, 0xf6, 0xd8, 0xcc, 0xcf	The `Token.Status` should be `EFI_ABORTED`.	3. The `Token.Status` should be `EFI_ABORTED`.
5.26.4.8.17	0x84cf72a7, 0x0d57, 0x4519, 0x8d, 0x94, 0x5a, 0x63, 0xeb, 0xff, 0x8c, 0x73	`EFI_MTFTP6_PROTOCOL.WriteFile() - WriteFile()` returns `EFI_SUCCESS` with valid parameters, `Token.Event` is not `NULL` and a mtftp6 error packet was received.	5.26.4.8.17 to 5.26.4.8.19 belong to one case. 1. Call `WriteFile()` with valid parameters, `Token.Event` is not `NULL` and a mtftp6 error packet was received, the return status should be `EFI_SUCCESS`.
5.26.4.8.18	0x35003a00, 0x715d, 0x4f05, 0x80, 0x0d, 0xec, 0xcc, 0x92, 0xed, 0x19, 0x51	The `Token.Event` should be signaled.	2. The `Token.Event` should be signaled.
5.26.4.8.19	0x11ce4fd8, 0x7d75, 0x49ec, 0x8a, 0x2b, 0x98, 0x57, 0xd9, 0xce, 0x5d, 0x66	The `Token.Status` should be `EFI_TFTP_ERROR`.	3. The `Token.Status` should be `EFI_TFTP_ERROR`.

Number	GUID	Assertion	Test Description
5.26.4.8.20	0xb1d3d500, 0x4afa, 0x465f, 0x8e, 0xac, 0x79, 0x0d, 0xf9, 0xef, 0x3f, 0xea	**EFI_MTFTP6_PROTOCOL .WriteFile() - WriteFile()** returns **EFI_SUCCESS** with valid parameters, **Token.Event** is not **NULL** and a icmp6 error packet was received.	5.26.4.8.20 to 5.26.4.8.22 belong to one case. 1. Call **WriteFile()** with valid parameters, **Token.Event** is not **NULL** and a icmp6 error packet was received, the return status should be **EFI_SUCCESS**.
5.26.4.8.21	0x42093ba6, 0x54ce, 0x408c, 0x82, 0x0f, 0xce, 0xba, 0xf4, 0x56, 0x58, 0xe9	The **Token.Event** should be signaled.	2. The **Token.Event** should be signaled.
5.26.4.8.22	0xc7ba7541, 0x6d62, 0x4143, 0x8e, 0x18, 0x6e, 0xbb, 0xfe, 0x2c, 0x42, 0xd8	The **Token.Status** should be **EFI_PORT_UNREACHABL E**.	3. The **Token.Status** should be **EFI_PORT_UNREACHABLE**.
5.26.4.8.23	0x08fffd13, 0x7cfb, 0x49ec, 0x8b, 0x02, 0x2a, 0x45, 0xb9, 0x78, 0x1d, 0xcd	**EFI_MTFTP6_PROTOCOL .WriteFile() - WriteFile()** returns **EFI_SUCCESS** with valid parameters, **Token.Event** is not **NULL** and a icmp6 error packet was received.	5.26.4.8.23 to 5.26.4.8.25 belong to one case. 1. Call **WriteFile()** with valid parameters, **Token.Event** is not **NULL** and no response was received, the return status should be **EFI_SUCCESS**.
5.26.4.8.24	0xeb45268b, 0xa856, 0x4ab8, 0xb4, 0xba, 0x38, 0xf9, 0x35, 0xd7, 0x99, 0x2c	The **Token.Event** should be signaled.	2. The **Token.Event** should be signaled.
5.26.4.8.25	0xa93f3a80, 0xeb22, 0x4ad8, 0xb5, 0x7c, 0xf5, 0x39, 0x7d, 0xe5, 0x39, 0x33	The **Token.Status** should be **EFI_TIMEOUT**.	3. The **Token.Status** should be **EFI_TIMEOUT**.

22.4.9 ReadDirectory()

Number	GUID	Assertion	Test Description
5.26.4.9.1	0x1947060b, 0x44a2, 0x4e22, 0x9b, 0xe9, 0x20, 0xf4, 0xa2, 0x9e, 0xb4, 0x2e	EFI_MTFTP6_PROTOCOL . ReadDirectory() - ReadDirectory() returns EFI_NOT_STARTED when the instance hasn't been configured.	Call ReadDirectory() when the instance hasn't been configured, the return status should be EFI_NOT_STARTED.
5.26.4.9.2	0x9d2a2470, 0x98de, 0x425a, 0xbb, 0x3f, 0xab, 0x01, 0x84, 0x13, 0xe6, 0x9c	EFI_MTFTP6_PROTOCOL . ReadDirectory() - ReadDirectory() returns EFI_INVALID_PARAMETER when Token is NULL.	Call ReadDirectory() when Token is NULL, the return status should be EFI_INVALID_PARAMETER.
5.26.4.9.3	0x6288a172, 0xbc68, 0x49c1, 0xa8, 0x85, 0x56, 0x5a, 0x9e, 0xf9, 0x5c, 0x46	EFI_MTFTP6_PROTOCOL . ReadDirectory() - ReadDirectory() returns EFI_INVALID_PARAMETER when Token.Filename is NULL.	Call ReadDirectory() when Token.Filename is NULL, the return status should be EFI_INVALID_PARAMETER.
5.26.4.9.4	0xe95a938b, 0x7d16, 0x4b40, 0xbd, 0x23, 0x81, 0x7b, 0xf6, 0xf1, 0xff, 0xb4	EFI_MTFTP6_PROTOCOL . ReadDirectory() - ReadDirectory() returns EFI_INVALID_PARAMETER when Token.OptionCount isn't zero and Token.OptionList is NULL.	Call ReadDirectory() when Token.OptionCount isn't zero and Token.OptionList is NULL, the return status should be EFI_INVALID_PARAMETER.
5.26.4.9.5	0x4ec4e899, 0x9f53, 0x461a, 0xb3, 0xe4, 0x21, 0xe4, 0x1e, 0x66, 0x56, 0x21	EFI_MTFTP6_PROTOCOL . ReadDirectory() - ReadDirectory() returns EFI_INVALID_PARAMETER when one or more options in Token.OptionList is wrong format.	Call ReadDirectory() when one or more options in Token.OptionList is wrong format, the return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.26.4.9.6	0xbb6ac976, 0xbe0b, 0x4329, 0x98, 0xe3, 0x1f, 0x2d, 0xc8, 0x63, 0x8f, 0x9a	EFI_MTFTP6_PROTOCOL. ReadDirectory() - ReadDirectory() returns EFI_INVALID_PARAMETER when one or more options in Token.Buffer and Token.CheckPacket are both NULL.	Call ReadDirectory() when one or more options in Token.Buffer and Token.CheckPacket are both NULL, the return status should be EFI_INVALID_PARAMETER.
5.26.4.9.7	0xa41fa6b3, 0xc128, 0x451f, 0x82, 0xf4, 0xf8, 0x92, 0x8e, 0xa8, 0x61, 0xfd	EFI_MTFTP6_PROTOCOL. ReadDirectory() - ReadDirectory() returns EFI_INVALID_PARAMETER when one or more options in Token.OverrideData. ServerIp is not valid unicast IPv6 address.	Call ReadDirectory() when one or more options in Token.OverrideData.ServerIp is not valid unicast IPv6 address, the return status should be EFI_INVALID_PARAMETER.
5.26.4.9.8	0xed4d5f77, 0x7856, 0x4c4a, 0x9a, 0x6b, 0x19, 0x66, 0xc7, 0xf6, 0x80, 0xfb	EFI_MTFTP6_PROTOCOL. ReadDirectory() - ReadDirectory() returns EFI_INVALID_PARAMETER when one or more options in Token.OptionList is not supported.	Call ReadDirectory() when one or more options in Token.OptionList is not supported, the return status should be EFI_UNSUPPORTED.
5.26.4.9.9	0xf6439066, 0xb46e, 0x484e, 0x9a, 0x99, 0xfe, 0xa9, 0xaf, 0x72, 0xf9, 0xce	EFI_MTFTP6_PROTOCOL. ReadDirectory() - ReadDirectory() returns EFI_ABORTED when current operation is aborted by user.	Call ReadDirectory() when current operation is aborted by user, the return status should be EFI_ABORTED.
5.26.4.9.10	0xd8f2b214, 0xbfc2, 0x4344, 0x85, 0x13, 0xfc, 0x07, 0x04, 0x3f, 0x63, 0x6f	EFI_MTFTP6_PROTOCOL. ReadDirectory() - ReadDirectory() returns EFI_TFTP_ERROR when a mtftp6 error packet was received.	Call ReadDirectory() when a mtftp6 error packet was received, the return status should be EFI_TFTP_ERROR.

Number	GUID	Assertion	Test Description
5.26.4.9.11	0x74cbaed3, 0x1521, 0x4677, 0x83, 0xb0, 0xca, 0xac, 0x84, 0x92, 0x27, 0x68	`EFI_MTFTP6_PROTOCOL.ReadDirectory() - ReadDirectory()` returns `EFI_PORT_UNREACHABLE` when a icmp6 port unreachable error packet was received.	Call `ReadDirectory()` when a icmp6 port unreachable error packet was received, the return status should be `EFI_PORT_UNREACHABLE`.
5.26.4.9.12	0x71038101, 0x41ba, 0x416e, 0xa7, 0xb3, 0xd6, 0x12, 0x27, 0x8f, 0xb9, 0xe5	`EFI_MTFTP6_PROTOCOL.ReadDirectory() - ReadDirectory()` returns `EFI_TIMEOUT` when no response was received.	Call `ReadDirectory()` when no response was received, the return status should be `EFI_TIMEOUT`.
5.26.4.9.13	0x177b35e7, 0x8e93, 0x48c4, 0x8f, 0x19, 0x7e, 0xe7, 0xf9, 0x9d, 0x65, 0x60	`EFI_MTFTP6_PROTOCOL.ReadDirectory() - ReadDirectory()` returns `EFI_SUCCESS` with valid parameters.	5.26.4.9.13 to 5.26.4.9.14 belong to one case. 1. Call `ReadDirectory()` with valid parameters, the return status should be `EFI_SUCCESS`.
5.26.4.9.14	0x63dd12dd, 0x62e4, 0x40d3, 0x88, 0x30, 0xac, 0x2e, 0x61, 0xd2, 0x08, 0xb8	The `Token.Status` should be `EFI_SUCCESS`.	2. The `Token.Status` should be `EFI_SUCCESS`.
5.26.4.9.15	0xbc2d0220, 0xa92b, 0x4281, 0x82, 0xf1, 0x45, 0xf1, 0x98, 0x10, 0x33, 0x05	`EFI_MTFTP6_PROTOCOL.ReadDirectory() - ReadDirectory()` returns `EFI_SUCCESS` with valid parameters.	5.26.4.9.15 to 5.26.4.9.17 belong to one case. 1. Call `ReadDirectory()` with valid parameters, the return status should be `EFI_SUCCESS`.
5.26.4.9.16	0xc495566f, 0x31a5, 0x47d3, 0x97, 0x50, 0xdf, 0xe9, 0xed, 0xfd, 0xe7, 0xfc	The `Token.Event` should be signaled.	2. The `Token.Event` should be signaled.
5.26.4.9.17	0xb734f8cc, 0x91c2, 0x4ce8, 0xa3, 0x11, 0x70, 0x70, 0xb8, 0x27, 0x6b, 0x03	The `Token.Status` should be `EFI_SUCCESS`.	3. The `Token.Status` should be `EFI_SUCCESS`.

Number	GUID	Assertion	Test Description
5.26.4.9.18	0xce7b5436, 0x3e80, 0x46d1, 0xbc, 0x6a, 0x04, 0x05, 0x91, 0xd8, 0xf2, 0x00	EFI_MTFTP6_PROTOCOL . ReadDirectory() - ReadDirectory() returns EFI_SUCCESS with valid parameters, Token.Event is not NULL and current operation is aborted by user.	5.26.4.9.18 to 5.26.4.9.20 belong to one case. 1. Call ReadDirectory() with valid parameters, Token.Event is not NULL and current operation is aborted by user, the return status should be EFI_SUCCESS.
5.26.4.9.19	0x06cc2106, 0x12e0, 0x4b26, 0x82, 0x84, 0xb6, 0x45, 0x26, 0x56, 0xb7, 0x67	The Token.Event should be signaled.	2. The Token.Event should be signaled.
5.26.4.9.20	0x2f8d4207, 0xcaa0, 0x4fe8, 0xae, 0x18, 0x09, 0xfc, 0xd8, 0x1f, 0x60, 0xb1	The Token.Status should be EFI_ABORTED.	3. The Token.Status should be EFI_ABORTED.
5.26.4.9.21	0xf19e2441, 0x2e9d, 0x4754, 0xaa, 0x1c, 0x9d, 0xff, 0x5d, 0xac, 0x7b, 0xfb	EFI_MTFTP6_PROTOCOL . ReadDirectory() - ReadDirectory() returns EFI_SUCCESS with valid parameters, Token.Event is not NULL and a mtftp6 error packet was received.	5.26.4.9.21 to 5.26.4.9.23 belong to one case. 1. Call ReadDirectory() with valid parameters, Token.Event is not NULL and a mtftp6 error packet was received, the return status should be EFI_SUCCESS.
5.26.4.9.22	0x6d29ada4, 0xb541, 0x4ed1, 0x9c, 0x54, 0x42, 0x97, 0xc6, 0x99, 0xa9, 0x2f	The Token.Event should be signaled.	2. The Token.Event should be signaled.
5.26.4.9.23	0x7ee4d2f0, 0x43a5, 0x4730, 0x88, 0x00, 0x7f, 0x72, 0xcd, 0x76, 0x6e, 0xb5	The Token.Status should be EFI_TFTP_ERROR.	3. The Token.Status should be EFI_TFTP_ERROR.

Number	GUID	Assertion	Test Description
5.26.4.9.24	0x24b159a5, 0x0d03, 0x408d, 0x84, 0x3d, 0x5b, 0xfd, 0xb8, 0xf7, 0x10, 0xc4	EFI_MTFTP6_PROTOCOL. ReadDirectory() - ReadDirectory() returns EFI_SUCCESS with valid parameters, Token.Event is not NULL and a icmp6 error packet was received.	5.26.4.9.24 to 5.26.4.9.26 belong to one case. 1. Call ReadDirectory() with valid parameters, Token.Event is not NULL and a icmp6 error packet was received, the return status should be EFI_SUCCESS.
5.26.4.9.25	0x450a81e4, 0xf424, 0x4399, 0x9a, 0xb9, 0xef, 0x2e, 0xa9, 0x4a, 0x7e, 0x46	The Token.Event should be signaled.	2. The Token.Event should be signaled.
5.26.4.9.26	0xc2b9002f, 0x9183, 0x4a4b, 0xba, 0x26, 0x3f, 0x72, 0x93, 0xed, 0xf0, 0xa4	The Token.Status should be EFI_PORT_UNREACHABLE.	3. The Token.Status should be EFI_PORT_UNREACHABLE.
5.26.4.9.27	0x1aef9df8, 0xcf77, 0x449a, 0xa1, 0x6b, 0x0b, 0x8d, 0x8e, 0x4a, 0xf9, 0x06	EFI_MTFTP6_PROTOCOL. ReadDirectory() - ReadDirectory() returns EFI_SUCCESS with valid parameters, Token.Event is not NULL and a icmp6 error packet was received.	5.26.4.9.27 to 5.26.4.9.29 belong to one case. 1. Call ReadDirectory() with valid parameters, Token.Event is not NULL and no response was received, the return status should be EFI_SUCCESS.
5.26.4.9.28	0xb5398e7d, 0x02cb, 0x4fd2, 0xa9, 0xdd, 0xd9, 0x75, 0x8f, 0x32, 0xd9, 0x26	The Token.Event should be signaled.	2. The Token.Event should be signaled.
5.26.4.9.29	0x279e1bfa, 0x5db9, 0x44ba, 0xbb, 0x3c, 0xe4, 0x3b, 0xf1, 0xeb, 0xa8, 0x70	The Token.Status should be EFI_TIMEOUT.	3. The Token.Status should be EFI_TIMEOUT.

22.4.10 Poll()

Number	GUID	Assertion	Test Description
5.26.4.10.1	0xdfb24a28, 0xc61c, 0x4ec0, 0x9e, 0x78, 0x3a, 0xcf, 0x85, 0x9f, 0xa8, 0x0e	**EFI_MTFTP6_PROTOCOL. Poll() – Poll()** returns **EFI_NOT_STARTED** when the instance hasn't been configured.	Call **Poll()** when the instance hasn't been configured, the return status should be **EFI_NOT_STARTED**.

23 Network Protocols VLAN and EAP

23.1 EFI_VLAN_CONFIG_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_VLAN_CONFIG_PROTOCOL Section.

23.1.1 Set()

Number	GUID	Assertion	Test Description
5.27.1.1.1	0xedbb5f4f, 0x4de7, 0x43ff, 0x82, 0x1c, 0x13, 0x80, 0x98, 0x95, 0xd1, 0x76	`EFI_VLAN_CONFIG_PROTOCOL.SET - SET()` returns `EFI_INVALID_PARAMETER` with an invalid `VlanId`.	Call `Set()` with valid parameters except an invalid `VlanId`, The return status should be `EFI_INVALID_PARAMETER`.
5.27.1.1.2	0x9c1292c2, 0xe03a, 0x438d, 0x9d, 0xab, 0x4e, 0xd0, 0xa9, 0xa8, 0xb6, 0x83	`EFI_VLAN_CONFIG_PROTOCOL.SET - SET()` returns `EFI_INVALID_PARAMETER` with an invalid `Priority`.	Call `Set()` with valid parameters except an invalid `Priority`, The return status should be `EFI_INVALID_PARAMETER`.
5.27.1.1.3	0xe3584990, 0x0b04, 0x48ea, 0x96, 0x3d, 0x36, 0xf7, 0x62, 0x29, 0x9f, 0x42	`EFI_VLAN_CONFIG_PROTOCOL.SET - SET()` returns `EFI_SUCCESS` with a valid `VlanId` and a valid `Priority`.	5.27.1.1.3 – 5.27.1.1.6 belong to one case 1. Call `Set()` with a valid `VlanId` and a valid `Priority`, The return status should be `EFI_SUCCESS`.
5.27.1.1.4	0xc14eb533, 0xc076, 0x4a9e, 0xb5, 0x6d, 0xea, 0x00, 0xce, 0xac, 0x7b, 0x2d	`EFI_VLAN_CONFIG_PROTOCOL.Find - Find()` returns `EFI_SUCCESS` with the same `VlanId`, a valid `NumberOfVlan` and a valid `Priority`.	2. Call `Find()` with the same `VlanId`, a valid `NumberOfVlan` and a valid `Priority`. The return status should be `EFI_SUCCESS`. The `NumberOfVlan` should be 1. The output `VlanId` and `Priority` in the `Entries` should be the same value of `VlanId`/`Priority` which are set in step1.
5.27.1.1.5	0x48deb1ad, 0xd59b, 0x404e, 0x88, 0xe7, 0x42, 0x53, 0xac, 0x0e, 0xce, 0x22	`EFI_VLAN_CONFIG_PROTOCOL.SET - SET()` returns `EFI_SUCCESS` with the same `VlanId` and a different `Priority`.	3. Call `Set()` with the same `VlanId` and a different `Priority`, The return status should be `EFI_SUCCESS`.

5.27.1.1.6	0x98f1580a, 0xa2b6, 0x4e61, 0x8b, 0xc9, 0x31, 0xb1, 0xac, 0xb0, 0x20, 0xb5	`EFI_VLAN_CONFIG_PROTOCOL.Find - Find()` returns `EFI_SUCCESS` with the same `VlanId`, a valid `NumberOfVlan` and a valid `Priority`.	4. Call `Find()` with the same `VlanId`, a valid `NumberOfVlan` and a valid `Priority`. The return status should be `EFI_SUCCESS`. The `NumberOfVlan` should be 1. The output `VlanId` and `Priority` in the `Entries` should be the same value of `VlanId`/`Priority` which are set in step2.

23.1.2 Find()

Number	GUID	Assertion	Test Description
5.27.1.2.1	0x07f07b52, 0x93e8, 0x43fe, 0xa7, 0x84, 0x01, 0x71, 0x02, 0x37, 0x66, 0x82	`EFI_VLAN_CONFIG_PROTOCOL.Find - Find()` returns `EFI_INVALID_PARAMETER` with an invalid `VlanId`.	Call `Find()` with valid parameters except an invalid `VlanId`, The return status should be `EFI_INVALID_PARAMETER`.
5.27.1.2.2	0xedb0b22d, 0xa6b5, 0x497d, 0xbb, 0x9b, 0x75, 0x85, 0x47, 0x11, 0x35, 0xc5	`EFI_VLAN_CONFIG_PROTOCOL.Find - Find()` returns `EFI_INVALID_PARAMETER` with `NumberOfVlan` been `NULL`.	Call `Find()` with valid parameters except `NumberOfVlan` being `NULL`, The return status should be `EFI_INVALID_PARAMETER`.
5.27.1.2.3	0x57d7d76b, 0x6b88, 0x44c9, 0x85, 0x0c, 0x0f, 0xb6, 0xcb, 0xe1, 0x9c, 0xd9	`EFI_VLAN_CONFIG_PROTOCOL.Find - Find()` returns `EFI_INVALID_PARAMETER` with `Entries` been `NULL`.	Call `Find()` with valid parameters(a valid `NumberOfVlan` and an NULL `VlanId`) except `Entries` being `NULL`, The return status should be `EFI_INVALID_PARAMETER`.
5.27.1.2.4	0x1de50bab, 0x1f3a, 0x4c62, 0x82, 0x93, 0x6e, 0x9a, 0x11, 0x03, 0xce, 0x6f	`EFI_VLAN_CONFIG_PROTOCOL.Find - Find()` returns `EFI_INVALID_PARAMETER` with `Entries` been `NULL`.	Call `Find()` with valid parameters(a valid `NumberOfVlan` and a valid `VlanId`) except `Entries` being `NULL`, The return status should be `EFI_INVALID_PARAMETER`.
5.27.1.2.5	0x49d1f535, 0x3b53, 0x4892, 0x98, 0x02, 0x5c, 0x19, 0xa3, 0x6d, 0xd1, 0x53	`EFI_VLAN_CONFIG_PROTOCOL.Find - Find()` returns `EFI_SUCCESS` or `EFI_NOT_FOUND` with valid parameters.	Call `Find()` with valid parameters(a valid `NumberOfVlan`, a valid Entries and NULL `VlanId`), The return status should be `EFI_SUCCESS` or `EFI_NOT_FOUND`. If `EFI_SUCCESS`. `NumberOfVlan` should be greater than 0 and `Entries` should not be `NULL`. If `EFI_NOT_FOUND`, `NumberOfVlan` should be 0 and `Entries` should be `NULL`.

Number	GUID	Assertion	Test Description
5.27.1.2.6	0x2f95fed6, 0xed1b, 0x4ac0, 0x9a, 0xa8, 0x97, 0x81, 0x76, 0xac, 0xcf, 0x8d	`EFI_VLAN_CONFIG_PROTOCOL.Find - Find()` returns `EFI_SUCCESS` with valid parameters.	5.27.1.2.6 – 5.27.1.2.7 belong to one case 1. Call `Set()` to config a Vlan 2. Call `Find()` with the same `VlanId` The return status should be `EFI_SUCCESS`. The `NumberOfVlan` should be 1. The output `VlanId` and `Priority` in the `Entries` should be the same value of `VlanId`/`Priority` which are set in step 1.
5.27.1.2.7	0xf4d6c7d9, 0x21bf, 0x48b5, 0xb0, 0x1d, 0x10, 0xf4, 0xfa, 0x11, 0xf7, 0x8d	`EFI_VLAN_CONFIG_PROTOCOL.Find - Find()` returns `EFI_NOT_FOUND` with valid parameters.	3. Call `Remove()` to delete the same VlanId 4. Call `Find()` with the same `VlanId` The return status should be `EFI_NOT_FOUND`. The `NumberOfVlan` should be 0. The `Entries` should be `NULL`.

23.1.3 Remove()

Number	GUID	Assertion	Test Description
5.27.1.3.1	0x1adaa7a4, 0xd1d3, 0x49d5, 0x97, 0xb4, 0xe4, 0x0f, 0x63, 0x1b, 0x68, 0xd0	`EFI_VLAN_CONFIG_PROTOCOL.Remove - Remove()` returns `EFI_INVALID_PARAMETER` with an invalid `VlanId`.	Call `Remove()` with valid parameters except an invalid `VlanId`, The return status should be `EFI_INVALID_PARAMETER`.
5.27.1.3.2	0xaa94b834, 0xf247, 0x4530, 0xb0, 0x6a, 0x49, 0x4e, 0x10, 0x37, 0xb5, 0xe5	`EFI_VLAN_CONFIG_PROTOCOL.Remove - Remove()` returns `EFI_NOT_FOUND` with an not set `VlanId`.	Call `Remove()` with valid parameters except an not set `VlanId`, The return status should be `EFI_NOT_FOUND`.
5.27.1.3.3	0x30991f39, 0x7410, 0x46ed, 0xa5, 0xe6, 0xdb, 0xc9, 0xf7, 0x86, 0x4f, 0xd3	`EFI_VLAN_CONFIG_PROTOCOL.Remove - Remove()` returns `EFI_SUCCESS` with valid parameters.	5.27.1.3.3 – 5.27.1.3.4 belong to one case 1. Call `Set()` to configure a `VlanId` 2. Call `Remove()` with the same `VlanId`. The return status should be `EFI_SUCCESS`.
5.27.1.3.4	0x28b96fd8, 0xc729, 0x4906, 0xa6, 0xbd, 0xda, 0xe4, 0xd0, 0x2a, 0x82, 0x1e	`EFI_VLAN_CONFIG_PROTOCOL.Remove - Remove()` returns `EFI_NOT_FOUND` with valid parameters.	3. Call `Remove()` with the same `VlanId`. The return status should be `EFI_NOT_FOUND`.

24 EFI Tape IO to Test

24.1 EFI_TAPE_IO_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_TAPE_IO_PROTOCOL Section.

Configuration

Before testing of `TapeRead()` and `TapeSapce()`, we must make tape ready by calling `TapeWrite()` and `TapeWriteFM()` to write some blocks and some FileMarks.

Required Elements

24.1.1 TapeRead()

Number	GUID	Assertion	Test Description
12.5.1.0.1	0xc42dcb51, 0x5101, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	check input parameters of testing `EFI_TAPE_IO_PROTOCOL.TapeRead().`	Check interface/environment valid.
12.5.1.1.1	0xc42dcb51, 0x5102, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeRead()` read some data from the tape.	Call `TapeRead()` with `(bufferSize=16384)` should return `EFI_SUCCESS`. Exit testing when error occurred. Please note the configuration.
12.5.1.1.2	0xc42dcb51, 0x512f, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	Verify the data getting from step **(12.5.1.1.1)**.	After success of step **(12.5.1.1.1)** check reading data is all correctly or not.
12.5.1.2.1	0xc42dcb51, 0x5103, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeRead()` Buffer invalid checking test.	Call `TapeRead()` with `(bufferSize!=0,Buffer=NULL)` should return `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
12.5.1.2.2	0xc42dcb51, 0x5104, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeRead()` `This=NULL` checking test.	Call `TapeRead()` with `(This=NULL)` should return `EFI_INVALID_PARAMETER`.
12.5.1.3.1	0xc42dcb51, 0x5105, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeRead()` parameters valid checking test.	Call `TapeRead()` with `(bufferSize=0,buffer=NULL)` should return `EFI_SUCCESS.`
12.5.1.3.2	0xc42dcb51, 0x5107, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeRead()` parameters valid checking test.	Call `TapeRead()` with `(bufferSize=0,buffer!=NULL)` should return `EFI_SUCCESS`.

24.1.2 TapeWrite()

Number	GUID	Assertion	Test Description
12.5.2.0.1	0xc42dcb51, 0x5108, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	check input parameters of testing `EFI_TAPE_IO_PROTOCOL.TapeWrite().`	Check interface/environment valid.
12.5.2.1.1	0xc42dcb51, 0x5109, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeWrite()` write some data to the tape.	Call `TapeWrite()` with `(bufferSize=16384)` should return `EFI_SUCCESS`. Exit testing when error occurred.
12.5.2.2.1	0xc42dcb51, 0x510a, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeWrite()` Buffer invalid checking test.	Call `TapeWrite()` with `(bufferSize!=0,Buffer=NULL)` should return `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
12.5.2.2.2	0xc42dcb51, 0x510b, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	EFI_TAPE_IO_PROTOCOL.TapeWrite() This invalid checking test.	Call TapeWrite() with (This =NULL) should return EFI_INVALID_PARAMETER.
12.5.2.3.1	0xc42dcb51, 0x510c, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	EFI_TAPE_IO_PROTOCOL.TapeWrite() parameters valid checking test A.	Call TapeWrite() with (bufferSize=0,buffer=NULL) should return EFI_SUCCESS.
12.5.2.3.2	0xc42dcb51, 0x510e, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	EFI_TAPE_IO_PROTOCOL.TapeWrite() parameters valid checking test B.	Call TapeWrite() with (bufferSize=0,buffer!=NULL) should return EFI_SUCCESS.

24.1.3 TapeRewind()

Number	GUID	Assertion	Test Description
12.5.3.0.1	0xc42dcb51, 0x5110, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	check input parameters of testing EFI_TAPE_IO_PROTOCOL.TapeRewind().	Check interface/environment valid.
12.5.3.1.1	0xc42dcb51, 0x5111, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	EFI_TAPE_IO_PROTOCOL.TapeRewind() rewind the tape.	Call TapeRewind() should return EFI_SUCCESS. Exit testing when error occurred.
12.5.3.2.1	0xc42dcb51, 0x5112, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	EFI_TAPE_IO_PROTOCOL.TapeRewind() parameters This invalid checking test.	Call TapeRewind(NULL) should return EFI_INVALID_PARAMETER.

24.1.4 TapeSpace()

Number	GUID	Assertion	Test Description
12.5.4.0.1	0xc42dcb51, 0x5118, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	check input parameters of testing `EFI_TAPE_IO_PROTOCOL.TapeSpace()`.	Check interface/environment valid.
12.5.4.1.0	0xc42dcb51, 0x511f, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeSpace()` make the tape testing ready.	Call `TapeRewind()` for tape ready. Exit testing when error occurred. Please note the configuration.
12.5.4.1.1	0xc42dcb51, 0x5119, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeSpace()` space some BLOCKs	Call `TapeSpace()` with `(spaceType = TAPE_SPACE_TYPE_BLOCK)` should return `EFI_SUCCESS`. Exit testing when error occurred.
12.5.4.1.2	0xc42dcb51, 0x511e, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeSpace()` space some FILEMARKs	Call `TapeSpace()` with `(spaceType = TAPE_SPACE_TYPE_FILEMARK)` should return `EFI_SUCCESS`. Exit testing when error occurred.
12.5.4.2.1	0xc42dcb51, 0x511a, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeSpace()` spaceDir < 0 testing	Call `TapeSpace()` with `(spaceDir < 0)` should return `EFI_SUCCESS`. Exit testing when error occurred.
12.5.4.3.1	0xc42dcb51, 0x511b, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeSpace()` spaceDir = 0 testing	Call `TapeSpace()` with `(spaceDir = 0)` should return `EFI_SUCCESS`. Exit testing when error occurred.

Number	GUID	Assertion	Test Description
12.5.4.4.1	0xc42dcb51, 0x511c, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeSpace()` spaceDir > 0 testing	Call `TapeSpace()` with `(spaceDir > 0)` should return `EFI_SUCCESS`. Exit testing when error occurred.
12.5.4.5.1	0xc42dcb51, 0x511d, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeSpace()` parameter spaceType invalid checking test.	Call `TapeSpace()` with `invalid spaceType` should return `EFI_INVALID_PARAMETER`.

24.1.5 TapeWriteFM()

Number	GUID	Assertion	Test Description
12.5.5.0.1	0xc42dcb51, 0x5120, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	check input parameters of testing `EFI_TAPE_IO_PROTOCOL.TapeWriteFM()`.	Check interface/environment valid.
12.5.5.1.1	0xc42dcb51, 0x5121, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeWriteFM()` write some FileMarks to the tape.	Call `TapeWriteFM()` with `(Count=1)` should return `EFI_SUCCESS`. Exit testing when error occurred.
12.5.5.2.1	0xc42dcb51, 0x5122, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeWriteFM()` parameter `This` invalid checking test.	Call `TapeWriteFM(NULL)` should return `EFI_INVALID_PARAMETER`.

24.1.6 TapeReset()

Number	GUID	Assertion	Test Description
12.5.6.0.1	0xc42dcb51, 0x5128, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	check input parameters of testing `EFI_TAPE_IO_PROTOCOL.TapeReset()`.	Check interface/environment valid.
12.5.6.1.1	0xc42dcb51, 0x5129, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeReset()` reset the tape.	Call `TapeReset()` with `(extendReset=TRUE)` should return `EFI_SUCCESS`.
12.5.6.1.2	0xc42dcb51, 0x512a, 0x4d36, 0xba,0x07, 0x9e,0xfc, 0x66,0xd1, 0x00,0xde	`EFI_TAPE_IO_PROTOCOL.TapeReset()` reset the tape.	Call `TapeReset()` with `(extendReset=FALSE)` should return `EFI_SUCCESS`.
12.5.6.2.1	0xc42dcb51, 0x5122, 0x4d36, 0xba, 0x07, 0x9e, 0xfc, 0x66, 0xd1, 0x00, 0xde	`EFI_TAPE_IO_PROTOCOL.TapeReset()` parameter `This` invalid checking test.	Call `TapeReset()` with invalid parameters `(This=NULL)` should return `EFI_INVALID_PARAMETER`.

25 Protocols Security Test

25.1 HASH Protocol Test

Reference Document:

UEFI Specification, EFI_HASH_PROTOCOL Section.

Configuration

- Call "`EFI_HASH_SERVICE_BINDING_PROTOCOL.CreateChild()`" before testing.
- Call "`EFI_HASH_SERVICE_BINDING_PROTOCOL.DestoryChild`" after testing.
- Execute testing of 25.4.1.1~25.4.1.3 and 25.4.2.1.1~25.4.2.5.2 for every hash protocol(SHA-x/MD5).

Required Elements

25.1.1 GetHashSize()

Number	GUID	Assertion	Test Description
25.4.1.0.1	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xae	check input parameters of testing `EFI_HASH_PROTOCOL.GetHashSize()`.	Check interface/environment valid.
25.4.1.1.1	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa2	`EFI_HASH_PROTOCOL.GetHashSize()` *HashSize* invalid checking test.	Call `GetHashSize()` with `(Hashsize=NULL)` should return `EFI_INVALID_PARAMETER`.
25.4.1.2.1	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa3	`EFI_HASH_PROTOCOL.GetHashSize()` HashAlgorithm invalid checking test A.	Call `GetHashSize()` with `(HashAlgorithm=NULL)` should return `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
25.4.1.2.2	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa4	`EFI_HASH_PROTOCOL.GetHashSize()` HashAlg invalid checking test B.	Call `GetHashSize()` with `(HashAlgorithm invalid)` should return `EFI_INVALID_PARAMETER`.
25.4.1.3.1	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa5	`EFI_HASH_PROTOCOL.GetHashSize()` get *`HashSize`* of the special HashAlgorithm.	Call `GetHashSize()` with `(HashAlgorithm =SHA-x/MD5)` should return `EFI_SUCCESS`. Exit testing when error occurred.

25.1.2 Hash()

Number	GUID	Assertion	Test Description
25.4.2.0.1	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xae	check input parameters of testing `EFI_HASH_PROTOCOL.Hash()`.	Check interface/environment valid.
25.4.2.1.1	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa6	`EFI_HASH_PROTOCOL.Hash()` Message invalid checking test.	Call `Hash()` with `(Message=NULL)` should return `EFI_INVALID_PARAMETER`.
25.4.2.1.2	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa7	`EFI_HASH_PROTOCOL.Hash()` Hash invalid checking test.	Call `Hash()` with `(Hash=NULL)` should return `EFI_INVALID_PARAMETER`.
25.4.2.2.1	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa8	`EFI_HASH_PROTOCOL.Hash()` HashAlgorithm invalid checking test.	Call `Hash()` with `(HashAlgorithm=NULL)` should return `EFI_INVALID_PARAMETER`.
25.4.2.2.2	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa9	`EFI_HASH_PROTOCOL.Hash()`HashAlgorithm invalid checking test.	Call `Hash()` with `invalid HashAlgorithm` should return `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
25.4.2.3.1	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xaa	`EFI_HASH_PROTOCOL.Hash()` Extend invalid checking test.	Call `Hash()` with `(HashAlgorithm=NULL and Extend=TRUE)` should return `EFI_INVALID_PARAMETER`.
25.4.2.4.1	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xab	`EFI_HASH_PROTOCOL.Hash()` hash some testing data.	Call `Hash()` with `(Extend=FALSE)` should return `EFI_SUCCESS`. Exit testing when error occurred.
25.4.2.4.2	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xac	Verify hash result getting from `EFI_HASH_PROTOCOL.Hash()` (25.4.2.4.1)	check hash result getting from `(25.4.2.4.1)` correct or not.
25.4.2.5.1	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xad	`EFI_HASH_PROTOCOL.Hash()` hash some extend testing data.	Call `Hash()` with `(Extend=TRUE)` should return `EFI_SUCCESS`. Exit testing when error occurred.
25.4.2.5.2	0xf2db2578, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xae	Verify hash result getting from `EFI_HASH_PROTOCOL.Hash()` (25.4.2.5.1)	check extend hash result getting from `(25.4.2.5.1)` correct or not.

25.2 AUTHENTICATION_INFO Protocol Test

Reference Document:

UEFI Specification, EFI_AUTHENTICATION_INFO_PROTOCOL Section.

Configuration

Required: prepare testing data by calling `EFI_AUTHENTICATION_INFO_PROTOCOL.Set()` before testing of `Get()`.

Required Elements

25.2.1 Get()

Number	GUID	Assertion	Test Description
25.1.1.1.1	0xf2db2579, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa3	**EFI_AUTHENTICATION_INFO_PROTOCOL.Get()** get authentication_info of the special *ControllerHandle*.	Call **Get()** with **(valid** *ControllerHandle***)** should return **EFI_SUCCESS**. Exit testing when error occurred.
25.1.1.2.1	0xf2db2579, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa4	**EFI_AUTHENTICATION_INFO_PROTOCOL.Get()** *ControllerHandle* invalid checking test.	Call **Get()** with **(***ControllerHandle***=NULL)** should return **EFI_INVALID_PARAMETER**.
25.1.1.3.1	0xf2db2579, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa5	**EFI_AUTHENTICATION_INFO_PROTOCOL.Get()** parameter *Buffer* invalid checking test.	Call **Get()** with **(***Buffer***=NULL)** should return **EFI_INVALID_PARAMETER**.

25.2.2 Set()

Number	GUID	Assertion	Test Description
25.1.2.1.1	0xf2db2579, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa6	**EFI_AUTHENTICATION_INFO_PROTOCOL.Set()** set authentication_info of the special *ControllerHandle*.	Call **Set()** with **(valid** *ControllerHandle* **and** *Buffer***)** should return **EFI_SUCCESS**. Exit testing when error occurred.
25.1.2.2.1	0xf2db2579, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa7	**EFI_AUTHENTICATION_INFO_PROTOCOL.Set()** *ControllerHandle* invalid checking test.	Call **Set()** with **(***ControllerHandle***=NULL)** should return **EFI_INVALID_PARAMETER**.
25.1.2.3.1	0xf2db2579, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa8	**EFI_AUTHENTICATION_INFO_PROTOCOL.Set()** parameter *Buffer* invalid checking test.	Call **Set()** with **(Buffer=NULL)** should return **EFI_INVALID_PARAMETER**.
25.1.2.4.1	0xf2db2579, 0xdc54, 0x4896, 0x83, 0x7f, 0x8d, 0xab, 0x41, 0xfb, 0xde, 0xa9	**EFI_AUTHENTICATION_INFO_PROTOCOL.Set()** parameter *length* invalid checking test.	Call **Set()** with **(GenericAuthenticationNodeStruct.length<18)** should return **EFI_INVALID_PARAMETER**.

25.3 EFI_HASH2_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_HASH2_PROTOCOL Section.

25.3.1 GetHashSize ()

Number	GUID	Assertion	Test Description
25.2.1.1.1	0xf70cb8e0, 0x2c12, 0x4976, 0xaf, 0xc9, 0xac, 0x90, 0xda, 0xae, 0x6e, 0x20	EFI_HASH2_PROTOCOL. GetHashSize() - GetHashSize() returns EFI_SUCCESS with valid parameters and HashSize match the HashAlgorithm.	1. Call GetHashSize() with the valid parameters, the return status should be EFI_SUCCESS and returned HashSize should match the HashAlgorithm.
25.2.1.1.2	0xb86858d8, 0xcb57, 0x4978, 0x9d, 0xed, 0xe7, 0xc7, 0xb1, 0x6, 0x75, 0xd7	EFI_HASH2_PROTOCOL. GetHashSize() - GetHashSize() returns EFI_UNSUPPORTED with unsupported HashAlgorithm or HashAlgorithm being NULL.	1. Call GetHashSize() with unsupported HashAlgorithm or HashAlgorithm being NULL, the return status should be EFI_UNSUPPORTED.
25.2.1.1.3	0x9a001932, 0x3abd, 0x4cca, 0x88, 0xb5, 0xdb, 0xa1, 0x58, 0xc5, 0xdb, 0xef	EFI_HASH2_PROTOCOL. GetHashSize() - GetHashSize() returns EFI_INVALID_PARAMETER when HashSize is NULL.	1. Call GetHashSize() when HashSize is NULL, the return status should be EFI_INVALID_PARAMETER.

25.3.2 Hash()

Number	GUID	Assertion	Test Description
25.2.1.2.1	0xf6905190, 0x3664, 0x4ff9, 0xac, 0x68, 0xce, 0x78, 0x24, 0x6b, 0x2a, 0x51	EFI_HASH2_PROTOCOL. Hash() - Hash() returns EFI_SUCCESS with valid parameters and Hash2Out should be correct.	1. Call GetHashSize() to get the supported HashAlgorithm. 2. Call Hash() with the supported HashAlgorithm. The return status should be EFI_SUCCESS. Hash ourput should be correct.
25.2.1.2.2	0x89690c0c, 0x63c1, 0x40ab, 0x9b, 0x91, 0xfe, 0xd2, 0x32, 0x1a, 0x3e, 0x99	EFI_HASH2_PROTOCOL. Hash() - Hash() returns EFI_UNSUPPORTED with unsupported HashAlgorithm or HashAlgorithm being NULL.	1. Call Hash() with unsupported HashAlgorithm or HashAlgorithm being NULL, the return status should be EFI_UNSUPPORTED.
25.2.1.2.3	0xb9cceaa1, 0x3b8f, 0x45e3, 0x8a, 0x27, 0x99, 0x45, 0x3e, 0xb4, 0xd1, 0xbb	EFI_HASH2_PROTOCOL. Hash() - Hash() returns EFI_INVALID_PARAMETER when Hash is NULL.	1. Call Hash() when Hash is NULL, the return status should be EFI_INVALID_PARAMETER.

25.3.3 HashInit()

Number	GUID	Assertion	Test Description
25.2.1.3.1	0x644e5fa7, 0x3d9b, 0x4a7b, 0xb1, 0x4e, 0x43, 0x34, 0x28, 0xf1, 0x60, 0xdb	EFI_HASH2_PROTOCOL. HashInit() - HashInit() returns EFI_UNSUPPORTED with unsupported HashAlgorithm or HashAlgorithm being NULL.	1. Call HashInit() with unsupported HashAlgorithm or HashAlgorithm being NULL, the return status should be EFI_UNSUPPORTED.
25.2.1.3.2	0x622e2357, 0xc5ff, 0x46b7, 0xab, 0xe7, 0xdb, 0x5e, 0x76, 0xbd, 0xca, 0xa9	EFI_HASH2_PROTOCOL. HashInit() - HashInit() returns EFI_ALREADY_STARTED when it follows the call to HashInit().	1. Call HashInit() when it follows the call to HashInit(), the return status should be EFI_ALREADY_STARTED.
25.2.1.3.3	0x69c8ed23, 0xf7fd, 0x4122, 0xb3, 0x1a, 0x46, 0xf8, 0x48, 0x11, 0xa5, 0x77	EFI_HASH2_PROTOCOL. HashInit() - HashInit() returns EFI_ALREADY_STARTED when it follows the call to HashUpdate().	1. Call HashInit() when it follows the call to HashUpdate(), the return status should be EFI_ALREADY_STARTED.

25.3.4 HashUpdate()

Number	GUID	Assertion	Test Description
25.2.1.4.1	0xa6a79ffd, 0x7e93, 0x4302, 0xb5, 0xaf, 0xe5, 0x43, 0xc5, 0x16, 0x35, 0x95	EFI_HASH2_PROTOCOL. HashUpdate() - HashUpdate() returns EFI_NOT_READY when it is not preceded by a call to HashInit().	1. Call HashUpdate() when it is not preceded by the call to HashInit(), the return status should be EFI_NOT_READY.
25.2.1.4.2	0x4021bf59, 0x8fab, 0x4a5e, 0xa8, 0x6b, 0x3e, 0xad, 0xa2, 0x78, 0xb3, 0x72	EFI_HASH2_PROTOCOL. HashUpdate() - HashUpdate() returns EFI_NOT_READY when it follows the call to Hash().	1. Call HashUpdate() when it follows the call to Hash(), the return status should be EFI_NOT_READY.
25.2.1.4.3	0xf7cd2a58, 0x18f9, 0x4285, 0xb9, 0x2b, 0x22, 0x76, 0x7e, 0xff, 0xc8, 0xf5	EFI_HASH2_PROTOCOL. HashUpdate() - HashUpdate() returns EFI_NOT_READY when it follows the call to HashFinal().	1. Call HashUpdate() when it follows the call to HashFinal(), the return status should be EFI_NOT_READY.

25.3.5 HashFinal()

Number	GUID	Assertion	Test Description
25.2.1.5.1	0xd66d9eb8, 0x52a9, 0x415d, 0xa9, 0x15, 0x7b, 0x50, 0xb8, 0x53, 0x34, 0x5a	EFI_Hash2_PROTOCOL.HashFinal() - HashFinal() returns EFI_SUCCESS with valid parameters.	1. Call GetHashSize() to get the supported HashAlgorithm. 2. Call HashInit() with the supported HashAlgorithm, the return status should be EFI_SUCCESS. 3. Call HashInit() with the supported HashAlgorithm, the return status should be EFI_ALREADY_STARTED. 4. Call HashUpdate() with the updated message, the return status should be EFI_SUCCESS. 5. Call HashUpdate() with the updated message, the return status should be EFI_SUCCESS. 6. Call HashFinal() to get the Hash output. The return status should be EFI_SUCCESS. Hash output should be correct.

25.2.1.5.2	0x459f2e7e, 0x1a98, 0x44c6, 0x97, 0xe, 0x38, 0x92, 0x67, 0xdb, 0xe1, 0x57	EFI_Hash2_PROTOCOL.HashFinal() - HashFinal() returns EFI_NOT_READY when it is not preceded by the call to HashInit()/HashUpdate().	1. Call HashFinal() when it is not preceded by the call to HashInit()/HashUpdate(), the return status should be EFI_NOT_READY.
25.2.1.5.3	0x57baa339, 0xab9b, 0x4cb7, 0x8e, 0xed, 0xeb, 0x97, 0x68, 0x82, 0xaf, 0x6b	EFI_Hash2_PROTOCOL.HashFinal() - HashFinal() returns EFI_NOT_READY when it is not preceded by the call to HashUpdate().	1. Call HashFinal() when it is not preceded by the call to HashUpdate(), the return status should be EFI_NOT_READY.
25.2.1.5.4	0x69af3be6, 0x3ac2, 0x467c, 0x8c, 0x41, 0x74, 0xd4, 0x53, 0x2f, 0x66, 0xa6	EFI_Hash2_PROTOCOL.HashFinal() - HashFinal() returns EFI_NOT_READY when it follows the call to Hash().	1. Call HashFinal() when it follows the call to Hash(), the return status should be EFI_NOT_READY.
25.2.1.5.5	0x6022b449, 0x9fe1, 0x4bd9, 0x84, 0x9c, 0x67, 0x9e, 0x7f, 0x7, 0xa5, 0xfe	EFI_Hash2_PROTOCOL.HashFinal() - HashFinal() returns EFI_INVALID_PARAMETER when Hash is NULL.	1. Call HashFinal() when Hash is NULL, the return status should be EFI_INVALID_PARAMETER.
25.2.1.5.6	0x2a6201e8, 0xe536, 0x4e92, 0xb6, 0x4e, 0x8e, 0xbd, 0xc6, 0xfe, 0xe0, 0x25	EFI_Hash2_PROTOCOL.HashFinal() - HashFinal() returns EFI_NOT_READY when it follows the call to HashFinal().	1. Call HashFinal() when it follows the call to HashFinal(), the return status should be EFI_NOT_READY.

25.4 EFI_PKCS7_VERIFY_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_PKCS7_VERIFY_PROTOCOL Section.

25.4.1 VerifyBuffer()

Number	GUID	Assertion	Test Description
25.3.1.1.1	0x5c0eec50, 0xa6ea, 0x413c, 0x8a, 0x46, 0x4a, 0xd1, 0x4a, 0x77, 0x76, 0xf1	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_SUCCESS when content signature was verified against hash of content, the signer's certificate was not found in RevokedDb, and was found in AllowedDb.	1. Call VerifyBuffer() when content signature was verified against hash of content, the signer's certificate was not found in RevokedDb, and was found in AllowedDb, the return status should be EFI_SUCCESS.
25.3.1.1.2	0x6ea61fbd, 0x1e46, 0x4854, 0x83, 0xf8, 0x22, 0x93, 0x24, 0x1a, 0x38, 0x67	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_BUFFER_TOO_SMALL when the size of buffer indicated by ContentSize is too small to hold the content. ContentSize should be updated to required size.	1. Call VerifyBuffer() when the size of buffer indicated by ContentSize is too small to hold the content, the return status should be EFI_BUFFER_TOO_SMALL. ContentSize should be updated to required size.
25.3.1.1.3	0x51af2845, 0x1bfe, 0x4bc3, 0x90, 0x69, 0x7b, 0x29, 0xbc, 0x7c, 0xc3, 0xc6	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_SUCCESS when the size of buffer indicated by ContentSize is big enough to hold the content, and retrive the correct content.	1. Call VerifyBuffer() when the size of buffer indicated by ContentSize is big enough to hold the content, and retrive the correct content, the return status should be EFI_SUCCESS.
25.3.1.1.4	0x912e23ef, 0x299c, 0x41ab, 0xa0, 0xf5, 0xfc, 0xbc, 0xf6, 0xfd, 0xd3, 0x32	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_SUCCESS when the content signature was verified against hash of content, signer is found in both AllowedDb and RevokedDb, the signing was allowed by reference to TimeStampDb.	1. Call VerifyBuffer() when the content signature was verified against hash of content, signer is found in both AllowedDb and RevokedDb, the signing was allowed by reference to TimeStampDb, the return status should be EFI_SUCCESS.

25.3.1.1.5	0x5ccc7dff, 0xc397, 0x4733, 0xb6, 0xc7, 0x88, 0xc4, 0x3e, 0x80, 0x6a, 0x67	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_UNSUPPORTED when SignedData is NULL or SignedDataSize is 0 or AllowedDb is NULL or Content is not NULL and ContentSize is NULL.	1. Call VerifyBuffer() when SignedData is NULL or SignedDataSize is 0 or AllowedDb is NULL or Content is not NULL and ContentSize is NULL, the return status should be EFI_INVALID_PARAMETER.
25.3.1.1.6	0xb1f546c3, 0x4e, 0x4e33, 0xb1, 0x81, 0x76, 0xf3, 0xf8, 0xb1, 0xd6, 0x5b	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_UNSUPPORTED when SignedData buffer is not correctly formatted for processing.	1. Call VerifyBuffer() when SignedData buffer is not correctly formatted for processing, the return status should be EFI_UNSUPPORTED.
25.3.1.1.7	0xf9382c57, 0xd51d, 0x4ba9, 0x91, 0x41, 0x30, 0xc6, 0x28, 0x8b, 0xd3, 0x64	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_ ABORTED when AllowedDb is invalid format.	1. Call VerifyBuffer() when AllowedDb is invalid format, the return status should be EFI_ ABORTED.
25.3.1.1.8	0x3b322e30, 0x8378, 0x441a, 0xba, 0x1d, 0xee, 0xe5, 0x53, 0xda, 0x21, 0x49	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_ABORTED when RevokedDb is invalid format.	1. Call VerifyBuffer() when RevokedDb is invalid format, the return status should be EFI_ ABORTED.
25.3.1.1.9	0xdfe02003, 0xb2ad, 0x46bc, 0xae, 0xe0, 0xf9, 0xb8, 0xd0, 0xec, 0xd3, 0x4a	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_ABORTED when TimeStampDb is invalid format.	1. Call VerifyBuffer() when TimeStampDb is invalid format, the return status should be EFI_ ABORTED.
25.3.1.1.10	0x8de626c4, 0x7112, 0x4a57, 0xb2, 0xbb, 0x30, 0xc, 0x5f, 0x2a, 0xc1, 0x8e	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_SECURITY_VIOLATION when Buffer is correctly formatted but signer is not in AllowedDb.	1. Call VerifyBuffer() when Buffer is correctly formatted but signer is not in AllowedDb, the return status should be EFI_SECURITY_VIOLATION.

25.3.1.1.11	0x399e1246, 0xd15a, 0x491a, 0xbb, 0x82, 0x99, 0xa4, 0xda, 0xb3, 0xac, 0x28	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_SECURITY_VIOLATION when Buffer is correctly formatted but signer is in RevokedDb.	1. Call VerifyBuffer() when Buffer is correctly formatted but signer is in RevokedDb, the return status should be EFI_SECURITY_VIOLATION.
25.3.1.1.12	0x670b4eab, 0xf28d, 0x42db, 0xa7, 0xbc, 0xad, 0xd, 0x59, 0x80, 0x49, 0xaf	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_SECURITY_VIOLATION when Buffer is correctly formatted but the content hash is in RevokedDb.	1. Call VerifyBuffer() when Buffer is correctly formatted but the content hash is in RevokedDb, the return status should be EFI_SECURITY_VIOLATION.
25.3.1.1.13	0xfd98e4e5, 0xf8af, 0x4dcf, 0x81, 0x1a, 0x6c, 0xf4, 0x99, 0x8a, 0x3, 0x9d	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_UNSUPPORTED when Signed data embedded in SignedData but InData is not NULL.	1. Call VerifyBuffer() when Signed data embedded in SignedData but InData is not NULL, the return status should be EFI_UNSUPPORTED.
25.3.1.1.14	0xb136e016, 0x4f80, 0x44bd, 0xba, 0xb0, 0x1c, 0x34, 0x8a, 0x2d, 0xa1, 0x8a	EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer() - VerifyBuffer() returns EFI_ NOT_FOUND when InData is NULL and no content embedded in SignedData.	1. Call VerifyBuffer() when InData is NULL and no content embedded in SignedData, the return status should be EFI_NOT_FOUND.

26 Protocols
EFI Firmware Management Test Case

26.1 EFI_FIRMWARE_MANAGEMENT_PROTOCOL

Reference Document:

UEFI 2.3 Specification, Chapter 32.

26.1.1 GetImageInfo()

Number	GUID	Assertion	Test Description
32.1.1.1.1	0xd02b40ae, 0x62f, 0x4155, 0xbb, 0xdd, 0x4, 0x29, 0x18, 0x94, 0xea, 0x31	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.GetImageInfo()** returns EFI_SUCCESS	Call function with all valid parameters. The function should return **EFI_SUCCESS**. Check for expected return code. Check **DescriptorVersion* is equal to 1. Check *ImageIndex* is between 1 and **DescriptorCount*. Check *AttributesSupported* has no bits set beyond bit 3. Check *AttributesSetting* has no bits set beyond bit 3. Check Compatibilities bits 1 thru 15 are 0s.
32.1.2.1.1	0x3789b80e, 0xab70, 0x4dc9, 0xbb, 0xbd, 0x70, 0x63, 0x76, 0x36, 0xab, 0x52	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.GetImageInfo()** returns EFI_BUFFER_TOO_SMALL	Call function with valid parameters, except **ImageInfoSize* = 1. The function should return **EFI_BUFFER_TOO_SMALL** and **ImageInfoSize* > 1.
32.1.2.1.2	0xca1d7706, 0x256b, 0x464e, 0xb6, 0xee, 0x50, 0x34, 0x1e, 0xec, 0x3c, 0x83	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.GetImageInfo()** returns EFI_INVALID_PARAMETER	Call function with valid parameters, except *ImageInfoSize* is NULL. The function should return **EFI_INVALID_PARAMETER**.

26.1.2 GetImage()

Number	GUID	Assertion	Test Description
32.2.1.1.1	0xff704c46, 0x3999, 0x4a28, 0xa3, 0x6e, 0x76, 0x8a, 0xb6, 0xad, 0x89, 0xd8	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.GetImage()**returns **EFI_SUCCESS** or **EFI_UNSUPPORTED**	Authentication not required. Call function with all valid parameters. The function should return **EFI_SUCCESS** or **EFI_UNSUPPORTED**.
32.2.2.1.1	0x3c8d87b2, 0x6a89, 0x4a6c, 0xbc, 0x75, 0xe6, 0x86, 0xa1, 0x49, 0x13, 0xf0	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.GetImage()**returns **EFI_BUFFER_TOO_SMALL**	Function is supported. Authentication not required. Call function with valid parameters, except **ImageSize* = 1. The function, if supported, should return **EFI_BUFFER_TOO_SMALL** and **ImageSize* > 1.
32.2.2.1.2	0x88031c96, 0x99bf, 0x4d2c, 0x9f, 0x57, 0xa7, 0x2, 0x6a, 0xbc, 0xd3, 0x51	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.GetImage()**returns **EFI_INVALID_PARAMETER**	Function is supported. Authentication not required. Call function with valid parameters, except *Image* is NULL. The function should return **EFI_INVALID_PARAMETER**.
32.2.2.1.3	0x7a386361, 0x3a5d, 0x4e58, 0x8a, 0x51, 0x4d, 0x93, 0xb6, 0x55, 0x95, 0xf4	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.GetImage()**returns **EFI_INVALID_PARAMETER** or **EFI_NOT_FOUND**	Function is supported. Authentication not required. Call function with valid parameters, except *ImageIndex* = 0 or *ImageIndex* = **DescriptorCount* + 1. The function should return **EFI_INVALID_PARAMETER** or **EFI_NOT_FOUND**.
32.2.2.1.4	0xd6a77629, 0x5afd, 0x4854, 0x87, 0xc8, 0xee, 0x9f, 0xc5, 0x3d, 0xbe, 0x3d	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.GetImage()**returns **EFI_SECURITY_VIOLATION**	Function is supported. Authentication required. Call function with valid parameters, except *Image* has a dummy authentication data. The function should return **EFI_SECURITY_VIOLATION**.

26.1.3 SetImage()

Number	GUID	Assertion	Test Description
32.3.2.1.1	0x4ea24764, 0xa6b1, 0x43b5, 0xb8, 0xa0, 0xd3, 0x3f, 0xdc, 0x8b, 0xc6, 0xe4	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.SetImage()**returns **EFI_INVALID_PARAMETER**	Function is supported. Authentication not required. Call function with valid parameters, except *Image* is NULL. The function should return **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
32.3.2.1.2	0xc82d1373, 0x1f87, 0x45f4, 0xaf, 0xfc, 0x10, 0xa7, 0xf7, 0xb0, 0x9c, 0xb0	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.SetImage()**returns **EFI_INVALID_PARAMETER** or **EFI_ABORTED**	Function is supported. Authentication not required. Call function with valid parameters, except *ImageIndex* = 0 or *ImageIndex* = **DescriptorCount* + 1. The function should return **EFI_INVALID_PARAMETER** or **EFI_ABORTED**.
32.3.2.1.3	0x2410a859, 0xdf6f, 0x4857, 0x92, 0x4a, 0x26, 0x37, 0x7, 0x11, 0xf, 0x1c	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.SetImage()**returns **EFI_SECURITY_VIOLATION**	Function is supported. Authentication not required. Call function with valid parameters, except *Image* has dummy authentication data. The function should return **EFI_SECURITY_VIOLATION**.

26.1.4 CheckImage()

Number	GUID	Assertion	Test Description
32.4.2.1.1	0x3987172c, 0xe6a0, 0x4099, 0xb1, 0x2b, 0xd8, 0xef, 0xf2, 0x62, 0x75, 0x93	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.CheckImage()** returns EFI_INVALID_PARAMETER	Function is supported. Authentication not required. Call function with valid parameters, except *Image* is NULL. The function should return **EFI_INVALID_PARAMETER**.
32.4.2.1.2	0xd6dad28e, 0x7f0f, 0x4f56, 0x9a, 0x93, 0x14, 0x7d, 0xb3, 0x74, 0x0, 0xc9	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.CheckImage()** returns EFI_SECURITY_VIOLATION	Function is supported. Authentication required. Call function with valid parameters, except *Image* has a dummy authentication data. The function should return **EFI_SECURITY_VIOLATION**.

26.1.5 GetPackageInfo()

Number	GUID	Assertion	Test Description
32.5.1.1.1	0x70884539, 0x9a34, 0x4146, 0x83, 0x3a, 0x4d, 0x89, 0x8b, 0x9c, 0x7e, 0xa4	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.GetPackageInfo()** returns **EFI_SUCCESS** or **EFI_UNSUPPORTED**	Call function with all valid parameters. The function should return **EFI_SUCCESS** or **EFI_UNSUPPORTED**. Check **AttributesSupported* has no bits set beyond bit 2. Check **AttributesSetting* has no bits set beyond bit 2.

26.1.6 SetPackageInfo()

Number	GUID	Assertion	Test Description
32.6.2.1.1	0xb5288fc3, 0xe906, 0x4468, 0x83, 0x3d, 0xd4, 0xa6, 0x58, 0xa5, 0x4f, 0xbd	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.SetPackageInfo()** returns EFI_INVALID_PARAMETER	Function is supported. Authentication not required. Call function with valid parameters, except ***PackageVersionName* is longer than the value returned in **PackageVersionNameMaxLen*. The function should return **EFI_INVALID_PARAMETER**.
32.6.2.1.2	0x57355301, 0x1343, 0x497f, 0xbe, 0xe0, 0x8e, 0x5c, 0x27, 0xd2, 0x40, 0x2	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.SetPackageInfo()** returns EFI_SECURITY_VIOLATION	Function is supported. Authentication is required. Call function with valid parameters, except *Image* is NULL. The function should return **EFI_SECURITY_VIOLATION**.
32.6.2.1.3	0xadeab82d, 0x7592, 0x40fe, 0x87, 0xa8, 0x93, 0x2b, 0xad, 0x97, 0xff, 0x5e	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.SetPackageInfo()** returns EFI_SECURITY_VIOLATION	Function is supported. Authentication is required. Call function with valid parameters, except *ImageSize* is 0. The function should return **EFI_SECURITY_VIOLATION**.
32.6.2.1.4	0x9be658d2, 0x1312, 0x4254, 0x91, 0x10, 0x59, 0x0, 0xd5, 0xfd, 0x6c, 0x6c	**EFI_FIRMWARE_MANAGEMENT_PROTOCOL.SetPackageInfo()** returns EFI_SECURITY_VIOLATION	Function is supported. Authentication is required. Call function with valid parameters, except *Image* has a dummy authentication data. The function should return **EFI_SECURITY_VIOLATION**.

27 Protocols HII Test

27.1 EFI_HII_FONT_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_HII_FONT_PROTOCOL Section.

27.1.1 StringToImage()

Number	GUID	Assertion	Test Description
5.18.1.1.1	0x6fca8706, 0x7d83, 0x4914, 0x8a, 0x16, 0x92, 0x0b, 0x07, 0xb1, 0x68, 0xb9	`HII_FONT_PROTOCOL.StringToImage - StringToImage()` returns `EFI_INVALID_PARAMETER` with `String` been `NULL`.	Call `StringToImage()` with valid parameters except `String` being `NULL`, The return status should be `EFI_INVALID_PARAMETER`.
5.18.1.1.2	0x80ee2790, 0x9ff7, 0x4abe, 0x90, 0xaf, 0x05, 0x4a, 0x86, 0x69, 0xba, 0x51	`HII_FONT_PROTOCOL.StringToImage - StringToImage()` returns `EFI_INVALID_PARAMETER` with `Blt` been `NULL.`	Call `StringToImage()` with valid parameters except `Blt` being `NULL`, The return status should be `EFI_INVALID_PARAMETER`.
5.18.1.1.3	0xe2f66ec3, 0x585a, 0x45ba, 0x8f, 0x7a, 0xd5, 0x18, 0x5f, 0xeb, 0x4e, 0x9a	`HII_FONT_PROTOCOL.StringToImage - StringToImage()` returns `EFI_INVALID_PARAMETER` with wrong flag combination.	Call `StringToImage()` with `Flag` being `EFI_HII_OUT_FLAG_CLEAN_X` with `EFI_HII_OUT_FLAG_WRAP`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.1.1.4	0xabf68512, 0x0bb8, 0x4ef8, 0x97, 0xc1, 0xda, 0x93, 0x55,0xda, 0x1b, 0x07	`HII_FONT_PROTOCOL.StringToImage - StringToImage()` returns `EFI_INVALID_PARAMETER` with wrong flag combination.	Call `StringToImage()` with `Flag` being `EFI_HII_OUT_FLAG_CLEAN_X` without `EFI_HII_OUT_FLAG_CLIP`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.1.1.5	0x6ff9c8b4, 0xeb8f, 0x4e0b, 0x9a, 0x97, 0x82, 0x94, 0x37, 0x0c, 0xdd, 0x3c	`HII_FONT_PROTOCOL.StringToImage - StringToImage()` returns `EFI_SUCCESS` with valid parameters.	Call `StringToImage()` with valid paramenters and use `EFI_GRAPHICS_OUTPUT_BLT_PIXEL` structure in `EFI_IMAGE_OUTPUT` structure.

Number	GUID	Assertion	Test Description
5.18.1.1.6	0x182cc281, 0xb462, 0x458f, 0xaa, 0xb6, 0xca, 0x98, 0xb5, 0x27, 0x37, 0x31	**HII_FONT_PROTOCOL. StringToImage - StringToImage()** returns **EFI_SUCCESS** with valid parameters.	Call **StringToImage()** with valid paramenters and use **EFI_GRAPHICS_OUTPUT_PROTOCOL** in **EFI_IMAGE_OUTPUT** structure.
5.18.1.1.7	0xcdf439d0, 0xe471, 0x4fe7, 0x86, 0x98, 0xf5, 0xb0, 0x5c, 0xcd, 0xa6, 0xae	**HII_FONT_PROTOCOL. StringToImage - StringToImage()** returns **EFI_SUCCESS** with valid parameters for all ASCII visible characters. Each images must equal to sys default glyph.	Call **StringToImage()** with valid paramenters and **StringInfo** = NULL.. Compare image output with system default font glyph image
5.18.1.1.8	0xa8f40eac, 0x8633, 0x40ca, 0x95, 0x6d, 0x75, 0xb2, 0x81, 0x50, 0x75, 0x39	**HII_FONT_PROTOCOL. StringToImage - StringToImage()** returns **EFI_SUCCESS** with valid parameters for all ASCII visible characters. Each image must equal to the specific font glyph.	Register a specific font packageCall **StringToImage()** with valid paramenters and **StringInfo** = specific font. Compare image output with specific font glyph image registered
5.18.1.1.9	0x42dc1626, 0x36ce, 0x421b, 0x8d, 0x66, 0x21, 0xb8, 0xaa, 0x43, 0x6c, 0x7b	**HII_FONT_PROTOCOL. StringToImage - StringToImage()** returns **EFI_SUCCESS** with parameters **EFI_HII_DIRECT_TO_SCREEN**	1.Call **StringToImage**() with **EFI_HII_DIRECT_TO_SCREEN**. For the final row, the **RowInfoArray.LineHeight** and **RowInfoArray.BaseLine** may describe pixels which are outside the limit specified by Blt. Height (unless **EFI_HII_OUT_FLAG_CLIP_CLEAN_Y** is specified) even though those pixels were not drawn. 2.The return code should be **EFI_SUCCESS** .

Number	GUID	Assertion	Test Description
5.18.1.1.10	0xf8b5b9b6, 0xc3c6, 0x4993, 0x9b, 0x3c, 0xbc, 0x8d, 0x91, 0xee, 0x8c, 0x20	HII_FONT_PROTOCOL. StringToImage - StringToImage() returns EFI_SUCCESS with parameter EFI_HII_OUT_FLAG_CLIP | EFI_HII_DIRECT_TO_SCREEN	1.Call StringToImage with EFI_HII_OUT_FLAG_CLIP | EFI_HII_DIRECT_TO_SCREEN. For the final row, the RowInfoArray.LineHeight and RowInfoArray.BaseLine 'May' describe pixels which are outside the limit specified by Blt. Height (unless EFI_HII_OUT_FLAG_CLIP_CLEAN_Y is specified) even though those pixels were not drawn. 2.The return code should be EFI_SUCCESS .
5.18.1.1.11	0x4c70adb5, 0xcc05, 0x435a, 0x8c, 0xc4, 0xce, 0xd1, 0x54, 0x6e, 0xd7, 0xf6	HII_FONT_PROTOCOL. StringToImage - StringToImage() returns EFI_SUCCESS with parameter EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_X | EFI_HII_DIRECT_TO_SCREEN	1.Call StringToImage() with EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_X | EFI_HII_DIRECT_TO_SCREEN. If a character's right-most on pixel cannot fit, then it will not be drawn at all. 2.The return code should be EFI_SUCCESS .
5.18.1.1.12	0xa000d36f, 0x2918, 0x448c, 0xad, 0x6d, 0x15, 0x77, 0xb5, 0x2f, 0xdc, 0x66	HII_FONT_PROTOCOL. StringToImage - StringToImage() returns EFI_SUCCESS with parameter EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_Y | EFI_HII_DIRECT_TO_SCREEN	1.Call StringToImage() with EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_Y | EFI_HII_DIRECT_TO_SCREEN. If a row's bottom-most pixel exceed screen *Height*, then it will not be drawn at all. 2.The return code should be EFI_SUCCESS .
5.18.1.1.13	0x266f881, 0x409b, 0x47e5, 0x8f, 0x22, 0x21, 0x7d, 0x14, 0xa4, 0x8a, 0xab	HII_FONT_PROTOCOL. StringToImage - StringToImage() returns EFI_SUCCESS with parameter EFI_HII_IGNORE_IF_NO_GLYPH | EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and *String* with line break opportunity	1.Call StringToImage() with EFI_HII_IGNORE_IF_NO_GLYPH | EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and *String* with line break opportunity (SPACE is a line break opportunity). Check display will wrapper at right place. 2.The return code should be EFI_SUCCESS .

Number	GUID	Assertion	Test Description
5.18.1.1.14	0x2fa4edd2, 0xa193, 0x4882, 0xae, 0x1e, 0xeb, 0xfe, 0xf5, 0x57, 0x42, 0xcc	**HII_FONT_PROTOCOL. StringToImage - StringToImage()** returns **EFI_SUCCESS** with parameter **EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN** and *String* without line break opportunity	1.Call **StringToImage()** with **EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN** and *String* without line break opportunity. *String* is designed to display as if **EFI_HII_OUT_FLAG_CLIP_CLEAN_X** is set. 2.The return code should be **EFI_SUCCESS** .
5.18.1.1.15	0x57300788, 0xba79, 0x4727, 0xb5, 0xe6, 0xe9, 0x20, 0xcd, 0x7e, 0xd6, 0x93	**HII_FONT_PROTOCOL. StringToImage - StringToImage()** returns **EFI_SUCCESS** with parameter **EFI_HII_IGNORE_LINE_BREAK | EFI_HII_DIRECT_TO_SCREEN**	1.Call **StringToImage()** with **EFI_HII_IGNORE_LINE_BREAK | EFI_HII_DIRECT_TO_SCREEN.** If a row's bottom-most pixel cannot fit, then it will not be drawn at all. This flag requires that **EFI_HII_OUT_FLAG_CLIP**beset. 2.The return code should be **EFI_SUCCESS** .
5.18.1.1.16	0xf3b0daef, 0xab51, 0x4ebc, 0x93, 0x51, 0x74, 0xf6, 0x18, 0xaa, 0x9f, 0x9f	**HII_FONT_PROTOCOL. StringToImage - StringToImage()** returns **EFI_SUCCESS** with parameter **EFI_HII_DIRECT_TO_SCREEN**	1.Register a new font package 2.Call **StringToImage()** with **EFI_HII_DIRECT_TO_SCREEN**. 3.Check **EFI_HII_DIRECT_TO_SCREEN** only case If **Blt** is not NULL, then **EFI_HII_OUT_FLAG_CLIP** is implied *String* is designed to displayed with one full line 4.The return code should be **EFI_SUCCESS** .
5.18.1.1.17	0x23ab3935, 0x483c, 0x4d75, 0xab, 0x3, 0xef, 0x50, 0x32, 0xea, 0x30, 0xbf	**HII_FONT_PROTOCOL. StringToImage - StringToImage()** returns **EFI_SUCCESS** with parameter **EFI_HII_OUT_FLAG_CLIP**	1.Register a new font package 2.Call **StringToImage()** with **EFI_HII_OUT_FLAG_CLIP**. 3. For the final row, the **RowInfoArray.LineHeight** and **RowInfoArray.BaseLine** may describe pixels which are outside the limit specified by **Blt.Height** (unless **EFI_HII_OUT_FLAG_CLIP_CLEAN_Y** is specified) even though those pixels were not drawn. 4.The return code should be **EFI_SUCCESS** .

Number	GUID	Assertion	Test Description
5.18.1.1.18	0x9e992f5a, 0x4a3b, 0x44d8, 0x89, 0x47, 0xca, 0x30, 0x92, 0x2b, 0x69, 0xa5	HII_FONT_PROTOCOL.StringToImage - StringToImage() returns EFI_SUCCESS with parameter EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_X | EFI_HII_DIRECT_TO_SCREEN	1.Register a new font package 2.Call StringToImage() with EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_X|EFI_HII_DIRECT_TO_SCREEN. 3. If a character's right-most on pixel cannot fit, then it will not be drawn at all. 4.The return code should be EFI_SUCCESS .
5.18.1.1.19	0xc8999c53, 0xd56, 0x4545, 0xbc, 0x55, 0x91, 0xf0, 0xd1, 0x1, 0x60, 0x4a	HII_FONT_PROTOCOL.StringToImage - StringToImage() returns EFI_SUCCESS with parameter EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_Y | EFI_HII_DIRECT_TO_SCREEN	1.Register a new font package 2.Call StringToImage() with EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_Y|EFI_HII_DIRECT_TO_SCREEN. 3.If a row's bottom-most pixel exceed screen Height, then it will not be drawn at all. 4.The return code should be EFI_SUCCESS .
5.18.1.1.20	0x9b71db4d, 0x5a06, 0x4246, 0x83, 0xd2, 0x9d, 0x31, 0x70, 0x73, 0x63, 0xd0	HII_FONT_PROTOCOL.StringToImage - StringToImage() returns EFI_SUCCESS with parameter EFI_HII_IGNORE_IF_NO_GLYPH | EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and *String* with line break opportunity	1.Register a new font package 2.Call StringToImage() with EFI_HII_IGNORE_IF_NO_GLYPH | EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and *String* with line break opportunity (SPACE is a line-break). 3.Check if the display is right. 4.The return code should be EFI_SUCCESS .
5.18.1.1.21	0xb0e526b1, 0xc399, 0x4e31, 0xb2, 0x97, 0xc1, 0x29, 0x18, 0x37, 0x95, 0x79	HII_FONT_PROTOCOL.StringToImage - StringToImage() returns EFI_SUCCESS with parameter EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and *String* without line break opportunity	1.Register a new font package 2.Call StringToImage() with EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and *String* without line break opportunity. 3. *String* is designed to display as if EFI_HII_OUT_FLAG_CLIP_CLEAN_X is set. 4.The return code should be EFI_SUCCESS .

Number	GUID	Assertion	Test Description
5.18.1.1.22	0xcbdae1b4, 0xc99b, 0x4a08, 0x9b, 0xf9, 0x76, 0x69, 0x77, 0x71, 0x66, 0x30	HII_FONT_PROTOCOL.StringToImage - StringToImage() returns EFI_SUCCESS with parameter EFI_HII_IGNORE_LINE_BREAK | EFI_HII_DIRECT_TO_SCREEN	1.Register a new font package 2.Call StringToImage() with EFI_HII_IGNORE_LINE_BREAK | EFI_HII_DIRECT_TO_SCREEN. 3. If a row's bottom-most pixel cannot fit, then it will not be drawn at all. This flag requires that EFI_HII_OUT_FLAG_CLIP isset. 4. The return code should be EFI_SUCCESS .
5.18.1.1.23	0x36a9a186, 0x363f, 0x4b4b, 0xa3, 0xaf, 0xa9, 0x9b, 0x29, 0x7a, 0x6d, 0x41	HII_FONT_PROTOCOL.StringToImage - StringToImage() returns EFI_SUCCESS with parameter EFI_HII_OUT_FLAG_TRANSPARENT	1.Register a new font package 2.Call StringToImage() with EFI_HII_OUT_FLAG_TRANSPARENT. 3. Check output buffer StringInfo background should be ignored according to EFI spec. 4. The return code should be EFI_SUCCESS .

27.1.2 StringIdToImage()

Number	GUID	Assertion	Test Description
5.18.1.2.1	0xf4e2c51e, 0x92a3, 0x4752, 0x92, 0x64, 0x27, 0xb1, 0x54, 0x21, 0x70, 0x3a	`HII_FONT_PROTOCOL.StringIdToImage - StringIdToImage()` returns `EFI_INVALID_PARAMETER` with `Blt` been `NULL`.	Call `StringIdToImage()` with valid parameters except `Blt` being `NULL`, The return status should be `EFI_INVALID_PARAMETER`.
5.18.1.2.2	0x9aecc9b3, 0x3bff, 0x4c7c, 0x96, 0x6b, 0xa9, 0x64, 0x84, 0xfe, 0xd9, 0x89	`HII_FONT_PROTOCOL.StringIdToImage - StringIdToImage()` returns `EFI_INVALID_PARAMETER` with `PackageList` been `NULL`.	Call `StringIdToImage()` with valid parameters except `PackageList` being `NULL`, The return status should be `EFI_INVALID_PARAMETER`.
5.18.1.2.3	0x479e2e87, 0xf833, 0x4d2b, 0xbb, 0x47, 0x16, 0x77, 0x7b, 0x52, 0xb6, 0x6a	`HII_FONT_PROTOCOL.StringIdToImage - StringIdToImage()` returns `EFI_NOT_FOUND` with an invalid `PackageList`.	Call `StringIdToImage()` with valid parameters except an invalid `PackageList`, The return status should be `EFI_NOT_FOUND`.
5.18.1.2.4	0xe1d5168a, 0x26da, 0x4000, 0xa9, 0xc8, 0x15, 0x85, 0xee, 0xea, 0x38, 0x33	`HII_FONT_PROTOCOL.StringIdToImage - StringIdToImage()` returns `EFI_NOT_FOUND` with `StringId` not in PackageList.	Call `StringIdToIamge()` with a `StringId` which isn't in `PackageList`. The return status should be `EFI_NOT_FOUND`.
5.18.1.2.5	0xfba0a646, 0x9942, 0x4790, 0x86, 0xef, 0xe8, 0x52, 0x32, 0xf1, 0xb5, 0xeb	`HII_FONT_PROTOCOL.StringIdToImage - StringToImage()` returns `EFI_INVALID_PARAMETER` with `invalid Flags combination.`	Call `StringIdToImage()` with `Flag` being `EFI_HII_OUT_FLAG_CLEAN_X` with `EFI_HII_OUT_FLAG_WRAP`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.1.2.6	0xd9b59551, 0xa799, 0x4c87, 0x89, 0xb3, 0x89, 0xc5, 0x6a, 0xb8, 0x43, 0x9f	`HII_FONT_PROTOCOL.StringIdToImage - StringToImage()` returns `EFI_INVALID_PARAMETER` with `invalid Flags combination.`	Call `StringIdToImage()` with `Flag` being `EFI_HII_OUT_FLAG_CLEAN_X` without `EFI_HII_OUT_FLAG_CLIP`. The return status should be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.18.1.2.7	0x3df4b27f, 0x7b07, 0x4a3d, 0xaa, 0x09, 0x60, 0xfa, 0xbe, 0x82, 0x99, 0x9f	HII_FONT_PROTOCOL. StringIdToImage - StringIdToImage() returns EFI_SUCCESS with valid parameters.	Call StringIdToImage() with valid paramenters and use EFI_GRAPHICS_OUTPUT_BLT_PIXEL structure in EFI_IMAGE_OUTPUT structure. The return status should EFI_SUCCESS.
5.18.1.2.8	0xedcca70f, 0xcb25, 0x4d22, 0x98, 0x5e, 0x18, 0x86, 0x66, 0x8c, 0xc1, 0x9c	HII_FONT_PROTOCOL. StringIdToImage - StringIdToImage() returns EFI_SUCCESS with valid parameters.	Call StringIdToImage() with valid paramenters and use EFI_GRAPHICS_OUTPUT_PROTOCOL in EFI_IMAGE_OUTPUT structure. The return status should EFI_SUCCESS.

27.1.3 GetGlyph()

5.18.1.3.1	0xb94b394f, 0x8e3e, 0x4adc, 0x8f, 0x5c, 0x64, 0x12, 0x69, 0xa2, 0xed, 0xfe	HII_FONT_PROTOCOL. GetGlyph - GetGlyph() returns EFI_INVALID_PARAMETER with Blt being NULL.	Call GetGlyph() with Blt being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.1.3.2	0xee445b90, 0xf370, 0x43fd, 0x83, 0xff, 0x00, 0x2d, 0x29, 0x1e, 0xcd, 0x42	HII_FONT_PROTOCOL. GetGlyph - GetGlyph() returns EFI_INVALID_PARAMETER with non NULL *Blt.	Call GetGlyph() with non NULL Blt. The return status should be EFI_INVALID_PARAMETER.
5.18.1.3.3	0x0687a598, 0xa2a6, 0x4073, 0xa7, 0x4f, 0x05, 0xae, 0x9c, 0xe2, 0x1e, 0x33	HII_FONT_PROTOCOL. GetGlyph - GetGlyph() returns EFI_SUCCESS with valid parameters.	Call GetGlyph() with valid parameters. The return status should be EFI_SUCCESS.

27.1.4 GetFontInfo()

Number	GUID	Assertion	Test Description
5.18.1.4.1	0xf43589d3, 0xfccd, 0x413f, 0xb7, 0x50, 0xf8, 0xb4, 0x00, 0xd2, 0x92, 0x7b	HII_FONT_PROTOCOL.GetFontInfo - GetFontInfo() returns EFI_INVALID_PARAMETER with invalid EFI_FONT_INFO_MASK Combination.	Call GetFontInfo() with StringInfoIn->FontInfoMask being invalid combination. The return status should be EFI_INVALID_PARAMETER.
5.18.1.4.2	0x6e5210d4, 0xead5, 0x4042, 0xac, 0x30, 0xa4, 0xfb, 0x8f, 0x9f, 0xf1, 0x9a	HII_FONT_PROTOCOL.GetFont - GetFont() returns EFI_SUCCESS with valid parameters	Call GetFontInfo() with valid parameters. The return status should be EFI_SUCCESS.
5.18.1.4.3	0x88294411, 0x3dd7, 0x4030, 0xb6, 0x40, 0x65, 0xa3, 0x85, 0x7b, 0x2f, 0x46	HII_FONT_PROTOCOL.GetFont - GetFont() returns EFI_SUCCESS with valid parameters(StringInfoIn is NULL)	Call GetFontInfo() with valid parameters(StringInfoIn is NULL). The return status should be EFI_SUCCESS.

27.2 EFI_HII_STRING_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_HII_STRING_PROTOCOL Section.

27.2.1 NewString()

Number	GUID	Assertion	Test Description
5.18.2.1.1	0xb0eb04d6, 0x3328, 0x4157, 0xa8, 0x8e, 0xe9, 0x9a, 0x15, 0x62, 0x6b, 0x88	HII_STRING_PROTOCOL.NewString - NewString() returns EFI_INVALID_PARAMETER with StringId being NULL.	Call NewString() with StringId being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.2.1.2	0x9223196c, 0xadf1, 0x4181, 0xbc, 0xc3, 0x1d, 0x9e, 0xa4, 0xcf, 0x7a, 0x8e	HII_STRING_PROTOCOL.NewString - NewString() returns EFI_INVALID_PARAMETER with Language being NULL.	Call NewString() with Language being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.2.1.3	0x8d9e83aa, 0x9bf1, 0x4466, 0xba, 0xba, 0xec, 0x14, 0xfd, 0xb3, 0x82, 0x14	HII_STRING_PROTOCOL.NewString - NewString() returns EFI_INVALID_PARAMETER with String being NULL.	Call NewString() with String being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.2.1.4	0x23b3df9d, 0x2330, 0x4db7, 0xa1, 0x71, 0x0c, 0x2a, 0x61, 0xb7, 0xd2, 0x24	HII_STRING_PROTOCOL.NewString - NewString() returns EFI_INVALID_PARAMETER with PackageList beinf NULL.	Call NewString() with PackageList being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.2.1.5	0x2077cb3b, 0xb8b4, 0x4ba9, 0xab, 0x49, 0x36, 0xc4, 0xe3, 0xb7, 0x1e, 0xb5	HII_STRING_PROTOCOL.NewString - NewString() returns EFI_SUCCESS with valid parameters and result checked.	Part 1: Call NewString() with valid parameters. The return Status should be EFI_SUCCESS.
5.18.2.1.6	0x8cd4cc42, 0xe5f0, 0x4f6f, 0x9f, 0x7d, 0x60, 0x47, 0x95, 0xd5, 0x05, 0x36	HII_STRING_PROTOCOL.NewString - output the string and compare with the original string.	Part2: Call GetString() to check the output string with the original string. They should be same.

27.2.2 GetString()

Number	GUID	Assertion	Test Description
5.18.2.2.1	0x640acc2d, 0x1174, 0x4735, 0x94, 0xb3, 0xbc, 0xe2, 0xca, 0xbb, 0x92, 0xc1	`HII_STRING_PROTOCOL.GetString - GetString()` returns `EFI_NOT_FOUND` with `StringId` being invalid.	Call `GetString()` with an invalid `StringId`. The return status should be `EFI_NOT_FOUND`.
5.18.2.2.2	0x3c0c9dfe, 0xe56e, 0x43ee, 0x80, 0x26, 0x55, 0xb1, 0x14, 0x29, 0x2c, 0x38	`HII_STRING_PROTOCOL.GetString - GetString()` returns `EFI_NOT_FOUND` with an invalid `PackageList`.	Call `GetString()` with an invalid `PackageList`. The return status should be `EFI_NOT_FOUND`.
5.18.2.2.3	0x0460a672, 0xcba9, 0x4ee8, 0x9e, 0x43, 0x9d, 0xba, 0x85, 0x52, 0x3f, 0xab	`HII_STRING_PROTOCOL.GetString - GetString()` returns `EFI_BUFFER_TOO_SMALL` with `StringSize` indicates the `String` is too small.	Call `GetString()` with `StringSize` which indicates the `String` buffer is small. The return status should `EFI_BUFFER_TOO_SMALL`. The `StringSize` is updated with the required size.
5.18.2.2.4	0xeed5460f, 0x826e, 0x4e1b, 0xad, 0x79, 0xb7, 0x3b, 0x58, 0xc9, 0x57, 0x01	`HII_STRING_PROTOCOL.GetString - GetString()` returns `EFI_INVALID_LANGUAGE` with `string` is not in the specified Language.	Call `GetString()` with string specified by `StringId` is available but not in the specified `Language`. The return status should be `EFI_INVALID_LANGUAGE`.
5.18.2.2.5	0xafd0b70c, 0xe1b4, 0x43c1, 0x94, 0x60, 0x96, 0xf5, 0x3e, 0xe9, 0xaa, 0xe9	`HII_STRING_PROTOCOL.GetString - GetString()` returns `EFI_INVALID_PARAMETER` with `Language` being `NULL`.	Call `GetString()` with `Language` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.2.2.6	0xcf15f5f5, 0x7eaf, 0x4e63, 0x80, 0xd2, 0x5c, 0x9b, 0x89, 0x02, 0x1b, 0xf8	`HII_STRING_PROTOCOL.GetString - GetString()` returns `EFI_INVALID_PARAMETER` with `String` being `NULL`.	Call `GetString()` with `String` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.18.2.2.7	0xc37a209f, 0xaeab, 0x4152, 0xbf, 0x74, 0x27, 0x27, 0xea, 0x48, 0x4f, 0x38	`HII_STRING_PROTOCOL.GetString - GetString()` returns `EFI_INVALID_PARAMETER` with `StringSize` being `NULL`.	Call `GetString()` with `StringSize` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.2.2.8	0x62a545c3, 0x3da2, 0x4f46, 0xb9, 0x07, 0xd4, 0xfe, 0x3e, 0xdf, 0x59, 0xc0	`HII_STRING_PROTOCOL.GetString - GetString()` returns `EFI_INVALID_PARAMETER` with `PackageList` been `NULL`.	Call `GetString()` with `PackageList` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.2.2.9	0x276f380d, 0x96d6, 0x46d5, 0x8a, 0xbb, 0x2a, 0xf3, 0xb7, 0x3c, 0x2d, 0x43	`HII_STRING_PROTOCOL.GetString - GetString()` returns `EFI_SUCCESS` with valid parameters and the result checked.	Step1: Call `NewString()` with valid parameters. Step2: Call `GetString()` with valid parameters. The return status should be `EFI_SUCCESS`. The output string should be same with the original one.

27.2.3 SetString()

Number	GUID	Assertion	Test Description
5.18.2.3.1	0xb7d699ce, 0xb3e9, 0x4327, 0x8b, 0x52, 0xdd, 0xd5, 0xa2, 0xff, 0xb9, 0x0c	HII_STRING_PROTOCOL .SetString - SetString() returns EFI_NOT_FOUND with StringId been invalid.	Call SetString() with an invalid StringId which is not in the database. The return status should be EFI_NOT_FOUND.
5.18.2.3.2	0xfda7ec68, 0xbf34, 0x4086, 0xad, 0x72, 0x26, 0xe1, 0xd6, 0xdd, 0x45, 0x48	HII_STRING_PROTOCOL .SetString - SetString() returns EFI_INVALID_PARAMETER with Language been NULL.	Call SetString() with Language being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.2.3.3	0xb66221c2, 0xc6e7, 0x4129, 0xb3, 0x83, 0xa6, 0x51, 0x26, 0x2b, 0xcf, 0x57	HII_STRING_PROTOCOL .SetString - SetString() returns EFI_INVALID_PARAMETER with String been NULL.	Call SetString() with String being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.2.3.4	0x7439d8aa, 0xe2f6, 0x4c3b, 0x98, 0x0c, 0x13, 0xbd, 0xab, 0x97, 0xff, 0x95	HII_STRING_PROTOCOL .SetString - SetString() returns EFI_NOT_FOUND with an invalid PackageList.	Call SetString() with an invalid PackageList. The return status should be EFI_NOT_FOUND.
5.18.2.3.5	0x66495376, 0x042b, 0x460a, 0xbb, 0x45, 0x19, 0xfd, 0x13, 0xf2, 0xe0, 0x2c	HII_STRING_PROTOCOL .SetString - SetString() returns EFI_INVALID_PARAMETER with PackageList been NULL.	Call SetString() with PackageList being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.2.3.6	0xf346d13b, 0xcbd0, 0x451f, 0xa6, 0x93, 0x75, 0xf1, 0xe9, 0xdd, 0x1f, 0x74	HII_STRING_PROTOCOL .SetString - SetString() returns EFI_SUCCESS with valid parameters and result checked	Part 1: Call SetString() with valid parameters. The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.18.2.3.7	0xbf8f4ae6, 0xf506, 0x43d2, 0xa6, 0x43, 0xa7, 0xb4, 0xb2, 0x33, 0xe8, 0xe0	`HII_STRING_PROTOCOL.SetString` - output the string and compare with the reset string.	Part2: Call `GetString()` to check the output string with the original string. They should be same.

27.2.4 GetLanguages()

Number	GUID	Assertion	Test Description
5.18.2.4.1	0x7a983202, 0x322e, 0x4d12, 0x90, 0xb3, 0xcf, 0x8b, 0x6e, 0xc4, 0x97, 0x5b	HII_STRING_PROTOCOL.GetLanguages - GetLanguages() returns EFI_INVALID_PARAMETER with Languages been NULL.	Call GetLanguages() with Languages being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.2.4.2	0xa9299182, 0xcd9a, 0x4014, 0xb4, 0x03, 0xe2, 0x67, 0xc7, 0xf4, 0x80, 0x7f	HII_STRING_PROTOCOL.GetLanguages - GetLanguages() returns EFI_INVALID_PARAMETER with LanguagesSize been NULL.	Call GetLanguages() with LanguagesSize being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.2.4.3	0x83a0f73c, 0xdd2c, 0x4652, 0x8e, 0xbe, 0x32, 0xd5, 0xf9, 0x8e, 0x24, 0xef	HII_STRING_PROTOCOL.GetLanguages - GetLanguages() returns EFI_NOT_FOUND with an invalid PackageList.	Call GetLanguages() with an invalid PackageList. The return status should be EFI_INVALID_PARAMETER.
5.18.2.4.4	0x696870ed, 0xfff5, 0x4b76, 0x9f, 0x82, 0xbe, 0x78, 0xf6, 0x58, 0x9b, 0x8b	HII_STRING_PROTOCOL.GetLanguages - GetLanguages() returns EFI_INVALID_PARAMETER with PackageList been NULL.	Call GetLanguages() with PackageList being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.2.4.5	0x65dca7c5, 0x85a0, 0x48a0, 0x9a, 0x49, 0xa9, 0xbb, 0xae, 0xa2, 0x55, 0xf3	HII_STRING_PROTOCOL.GetLanguages - GetLanguages() returns EFI_BUFFER_TOO_SMALL with LanguagesSize indicates the Languages is too small.	Call GetLanguages() with LanguagesSize which indicates the Languages buffer is small. The return status should EFI_BUFFER_TOO_SMALL. The LanguagesSize is updated with the required size.
5.18.2.4.6	0xba61367b, 0x33b6, 0x41cc, 0x94, 0x60, 0x54, 0x75, 0xf1, 0xe5, 0x81, 0x89	HII_STRING_PROTOCOL.GetLanguages - GetLanguages() returns EFI_SUCCESS with valid parameters.	Call GetLanguages() with valid parameters. The return status should be EFI_SUCCESS.

27.2.5 GetSecondaryLanguages()

Number	GUID	Assertion	Test Description
5.18.2.5.1	0xff558856, 0xcf19, 0x47b2, 0x89, 0xc0, 0xdb, 0xdf, 0x0e, 0xf5, 0x31, 0xe2	`HII_STRING_PROTOCOL.GetSecondaryLanguages - GetSecondaryLanguages()` returns `EFI_INVALID_PARAMETER` with `FirstLanguage` been `NULL`.	Call `GetSecondaryLanguages()` with `FirstLanguage` being `NULL`. The return status should `EFI_INVALID_PARAMETER`.
5.18.2.5.2	0x05c043da, 0xd0dd, 0x4833, 0xa1, 0x27, 0x92, 0x3b, 0x6a, 0x58, 0x05, 0xdc	`HII_STRING_PROTOCOL.GetSecondaryLanguages - GetSecondaryLanguages()` returns `EFI_INVALID_PARAMETER` with `SecondLanguages` been `NULL`.	Call `GetSecondaryLanguages()` with `SecondLanguages` being `NULL`. The return status should `EFI_INVALID_PARAMETER`.
5.18.2.5.3	0xa891d992, 0x6296, 0x4670, 0xa5, 0xbe, 0x5c, 0x53, 0xaa, 0xc0, 0x34, 0x48	`HII_STRING_PROTOCOL.GetSecondaryLanguages - GetSecondaryLanguages()` returns `EFI_INVALID_PARAMETER` with `SecondLanguagesSize` been `NULL`.	Call `GetSecondaryLanguages()` with `SecondLanguagesSize` being `NULL`. The return status should `EFI_INVALID_PARAMETER`.
5.18.2.5.4	0x050d991f, 0xd6f0, 0x4a07, 0x91, 0x6d, 0x58, 0xde, 0xc2, 0xec, 0xf3, 0x2f	`HII_STRING_PROTOCOL.GetSecondaryLanguages - GetSecondaryLanguages()` returns `EFI_NOT_FOUND` with an invalid `PackageList`.	Call `GetSecondaryLanguages()` with an invalid `PackageList`. The return status should `EFI_NOT_FOUND`.
5.18.2.5.5	0x68d1489e, 0x587b, 0x44e5, 0xb8, 0x72, 0x17, 0xc1, 0x1e, 0xc9, 0xd3, 0xf7	`HII_STRING_PROTOCOL.GetSecondaryLanguages - GetSecondaryLanguages()` returns `EFI_INVALID_PARAMETER` with `PackageList` been `NULL`.	Call `GetSecondaryLanguages()` with `PackageList` being `NULL`. The return status should `EFI_INVALID_PARAMETER`.

Number	GUID	Assertion	Test Description
5.18.2.5.6	0xa25ea8dd, 0x5681, 0x4912, 0xb5, 0xda, 0xe3, 0x04, 0x36, 0x7c, 0x23, 0x89	HII_STRING_PROTOCOL.GetSecondaryLanguages - GetSecondaryLanguages() returns EFI_NOT_FOUND with FirstLanguage is not present in the PackageList.	Call GetSecondaryLanguages() with FirstLanguage which is not in the specified PackageList. The return status should EFI_NOT_FOUND.
5.18.2.5.7	0x6750c8c6, 0x54b5, 0x4a95, 0xa4, 0x15, 0x44, 0xbc, 0x64, 0xb1, 0x9f, 0x81	HII_STRING_PROTOCOL.GetLanguages - GetSecondaryLanguages() returns EFI_BUFFER_TOO_SMALL with SecondLanguagesSize indicates the SecondLanguages is too small.	Call GetSecondaryLanguages() with SecondLanguagesSize which indicates the SecondLanguages buffer is small. The return status should EFI_BUFFER_TOO_SMALL. The SecondLanguagesSize is updated with the required size.
5.18.2.5.8	0x302b21ca, 0xbc47, 0x4c26, 0xa0, 0x21, 0x24, 0x2d, 0xba, 0x57, 0x42, 0x65	HII_STRING_PROTOCOL.GetSecondaryLanguages - GetSecondaryLanguages() returns EFI_SUCCESS with SecondLanguagesSize is large enough.	Call GetSecondaryLanguages() with valid parameters. The return status should be EFI_SUCCESS.

27.3 EFI_HII_IMAGE_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_HII_IMAGE_PROTOCOL Section.

27.3.1 NewImage()

Number	GUID	Assertion	Test Description
5.18.3.1.1	0x20eafa16, 0xc9cd, 0x41b3, 0x96, 0x81, 0x46, 0x7b, 0x7f, 0x17, 0x3d, 0x71	**HII_IMAGE_PROTOCOL. NewImage -** **NewImage()** returns **EFI_INVALID_PARAMETER** with **ImageId** been **NULL**.	Call **NewImage()** with **ImageId** being **NULL**, The return status should be **EFI_INVALID_PARAMETER**.
5.18.3.1.2	0x0227338d, 0xb459, 0x4209, 0xb1, 0xa0, 0x10, 0x3c, 0xe8, 0x3e, 0x71, 0xf5	**HII_IMAGE_PROTOCOL. NewImage -** **NewImage()** returns **EFI_INVALID_PARAMETER** with **Image** been **NULL.**	Call **NewImage()** with **Image** being **NULL**, The return status should be **EFI_INVALID_PARAMETER**.
5.18.3.1.3	0x4930f94e, 0x6bdb, 0x42aa, 0xaf, 0xde, 0x87, 0x55, 0x55, 0x2c, 0x77, 0x1d	**HII_IMAGE_PROTOCOL. NewImage -** **NewImage()** returns **EFI_NOT_FOUND** with **PackageList** been **NULL**.	Call **NewImage()** with **PackageList** being **NULL**, The return status should be **EFI_NOT_FOUND**.
5.18.3.1.4	0x170bc177, 0xa2f7, 0x46ba, 0xa8, 0xd6, 0x09, 0xe5, 0xa4, 0xb1, 0x81, 0x8f	**HII_IMAGE_PROTOCOL. NewImage -** **NewImage()** returns **EFI_SUCCESS** with valid parameters and result checked.	Call **NewImage()** with valid parameters, The return status should be **EFI_SUCCESS**.

27.3.2 GetImage()

Number	GUID	Assertion	Test Description
5.18.3.2.1	0x55488ca5, 0x2a0c, 0x4dcb, 0xbc, 0x7d, 0xca, 0xaf, 0x05, 0x2f, 0xac, 0x13	HII_IMAGE_PROTOCOL.GetImage - GetImage() returns EFI_NOT_FOUND with ImageId been invalid.	Call GetImage() with an invalid ImageId which is not in the database. The return status should be EFI_NOT_FOUND.
5.18.3.2.2	0xdde7e63e, 0xa889, 0x47ce, 0xad, 0xe1, 0x15, 0x0b, 0xb8, 0xa3, 0x8e, 0x10	HII_IMAGE_PROTOCOL.GetImage - GetImage() returns EFI_BUFFER_TOO_SMALL with ImageSize is small.	Call GetImage() with ImageSize which indicates the Image buffer is small. The return status should EFI_BUFFER_TOO_SMALL. The ImageSize is updated with the required size.
5.18.3.2.3	0xa1f286a0, 0x26da, 0x4919, 0xa3, 0xc4, 0x90, 0x5b, 0x18, 0x03, 0x6c, 0x36	HII_IMAGE_PROTOCOL.GetImage - GetImage() returns EFI_INVALID_PARAMETER with Image been NULL.	Call GetImage() with Image being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.3.2.4	0x17a11dcc, 0x8d3d, 0x40dc, 0xb0, 0x9c, 0x37, 0xfc, 0x8e, 0x72, 0x46, 0xab	HII_IMAGE_PROTOCOL.GetImage - GetImage() returns EFI_INVALID_PARAMETER with ImageSize been NULL.	Call GetImage() with ImageSize being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.3.2.5	0x51363bef, 0x2eb6, 0x4eef, 0x86, 0xdf, 0x48, 0xf1, 0x87, 0x75, 0x6f, 0x9e	HII_IMAGE_PROTOCOL.GetImage - GetImage() returns EFI_NOT_FOUND with an invalid PackageList.	Call GetImage() with an invalid PackageList. The return status should be EFI_NOT_FOUND.
5.18.3.2.6	0x05fc7f10, 0xe1ef, 0x4fd0, 0x91, 0x3d, 0x86, 0x46, 0x53, 0x7e, 0x4c, 0xbd	HII_IMAGE_PROTOCOL.GetImage - GetImage() returns EFI_NOT_FOUND with PackageList been NULL.	Call GetImage() with PackageList being NULL. The return status should be EFI_NOT_FOUND.

Number	GUID	Assertion	Test Description
5.18.3.2.7	0x14cd0647, 0x3fd7, 0x4831, 0x9e, 0xa5, 0x9b, 0x3d, 0xd7, 0xc8, 0xeb, 0xb7	`HII_IMAGE_PROTOCOL.GetImage - GetImage()` returns `EFI_SUCCESS` with valid parameters and the result checked.	Call `GetImage()` with valid parameters, The return status should be `EFI_SUCCESS`.

27.3.3 SetImage()

Number	GUID	Assertion	Test Description
5.18.3.3.1	0x9af36ab7, 0x8bd2, 0x417b, 0xa5, 0x10, 0x1f, 0x22, 0x99, 0x13, 0x72, 0x64	HII_IMAGE_PROTOCOL.SetImage - SetImage() returns EFI_NOT_FOUND with ImageId been invalid	Call SetImage() with an invalid ImageId which is not in the database. The return status should be EFI_NOT_FOUND.
5.18.3.3.2	0x5d9b72d9, 0x01f4, 0x47cd, 0x96, 0xbb, 0xb1, 0xf2, 0xf2, 0x1f, 0xf7, 0x2a	HII_IMAGE_PROTOCOL.SetImage - SetImage() returns EFI_INVALID_PARAMETER with Image been NULL.	Call SetImage() with Image being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.3.3.3	0xa411c5ef, 0x0eeb, 0x4a9a, 0x85, 0x9a, 0x4a, 0x64, 0x0d, 0xa6, 0x16, 0xf7	HII_IMAGE_PROTOCOL.SetImage - SetImage() returns EFI_NOT_FOUND with an invalid PackageList.	Call SetImage() with an invalid PackageList. The return status should be EFI_NOT_FOUND.
5.18.3.3.4	0x870c9c4c, 0xe099, 0x4024, 0xac, 0x3a, 0x7b, 0x8c, 0x30, 0x98, 0x8c, 0x2e	HII_IMAGE_PROTOCOL.SetImage - SetImage() returns EFI_NOT_FOUND with PackageList been NULL.	Call SetImage() with PackageList being NULL. The return status should be EFI_NOT_FOUND.
5.18.3.3.5	0xc99ad1a4, 0x3f5b, 0x46dc, 0xb4, 0x85, 0xb2, 0x23, 0x9d, 0xef, 0xbc, 0x2c	HII_IMAGE_PROTOCOL.SetImage - SetImage() returns EFI_SUCCESS with valid parameters and result checked.	Call SetImage() with valid parameters, The return status should be EFI_SUCCESS.

27.3.4 DrawImage()

Number	GUID	Assertion	Test Description
5.18.3.4.1	0x4bb8ee94, 0x8a57, 0x470f, 0x9d, 0xd5, 0xef, 0x81, 0xea, 0xd9, 0xd6, 0xad	HII_IMAGE_PROTOCOL.DrawImage - DrawImage() returns EFI_INVALID_PARAMETER with Image been NULL.	Call DrawImage() with Image being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.3.4.2	0xd9034d5d, 0xde07, 0x4458, 0x92, 0xb7, 0x4c, 0xd1, 0x50, 0x1c, 0xe8, 0x90	HII_IMAGE_PROTOCOL.DrawImage - DrawImage() returns EFI_INVALID_PARAMETER with EFI_HII_DRAW_FLAG_FORCE_TRANS and Blt been NULL.	Call DrawImage() with Flags being EFI_HII_DRAW_FLAG_FORCE_TRANS and Blt being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.3.4.3	0x5c232904, 0x23f8, 0x4b0f, 0x9c, 0x85, 0xb7, 0xe8, 0xa5, 0xc9, 0x80, 0x05	HII_IMAGE_PROTOCOL.DrawImage - DrawImage() returns EFI_INVALID_PARAMETER with EFI_HII_DIRECT_TO_SCREEN and no screen.	Call DrawIamge() with Flags being EFI_HII_DIRECT_TO_SCREEN and use EFI_GRAPHICS_OUTPUT_BLT_PIXEL structure in EFI_IMAGE_OUTPUT structure. The return status should be EFI_INVALID_PARAMETER.
5.18.3.4.4	0xf9e86ff1, 0x611c, 0x41b8, 0xb0, 0x8d, 0x2a, 0xe2, 0x5e, 0x34, 0x2a, 0x1d	HII_IMAGE_PROTOCOL.DrawImage - DrawImage() returns EFI_INVALID_PARAMETER with EFI_HII_DRAW_FLAG_CLIP and Blt points to NULL.	Call DrawImage() with Flags being EFI_HII_DRAW_FLAG_CLIP and Blt being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.3.4.5	0x11ae81e8, 0xfe20, 0x472d, 0x8c, 0xdb, 0x40, 0xb7, 0x56, 0x09, 0xd9, 0xdc	HII_IMAGE_PROTOCOL.DrawImage - DrawImage() returns EFI_INVALID_PARAMETER with EFI_HII_DRAW_FLAG_DEFAULT, Blt points to NULL and Image->Flags is EFI_IMAGE_TRANSPARENT.	Call DrawImage() with Blt being NULL, Flags being EFI_HII_DRAW_FLAG_DEFAULT and Image->Flags being EFI_IMAGE_TRANSPARENT. The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.18.3.4.6	0x6e409e86, 0x16d3, 0x4b31, 0x96, 0x71, 0xf9, 0x2c, 0xe6, 0x26, 0x1b, 0xcf	HII_IMAGE_PROTOCOL.DrawImage - DrawImage() returns EFI_SUCCESS with valid parameter.	Call DrawImage() with Flags being EFI_HII_DRAW_FLAG_FORCE_OPAQUE, Blt being NULL and other valid parameters. The return status should be EFI_SUCCESS.
5.18.3.4.7	0xedbef6eb, 0xf68f, 0x4154, 0xb0, 0x12, 0xb9, 0xd7, 0x55, 0x3b, 0xa6, 0x0a	HII_IMAGE_PROTOCOL.DrawImage - DrawImage() returns EFI_SUCCESS with valid parameter.	Call DrawImage() with Flags being valid combination, Blt being not NULL and other valid parameters. The return status should be EFI_SUCCESS.

27.3.5 DrawImageId()

Number	GUID	Assertion	Test Description
5.18.3.5.1	0xcb1936c7, 0x53c7, 0x4a65, 0xa5, 0x3d, 0x85, 0xc2, 0x35, 0x72, 0xff, 0x29	HII_IMAGE_PROTOCOL .DrawImageId - DrawImageId() returns EFI_NOT_FOUND with an invalid PackageList.	Call DrawImageId() with an invalid PackageList. The return status should be EFI_NOT_FOUND.
5.18.3.5.2	0xb1372c26, 0x3de4, 0x4a5c, 0x8a, 0x1f, 0x71, 0x4a, 0x7b, 0x07, 0x0e, 0x67	HII_IMAGE_PROTOCOL .DrawImageId - DrawImageId() returns EFI_NOT_FOUND with PackageList been NULL.	Call DrawImageId() with PackageList being NULL. The return status should be EFI_NOT_FOUND.
5.18.3.5.3	0x23a7fcfd, 0x4d0f, 0x4460, 0xb8, 0xcc, 0x7a, 0xfa, 0xf7, 0x4d, 0xe5, 0xaa	HII_IMAGE_PROTOCOL .DrawImageId - DrawImageId() returns EFI_NOT_FOUND with invalid ImageId.	Call DrawImageId() with an invalid ImageId which is not in the specified PackageList. The return status should be EFI_NOT_FOUND.
5.18.3.5.4	0x5433fcf6, 0x06f4, 0x45f3, 0x91, 0x23, 0x79, 0x5f, 0x49, 0x69, 0x77, 0x4d	HII_IMAGE_PROTOCOL .DrawImageId - DrawImageId() returns EFI_NOT_FOUND with invalid PackageList.	Call DrawImageId() with an invalid PackageList which is not in the database. The return status should be EFI_NOT_FOUND.
5.18.3.5.5	0x2df19349, 0xec8c, 0x42f7, 0x9f, 0x8e, 0x1d, 0x56, 0x13, 0x6c, 0x95, 0xbc	HII_IMAGE_PROTOCOL .DrawImageId - DrawImage() returns EFI_INVALID_PARAMETER with EFI_HII_DRAW_FLAG_FORCE_TRANS and Blt been NULL.	Call DrawImageId() with Flags being EFI_HII_DRAW_FLAG_FORCE_TRANS and Blt being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.3.5.6	0x059732f0, 0x431e, 0x4ad3, 0x92, 0xa0, 0x4b, 0xda, 0xaa, 0x8d, 0x98, 0x92	HII_IMAGE_PROTOCOL .DrawImage - DrawImage() returns EFI_INVALID_PARAMETER with EFI_HII_DIRECT_TO_SCREEN and no screen.	Call DrawIamgeId() with Flags being EFI_HII_DIRECT_TO_SCREEN and use EFI_GRAPHICS_OUTPUT_BLT_PIXEL structure in EFI_IMAGE_OUTPUT structure. The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.18.3.5.7	0xd12320fa, 0x063e, 0x48e3, 0x85, 0xd5, 0x1c, 0x9b, 0x7c, 0x48, 0x71, 0x13	`HII_IMAGE_PROTOCOL.DrawImageId - DrawImageId()` returns `EFI_INVALID_PARAMETER` with `EFI_HII_DRAW_FLAG_CLIP` and `Blt` points to `NULL.`	Call `DrawImageId()` with `Flags` being `EFI_HII_DRAW_FLAG_CLIP` and `Blt` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.3.5.8	0xb3e326cb, 0x67bc, 0x49a7, 0x8c, 0xb6, 0xc3, 0xec, 0x3b, 0x83, 0x20, 0x1e	`HII_IMAGE_PROTOCOL.DrawImageId - DrawImageId()` returns `EFI_INVALID_PARAMETER` with `EFI_HII_DRAW_FLAG_DEFAULT`, `Blt` points to `NULL and Image->Flags is EFI_IMAGE_TRANSPARENT.`	Call `DrawImageId()` with `Blt` being `NULL`, `Flags` being `EFI_HII_DRAW_FLAG_DEFAULT` and `Image`->`Flags` being `EFI_IMAGE_TRANSPARENT`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.3.5.9	0xaeeb761e, 0x1b38, 0x4b06, 0x8d, 0x26, 0xf3, 0x6f, 0xde, 0xa4, 0x3d, 0x88	`HII_IMAGE_PROTOCOL.DrawImageId - DrawImageId()` returns `EFI_SUCCESS` with valid parameter.	Call `DrawImageId()` with `Flags` being `EFI_HII_DRAW_FLAG_FORCE_OPAQUE`, `Blt` being `NULL` and other valid parameters. The return status should be `EFI_SUCCESS`.
5.18.3.5.10	0x2b844dec, 0xc8cf, 0x442c, 0x89, 0xc0, 0x9f, 0x44, 0xe0, 0x96, 0x4b, 0xcb	`HII_IMAGE_PROTOCOL.DrawImageId - DrawImageId()` returns `EFI_SUCCESS` with valid parameter.	Call `DrawImage()` with `Flags` being valid combination, `Blt` being not `NULL` and other valid parameters. The return status should be `EFI_SUCCESS`.

27.4 EFI_HII_DATABASE_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_HII_DATABASE_PROTOCOL Section.

27.4.1 NewPackageList ()

Number	GUID	Assertion	Test Description
5.18.4.1.1	0x17364518, 0x35c4, 0x481a, 0x82, 0x45, 0xdd, 0x8b, 0x85, 0xbf, 0x01, 0x7c	HII_DATABASE_PROTOCOL.NewPackageList - NewPackageList() returns EFI_INVALID_PARAMETER with PackageList being NULL.	Call NewPackageList() with PackageList being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.4.1.2	0xd12127b3, 0x3a61, 0x498d, 0xbb, 0x8f, 0x9f, 0x9e, 0xb3, 0x9a, 0xfd, 0x95	HII_DATABASE_PROTOCOL.NewPackageList - NewPackageList() returns EFI_INVALID_PARAMETER with Handle being NULL.	Call NewPackageList() with Handle being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.4.1.3	0x3ee6503d, 0x5fab, 0x4f51, 0x9a, 0xee, 0xc9, 0x0f, 0x9d, 0x73, 0xe5, 0xd7	HII_DATABASE_PROTOCOL.NewPackageList - NewPackageList() returns EFI_SUCCESS with valid inputs	Call NewPackageList() with valid parameters. The return status should be EFI_SUCCESS.

27.4.2 RemovePackageList ()

Number	GUID	Assertion	Test Description
5.18.4.2.1	0x244e5792, 0x471b, 0x456b, 0x8b, 0xfe, 0x1f, 0x68, 0xeb, 0x8f, 0xcd, 0xd0	HII_DATABASE_PROTOCOL.RemovePackageList - RemovePackageList() returns EFI_NOT_FOUND with Handle being NULL.	Call RemovePackageList() with Handle being NULL. The return status should be EFI_NOT_FOUND.
5.18.4.2.2	0x4f2588b4, 0xadb6, 0x48ba, 0xac, 0x53, 0x97, 0x3e, 0x05, 0x64, 0x5d, 0x4f	HII_DATABASE_PROTOCOL.RemovePackageList - RemovePackageList() returns EFI_NOT_FOUND with Handle has already been removed once.	Call RemovePackageList() with Handle which has been removed. The return status should be EFI_NOT_FOUND.
5.18.4.2.3	0x28c05503, 0x33ce, 0x41ae, 0x90, 0x2e, 0xbc, 0x34, 0xe0, 0xb8, 0x0e, 0x9d	HII_DATABASE_PROTOCOL.RemovePackageList - RemovePackageList() returns EFI_NOT_FOUND with an invalid Handle.	Call RemovePackageList() with an invalid Handle. The return status should be EFI_NOT_FOUND.
5.18.4.2.4	0xb4a3a9ac, 0x0dfa, 0x4025, 0xa6, 0x36, 0xac, 0x53, 0x19, 0x7a, 0x5e, 0xd2	HII_DATABASE_PROTOCOL.RemovePackageList - RemovePackageList() returns EFI_SUCCESS with valid inputs.	Part1: Call RemovePackageList() with valid parameters. The return status should be EFI_SUCCESS.
5.18.4.2.5	0xad310e29, 0x2112, 0x485b, 0xa4, 0xdc, 0xc8, 0xec, 0xf8, 0x49, 0x7b, 0xc9	HII_DATABASE_PROTOCOL.RemovePackageList - ExportPackageLists() returns EFI_NOT_FOUND when RemovePackageList work.	Part2: Call ExportPackageList() with Handle which has been removed. The return status should be EFI_NOT_FOUND.

27.4.3 UpdatePackageList ()

Number	GUID	Assertion	Test Description
5.18.4.3.1	0xb4bf4c19, 0x64cc, 0x4efe, 0xa7, 0x21, 0x3f, 0xc2, 0x07, 0x88, 0x51, 0xb4	`HII_DATABASE_PROTOCOL.UpdatePackageList - UpdatePackageList()` returns `EFI_NOT_FOUND` with `Handle` being `NULL.`	Call `UpdatePackageList()` with `Handle` being `NULL`. The return status should be `EFI_NOT_FOUND`.
5.18.4.3.2	0xcd591535, 0x7df7, 0x4f99, 0x9d, 0x13, 0x3b, 0x8e, 0x39, 0x85, 0x39, 0x6f	`HII_DATABASE_PROTOCOL.UpdatePackageList - UpdatePackageList()` returns `EFI_NOT_FOUND` with `Handle` has already been removed before.	Call `UpdatePackageList()` with `Handle` which has been removed. The return status should be `EFI_NOT_FOUND`.
5.18.4.3.3	0x2a2f8bf0, 0x0c27, 0x41f3, 0xae, 0x19, 0xb0, 0x66, 0x16, 0x92, 0x5c, 0x0b	`HII_DATABASE_PROTOCOL.UpdatePackageList - UpdatePackageList()` returns `EFI_NOT_FOUND` with an invalid `handle`.	Call `UpdatePackageList()` with an invalid `Handle`. The return status should be `EFI_NOT_FOUND`.
5.18.4.3.4	0xfcb45969, 0x37f8, 0x430e, 0x86, 0x99, 0x7f, 0x89, 0xde, 0x52, 0x6f, 0x94	`HII_DATABASE_PROTOCOL.UpdatePackageList - UpdatePackageList()` returns `EFI_INVALID_PARAMETER` with `PackageList` been `NULL`.	Call `UpdatePackageList()` with `PackageList` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.4.3.5	0xe1f18c0b, 0xfa2f, 0x488a, 0x80, 0x25, 0x77, 0x35, 0x49, 0x55, 0x36, 0xe0	`HII_DATABASE_PROTOCOL.UpdatePackageList - UpdatePackageList()` returns `EFI_SUCCESS` with valid inputs	Call `UpdatePackageList()` with valid parameters. The return status should be `EFI_SUCCESS`.

27.4.4 ListPackageLists()

Number	GUID	Assertion	Test Description
5.18.4.4.1	0x7b5c4246, 0xe6b3, 0x4eb0, 0xaf, 0xc4, 0x23, 0xb1, 0xbf, 0xfd, 0x46, 0x39	`HII_DATABASE_PROTOCOL.ListPackageLists - ListPackageLists()` returns `EFI_INVALID_PARAMETER` with `Handle` being `NULL`.	Call `ListPackageList()` with `Handle` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.4.4.2	0x9268a2d0, 0xc922, 0x42bc, 0xb0, 0x5d, 0x3d, 0x18, 0xab, 0xf2, 0xe9, 0x37	`HII_DATABASE_PROTOCOL.ListPackageLists - ListPackageLists()` returns `EFI_INVALID_PARAMETER` with `HandleBufferLength` being `NULL`.	Call `ListPackageList()` with `HandleBufferLength` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.4.4.3	0x7c577327, 0x562c, 0x4333, 0x9b, 0x81, 0x9b, 0xf6, 0xf2, 0x80, 0x83, 0xec	`HII_DATABASE_PROTOCOL.ListPackageLists - ListPackageLists()` returns `EFI_NOT_FOUND` when no matching `handles` were found.	Call `ListPackageList()` with no match `Handle` being found. The return status should be `EFI_NOT_FOUND`.
5.18.4.4.4	0xf5771b8e, 0x6db5, 0x473d, 0xba, 0x32, 0x21, 0xfe, 0xf2, 0x7f, 0x05, 0xf2	`HII_DATABASE_PROTOCOL.ListPackageLists - ListPackageLists()` returns `EFI_BUFFER_TOO_SMALL` when the `HandleBufferLength` indicates the buffer is too small.	Part1: Call `ListPackageList()` with `HandleBufferLength` which indicates the `Handle` buffer is small. The return status should `EFI_BUFFER_TOO_SMALL`.
5.18.4.4.5	0x08c276ef, 0x185c, 0x4eac, 0xbe, 0x84, 0x7d, 0xb0, 0x8c, 0x38, 0x5f, 0xe7	`HII_DATABASE_PROTOCOL.ListPackageLists - ListPackageLists()` returns `EFI_BUFFER_TOO_SMALL` when the `HandleBufferLength` indicates the buffer is too small and return the needed buffer length.	Part 2: The `HandleBufferLength` is updated with the required size.

Number	GUID	Assertion	Test Description
5.18.4.4.6	0x212bb7e2, 0xa998, 0x4ede, 0xba, 0x08, 0x8d, 0x8c, 0xda, 0x9d, 0xb7, 0xd4	`HII_DATABASE_PROTOCOL.ListPackageLists - ListPackageLists()` returns `EFI_INVALID_PARAMETER` with `PackageType` is not `Guid` and `PackageGuid` is not `NULL`.	Call `ListPackageList()` with no Guid `PackageType` and no `NULL` `PackageGuid`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.4.4.7	0x9b711922, 0x06d3, 0x4ba4, 0x98, 0x5b, 0x50, 0x72, 0x46, 0x94, 0x8b, 0xb2	`HII_DATABASE_PROTOCOL.ListPackageLists - ListPackageLists()` returns `EFI_INVALID_PARAMETER` with `PackageType` is `EFI_HII_DATABASE_TYPE_GUID` and `PackageGuid` is `NULL.`	Call `ListPackageList()` with Guid `PackageType` and `PackageGuid` being `NULL`. The return status should `EFI_INVALID_PARAMETER`.
5.18.4.4.8	0x1dd024a0, 0xc53b, 0x439e, 0x86, 0x43, 0xc3, 0xe2, 0x82, 0x1f, 0x34, 0x75	`HII_DATABASE_PROTOCOL.ListPackageLists - ListPackageLists()` returns `EFI_SUCCESS` with valid inputs and return length checked.	Call `ListPackageList()` with valid parameters. The return status should be `EFI_SUCCESS`.

27.4.5 ExportPackageLists ()

Number	GUID	Assertion	Test Description
5.18.4.5.1	0xdc1afed1, 0x5be4, 0x4488, 0xaf, 0xeb, 0x75, 0x70, 0xb6, 0x3d, 0xea, 0xc4	HII_DATABASE_PROTOCOL.ExportPackageLists - ExportPackageLists() returns EFI_INVALID_PARAMETER with BufferSize being NULL.	Call ExportPackageList() with BufferSize being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.4.5.2	0xd25ed0fa, 0xe829, 0x4e68, 0xbb, 0xa3, 0xef, 0x82, 0x5a, 0xa0, 0xba, 0x85	HII_DATABASE_PROTOCOL.ExportPackageLists - ExportPackageLists() returns EFI_INVALID_PARAMETER with Buffer being NULL	Call ExportPackageList() with Buffer being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.4.5.3	0x0462bf1f, 0xce31, 0x4314, 0xbd, 0x34, 0x40, 0x4a, 0x05, 0x04, 0xd3, 0x0c	HII_DATABASE_PROTOCOL.ExportPackageLists - ExportPackageLists() returns EFI_BUFFER_TOO_SMALL with BufferSize indicates the buffer is too small.	Part1: Call ExportPackageList() with BufferSize which indicates the Buffer is small. The return status should EFI_BUFFER_TOO_SMALL.
5.18.4.5.4	0xf03af69e, 0x3bba, 0x4092, 0xb0, 0x40, 0x75, 0x4b, 0x42, 0x6b, 0x2f, 0xd0	HII_DATABASE_PROTOCOL.ExportPackageLists - ExportPackageLists() returns EFI_BUFFER_TOO_SMALL with BufferSize indicates the buffer is too small and return the needed BufferSize.	Part2: The BufferSize is updated with the required size.
5.18.4.5.5	0x55ce12c1, 0x35eb, 0x4d8c, 0xbf, 0xd9, 0x9b, 0x0c, 0x52, 0x4d, 0xc0, 0x76	HII_DATABASE_PROTOCOL.ExportPackageLists - ExportPackageLists() returns EFI_NOT_FOUND with handle has been already removed once.	Call ExportPackageList() with Handle which has been removed once. The return status should be EFI_NOT_FOUND.

Number	GUID	Assertion	Test Description
5.18.4.5.6	0x22a02d74, 0xc2a8, 0x439f, 0xbd, 0x4c, 0xf6, 0xb0, 0x1a, 0xbe, 0x03, 0xe4	`HII_DATABASE_PROTOCOL.ExportPackageLists – ExportPackageLists()` returns `EFI_NOT_FOUND` with the invalid handle.	Call `ExportPackageList()` with an invalid `Handle`. The return status should be `EFI_NOT_FOUND`.
5.18.4.5.7	0xc9741024, 0x3073, 0x4827, 0x92, 0x23, 0x06, 0x33, 0x96, 0x0b, 0x8d, 0x6d	`HII_DATABASE_PROTOCOL.ExportPackageLists – ExportPackageLists()` returns `EFI_SUCCESS` with valid inputs and result checked.	Call `ExportPackageList()` with valid parameters. The return status should be `EFI_SUCCESS`.

27.4.6 RegisterPackageNotify()

Number	GUID	Assertion	Test Description
5.18.4.6.1	0x1665f366, 0x70af, 0x4348, 0xbb, 0xc8, 0xb1, 0xaf, 0x38, 0xe1, 0x2d, 0xfd	`HII_DATABASE_PROTOCOL.RegisterPackageNotify - RegisterPackageNotify()` returns `EFI_INVALID_PARAMETER` with `NotifyHandle` been `NULL`.	Call `RegisterPackageNotify()` with `NotifyHandle` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.4.6.2	0x7541d67b, 0xe837, 0x46bf, 0x85, 0x7e, 0xbc, 0x22, 0xf2, 0xe1, 0x0d, 0x60	`HII_DATABASE_PROTOCOL.RegisterPackageNotify - RegisterPackageNotify()` returns `EFI_INVALID_PARAMETER` with `PackageType` is not `Guid` and `PackageGuid` not been `NULL`.	Call `RegisterPackageNotify()` with no Guid `PackageType` and no `NULL PackageGuid`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.4.6.3	0x625abf38, 0x2d02, 0x46b2, 0xae, 0xa9, 0xcc, 0x5b, 0x0c, 0x83, 0xf1, 0x69	`HII_DATABASE_PROTOCOL.RegisterPackageNotify - RegisterPackageNotify()` returns `EFI_INVALID_PARAMETER` with `PackageType` is `EFI_HII_PACKAGE_TYPE_GUID` and `PackageGuid` been `NULL.`	Call `RegisterPackageNotify()` with Guid `PackageType` and `NULL PackageGuid`. The return status should be `EFI_INVALID_PARAMETER`.

27.4.7 UnregisterPackageNotify()

Number	GUID	Assertion	Test Description
5.18.4.7.1	0xef67f1ff, 0x9b53, 0x40ac, 0x8e, 0xec, 0xca, 0x5c, 0x59, 0xfd, 0xbd, 0x0d	HII_DATABASE_PROTOCOL.UnregisterPackageNotify - UnregisterPackageNotify() returns EFI_NOT_FOUND with the NotifyHandle has been removed already.	Call UnRegisterPackageNotify() with NotifyHandle which has been removed once. The return status should be EFI_NOT_FOUND.
5.18.4.7.2	0xc5266e09, 0xe5e8, 0x4c85, 0xb3, 0x0a, 0xc9, 0x83, 0x04, 0x4f, 0x23, 0xfc	HII_DATABASE_PROTOCOL.UnregisterPackageNotify - UnregisterPackageNotify() returns EFI_NOT_FOUND with an invalid NotifyHandle.	Call UnRegisterPackageNotify() with NotifyHandle which can't be found in the database. The return status should be EFI_NOT_FOUND.
5.18.4.7.3	0x51c64bb1, 0x3266, 0x4ccd, 0x82, 0xde, 0xed, 0x6b, 0xa7, 0x68, 0x35, 0xe5	HII_DATABASE_PROTOCOL.UnregisterPackageNotify - UnregisterPackageNotify() returns EFI_NOT_FOUND with NotifyHandle been NULL.	Call UnRegisterPackageNotify() with NotifyHandle being NULL. The return status should be EFI_NOT_FOUND.

27.4.8 FindKeyboardLayouts()

Number	GUID	Assertion	Test Description
5.18.4.8.1	0xa61bf4b2, 0xb1e0, 0x4e62, 0x95, 0x2d, 0xa0, 0x68, 0x98, 0x48, 0x06, 0xb2	HII_DATABASE_PROTOCOL.FindKeyboardLayouts - FindKeyboardLayouts() returns EFI_INVALID_PARAMETER with KeyGuidBufferLength been NULL.	Call FindKeyboardLayouts() with KeyGuidBufferLength being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.4.8.2	0x1ea6e881, 0x6f47, 0x4fdc, 0x8b, 0x8c, 0xba, 0x33, 0x9a, 0x13, 0xbe, 0xc0	HII_DATABASE_PROTOCOL.FindKeyboardLayouts - FindKeyboardLayouts() returns EFI_INVALID_PARAMETER with KeyGuidBuffer been NULL.	Call FindKeyboardLayouts() with KeyGuidBuffer being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.4.8.3	0xc3bacca3, 0x901a, 0x49ad, 0xa9, 0x86, 0x41, 0x62, 0xff, 0xb3, 0xa1, 0x8f	HII_DATABASE_PROTOCOL.FindKeyboardLayouts - FindKeyboardLayouts() returns EFI_BUFFER_TOO_SMALL with KeyGuidBufferLength indicates the buffer is too small.	Call FindKeyboardLayouts() with KeyGuidBufferLength which indicates KeyGuidBuffer is small. The return status should be EFI_BUFFER_TOO_SMALL. The KeyGuidBufferLength should be updated with required length.
5.18.4.8.4	0x1dc41f45, 0x9e3a, 0x41e2, 0x8f, 0x99, 0x8d, 0x4d, 0x39, 0x32, 0x12, 0x85	HII_DATABASE_PROTOCOL.FindKeyboardLayouts - FindKeyboardLayouts() returns EFI_SUCCESS with valid inputs.	Call FindKeyboardLayouts() with valid parameters. The return status should be EFI_SUCCESS. The KeyGuidBufferLength should be updated with actual length.

27.4.9 GetKeyboardLayout()

Number	GUID	Assertion	Test Description
5.18.4.9.1	0xbc4b4ea1, 0x069c, 0x459c, 0x8c, 0x22, 0x68, 0x19, 0x01, 0x71, 0x78, 0x48	HII_DATABASE_PROTOCOL.GetKeyboardLayout – GetKeyboardLayout() returns EFI_INVALID_PARAMETER with KeyboardLayoutLength been NULL.	Call GetKeyboardLayout() with KeyboardLayoutLength being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.4.9.2	0xe2aeca1e, 0x5c50, 0x4ee7, 0x8f, 0x69, 0x46, 0xa7, 0xb9, 0x01, 0x3e, 0x0d	HII_DATABASE_PROTOCOL.GetKeyboardLayout – GetKeyboardLayout() returns EFI_INVALID_PARAMETER with KeyboardLayout been NULL.	Call GetKeyboardLayout() with KeyboardLayout being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.4.9.3	0x0d41d349, 0xe5f4, 0x43d5, 0x85, 0x0e, 0xfe, 0x4f, 0x08, 0x5a, 0xbf, 0xb2	HII_DATABASE_PROTOCOL.GetKeyboardLayout – GetKeyboardLayout() returns EFI_BUFFER_TOO_SMALL with KeyboardLayoutLength not enough.	Call GetKeyboardLayout() with KeyboardLayoutLength which indicates KeyboardLayout is small. The return status should be EFI_BUFFER_TOO_SMALL. The KeyboardLayoutLength should be updated with required length.
5.18.4.9.4	0xc2732202, 0x48ca, 0x49f8, 0xbb, 0x18, 0xd3, 0x6c, 0xe1, 0xb4, 0x83, 0xfa	HII_DATABASE_PROTOCOL.GetKeyboardLayout – GetKeyboardLayout() returns EFI_NOT_FOUND with the requested keyboard layout not found.	Call GetKeyboardLayout() with a Guid which can’t be found in the database. The return status should be EFI_NOT_FOUND.
5.18.4.9.5	0x4ffc59ee, 0xefb8, 0x4533, 0x81, 0x4f, 0x85, 0xed, 0x90, 0x93, 0x44, 0xc7	HII_DATABASE_PROTOCOL.GetKeyboardLayout – GetKeyboardLayout() returns EFI_SUCCESS with valid inputs.	Call GetKeyboardLayout() with valid parameters. The return status should be EFI_SUCCESS.

27.4.10 SetKeyboardLayout()

Number	GUID	Assertion	Test Description
5.18.4.10.1	0xad8c6cdc, 0xc749, 0x42e6, 0x88, 0xf7, 0x73, 0x44, 0x7c, 0x38, 0x9e, 0x4d	`HII_DATABASE_PROTOCOL.SetKeyboardLayout - SetKeyboardLayout()` returns `EFI_INVALID_PARAMETER` with `KeyGuid` set to be `NULL`.	Call `SetKeyboardLayout()` with `KeyGuid` being `NULL`. The return status should be `EFI_INVALID_PARAMETER`.
5.18.4.10.2	0x07018fe1, 0xdde0, 0x449b, 0xa5, 0xe2, 0xb1, 0x7a, 0xb5, 0x68, 0x7c, 0x97	`HII_DATABASE_PROTOCOL.SetKeyboardLayout - SetKeyboardLayout()` returns `EFI_NOT_FOUND` with the referenced keyboard layout not found.	Call `SetKeyboardLayout()` with `KeyGuid` which can't be found in database. The return status should be `EFI_NOT_FOUND`.
5.18.4.10.3	0xe7a3dffa, 0x4cca, 0x4402, 0x8f, 0xf1, 0xe3, 0xf3, 0x16, 0xf5, 0x45, 0x1f	`HII_DATABASE_PROTOCOL.SetKeyboardLayout - SetKeyboardLayout()` returns `EFI_SUCCESS` with valid inputs.	Call `SetKeyboardLayout()` with valid parameters. The return status should be `EFI_SUCCESS`.

27.4.11 GetPackageListHandle()

Number	GUID	Assertion	Test Description
5.18.4.11.1	0x373b128d, 0x2216, 0x415b, 0xbb, 0xb1, 0x99, 0x0e, 0xe3, 0x79, 0xf2, 0x85	HII_DATABASE_PROTOCOL.GetPackageListHandle - GetPackageListHandle() returns EFI_INVALID_PARAMETER with DriverHandle been NULL.	Call GetPackageListHandle () with DriverHandle being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.4.11.2	0xb50cffb8, 0x7b74, 0x4b93, 0xb4, 0x87, 0xb3, 0x39, 0xf4, 0x7e, 0xa6, 0x25	HII_DATABASE_PROTOCOL.GetPackageListHandle - GetPackageListHandle () returns EFI_INVALID_PARAMETER with a PackageListHandle which has been removed.	Call GetPackageListHandle () with a PackageListHandle which has been removed. The return status should be EFI_INVALID_PARAMETER.
5.18.4.11.3	0x27a38687, 0x398a, 0x4d65, 0xab, 0x7b, 0x4d, 0xf2, 0xd1, 0x1f, 0x21, 0xa0	HII_DATABASE_PROTOCOL.GetPackageListHandle - GetPackageListHandle() returns EFI_INVALID_PARAMETER with an invalid PackageListHandle.	Call GetPackageListHandle () with an invalid PackageListHandle. The return status should be EFI_INVALID_PARAMETER .
5.18.4.11.4	0x2bc2dae8, 0x2692, 0x487a, 0x94, 0x9d, 0xa7, 0x45, 0x08, 0x82, 0x65, 0x11	HII_DATABASE_PROTOCOL. GetPackageListHandle - GetPackageListHandle() returns EFI_INVALID_PARAMETER with PackageListHandle being NULL .	Call GetPackageListHandle () with PackageListHandle being NULL. The return status should be EFI_INVALID_PARAMETER.
5.18.4.11.5	0xa81329db, 0xcc91, 0x491c, 0xb1, 0x2a, 0x44, 0x0d, 0xf7, 0xed, 0x77, 0xc6	HII_DATABASE_PROTOCOL. GetPackageListHandle - GetPackageListHandle() returns EFI_SUCCESS with valid inputs.	Call GetPackageListHandle () with valid parameters. The return status should be EFI_SUCCESS.

27.5 EFI_HII_CONFIG_ROUTING_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_HII_CONFIG_ROUTING_PROTOCOL Section.

27.5.1 ExtractConfig()

Number	GUID	Assertion	Test Description
5.18.5.1.1	0x04697ed6, 0xcb4e, 0x4e02, 0xbb, 0x8e, 0x9b, 0x76, 0x0b, 0x90, 0xe2, 0xcd	HII_CONFIG_ROUTING_PROTOCOL.ExtractConfig - ExtractConfig() returns EFI_INVALID_PARAMETER with Request been NULL.	Call ExtractConfig() with valid parameters except Request being NULL, The return status should be EFI_INVALID_PARAMETER.
5.18.5.1.2	0x4a1e3525, 0x5247, 0x40dc, 0x93, 0xf7, 0x81, 0x30, 0x6a, 0xce, 0x20, 0xb5	HII_CONFIG_ROUTING_PROTOCOL.ExtractConfig - ExtractConfig() returns EFI_INVALID_PARAMETER with Progress been NULL.	Call ExtractConfig() with valid parameters except Progress being NULL, The return status should be EFI_INVALID_PARAMETER.
5.18.5.1.3	0x05b967d0, 0xe19d, 0x46d8, 0x87, 0xd8, 0x7d, 0x29, 0x65, 0x53, 0x61, 0xc7	HII_CONFIG_ROUTING_PROTOCOL.ExtractConfig - ExtractConfig() returns EFI_INVALID_PARAMETER with Results been NULL.	Call ExtractConfig() with valid parameters except Results being NULL, The return status should be EFI_INVALID_PARAMETER.
5.18.5.1.4	0xee200b58, 0x3714, 0x4cb6, 0x91, 0xc6, 0x31, 0xbe, 0xbd, 0xf4, 0x64, 0x96	HII_CONFIG_ROUTING_PROTOCOL.ExtractConfig - ExtractConfig() returns EFI_NOT_FOUND if Routing data doesn't match any known driver.	Call ExtractConfig() with an invalid Request. The ConfigHdr of Request can't be found in current system. The return status should be EFI_NOT_FOUND. Progress should be set to the "G" in the "GUID" of the routing header that doesn't match.
5.18.5.1.5	0xa18aebb6, 0x140f, 0x454f, 0x8f, 0xe5, 0x34, 0xdd, 0x38, 0xd8, 0xb0, 0xf0	HII_CONFIG_ROUTING_PROTOCOL.ExtractConfig - ExtractConfig() returns EFI_INVALID_PARAMETER if name in Request can't match any known driver.	Call ExtractConfig() with an invalid Request. The name in Request can't be found in current system. The return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.18.5.1.6	0x67adfcdd, 0xda46, 0x4eb8, 0x82, 0x9d, 0xa4, 0x92, 0x8c, 0x10, 0xba, 0x68	HII_CONFIG_ROUTING_PROTOCOL.ExtractConfig - ExtractConfig() returns EFI_SUCCESS with valid parameter and Progress points to the Request's NULL terminator.	Call ExtractConfig() with valid parameters. The return status should be EFI_SUCCESS and Progress points to the Request's NULL terminator.
5.18.5.1.7	0xf91ef5f3, 0xe0c6, 0x4aca, 0xa0, 0xd0, 0x5, 0xf9, 0xb1, 0x6a, 0x13, 0xbd	HII_CONFIG_ROUTING_PROTOCOL.ExtractConfig - ExtractConfig() returns EFI_SUCCESS & Check if Results is in <MultiConfigAltResp> format	1.Call ExtractConfig() with valid parameters. 2.Check if Results is in <MultiConfigAltResp> format. The return status should be EFI_SUCCESS

27.5.2 ExportConfig()

Number	GUID	Assertion	Test Description
5.18.5.2.1	0x81f9658b, 0xbae2, 0x4e08, 0x87, 0xe3, 0x75, 0xe4, 0xe1, 0x47, 0x13, 0xba	HII_CONFIG_ROUTING_PROTOCOL.ExportConfig - ExportConfig() returns EFI_INVALID_PARAMETER with Request been NULL.	Call ExportConfig() with Request being NULL, The return status should be EFI_INVALID_PARAMETER.
5.18.5.2.2	0xe23425ee, 0xaa38, 0x4074, 0xa1, 0xaa, 0xad, 0x5d, 0x98, 0x5a, 0x34, 0xe4	HII_CONFIG_ROUTING_PROTOCOL.ExportConfig - ExportConfig () returns EFI_SUCCESS with valid parameter.	Call ExportConfig() with valid parameter, The return status should be EFI_SUCCESS.

27.5.3 RouteConfig()

Number	GUID	Assertion	Test Description
5.18.5.3.1	0x3a5c09d6, 0x0532, 0x4b4d, 0x87, 0xc8, 0x5e, 0x20, 0x33, 0x78, 0xbc, 0x3f	**HII_CONFIG_ROUTING_PROTOCOL.RouteConfig – RouteConfig()** returns **EFI_INVALID_PARAMETER** with **Configuration** been **NULL**.	Call **RouteConfig()** with **Configuration** being **NULL**, The return status should be **EFI_INVALID_PARAMETER**.
5.18.5.3.2	0x18cdf3f8, 0xf9e6, 0x4128, 0xa4, 0xa6, 0x88, 0xea, 0x88, 0x5d, 0x59, 0x7c	**HII_CONFIG_ROUTING_PROTOCOL.RouteConfig – RouteConfig()** returns **EFI_NOT_FOUND** if Routing data was not found.	Call **RouteConfig()** with an invalid **Configuration**. The ConfigHdr of **Configuration** can't be found in current system. The return status should be **EFI_NOT_FOUND**.
5.18.5.3.3	0x20833aeb, 0x9ff1, 0x4315, 0xb1, 0x0f, 0x31, 0x7c, 0x7b, 0x92, 0x45, 0x21	**HII_CONFIG_ROUTING_PROTOCOL.RouteConfig – RouteConfig()** returns **EFI_SUCCESS** with valid parameter and **Progress** points to the **Configuration**'s **NULL** terminator.	Call **RouteConfig()** with valid parameters. The return status should be **EFI_SUCCESS** and **Progress** points to the **Configuration**'s **NULL** terminator.

27.5.4 BlockToConfig()

Number	GUID	Assertion	Test Description
5.18.5.4.1	0xb1dfee09, 0x73e5, 0x4659, 0x9a, 0xc6, 0x59, 0x46, 0xc1, 0xa1, 0x53, 0xcb	HII_CONFIG_ROUTING_PROTOCOL.BlockToConfig - BlockToConfig() returns EFI_INVALID_PARAMETER with ConfigRequest been NULL.	Call BlockToConfig() with valid parameters except ConfigRequest being NULL, The return status should be EFI_INVALID_PARAMETER.
5.18.5.4.2	0x544bf56b, 0x3bdc, 0x46d5, 0x88, 0x4f, 0x19, 0xde, 0x76, 0x19, 0xef, 0xd3	HII_CONFIG_ROUTING_PROTOCOL.BlockToConfig - BlockToConfig() returns EFI_INVALID_PARAMETER with Block been NULL.	Call BlockToConfig() with valid parameters except Block being NULL, The return status should be EFI_INVALID_PARAMETER.
5.18.5.4.3	0xee6f8fd8, 0x951d, 0x4976, 0x86, 0xf0, 0xae, 0x7f, 0x5c, 0x69, 0x5b, 0x40	HII_CONFIG_ROUTING_PROTOCOL.BlockToConfig - BlockToConfig() returns EFI_INVALID_PARAMETER with <ConfigElement> in ConfigRequest being a <NvConfig>.	Call BlockToConfig() with valid parameters except <ConfigElement> in ConfigRequest being a <NvConfig>, The return status should be EFI_INVALID_PARAMETER and Progress points to '&' of the first non-<BlockName>.
5.18.5.4.4	0xd38890ec, 0xd43e, 0x4e28, 0xab, 0x47, 0xef, 0x67, 0xeb, 0x2d, 0x3d, 0x92	HII_CONFIG_ROUTING_PROTOCOL.BlockToConfig - BlockToConfig() returns EFI_DEVICE_ERROR if Block is not large enough.	Call BlockToConfig() with with valid parameters except Block is not large enough. The return status should be EFI_DEVICE_ERROR.
5.18.5.4.5	0x8b1b960c, 0xda67, 0x423c, 0x85, 0x31, 0x76, 0x28, 0x0d, 0xb8, 0x2a, 0xc1	HII_CONFIG_ROUTING_PROTOCOL.BlockToConfig - BlockToConfig() returns EFI_SUCCESS with valid parameter and Progress points to the ConfigRequest's NULL terminator.	Call BlockToConfig() with valid parameters. The return status should be EFI_SUCCESS and Progress points to the ConfigRequest's NULL terminator.

27.5.5 ConfigToBlock ()

Number	GUID	Assertion	Test Description
5.18.5.5.1	0x76ab8420, 0x7c61, 0x4ebc, 0x8b, 0x5b, 0x62, 0xa3, 0x35, 0x64, 0x6f, 0x8f	HII_CONFIG_ROUTING_PROTOCOL.ConfigToBlock - ConfigToBlock() returns EFI_INVALID_PARAMETER with ConfigResp been NULL.	Call ConfigToBlock() with valid parameters except ConfigResp being NULL, The return status should be EFI_INVALID_PARAMETER.
5.18.5.5.2	0xcc284047, 0x45d6, 0x4fec, 0x88, 0x50, 0x70, 0x3f, 0x45, 0x22, 0x01, 0xdc	HII_CONFIG_ROUTING_PROTOCOL.ConfigToBlock - ConfigToBlock() returns EFI_INVALID_PARAMETER with Block been NULL. Progress should point to the first character of ConfigResp.	Call ConfigToBlock() with valid parameters except Block being NULL, The return status should be EFI_INVALID_PARAMETER and Progress should point to the first character of ConfigResp.
5.18.5.5.3	0x2d30da76, 0x9ec7, 0x480e, 0xb9, 0xe9, 0x6d, 0x50, 0x0d, 0x89, 0x21, 0xad	HII_CONFIG_ROUTING_PROTOCOL.ConfigToBlock - ConfigToBlock() returns EFI_INVALID_PARAMETER with <RequestElement> in ConfigResp being a <Lable>.	Call ConfigToBlock() with valid parameters except < RequestElement > in ConfigResp being a <Lable>. The return status should be EFI_INVALID_PARAMETER and Progress points to '&' of the first non-<BlockName>.
5.18.5.5.4	0xa5b33ea4, 0x767b, 0x489a, 0xb3, 0x7b, 0xf9, 0xef, 0xfd, 0x62, 0xbc, 0x7b	HII_CONFIG_ROUTING_PROTOCOL.ConfigToBlock - ConfigToBlock() returns EFI_DEVICE_ERROR if Block is not large enough.	Call ConfigToBlock() with valid parameters except Block is not large enough. The return status should be EFI_DEVICE_ERROR.
5.18.5.5.5	0x59b759ff, 0x6c84, 0x407a, 0x9e, 0x24, 0x71, 0xe0, 0x65, 0x2d, 0xe3, 0x30	HII_CONFIG_ROUTING_PROTOCOL.ConfigToBlock - ConfigToBlock() returns EFI_SUCCESS with valid parameter and Progress points to the ConfigResp's NULL terminator.	Call ConfigToBlock() with valid parameters. The return status should be EFI_SUCCESS and Progress points to the ConfigResp's NULL terminator.

27.5.6 GetAltCfg ()

Number	GUID	Assertion	Test Description
5.18.5.6.1	0x1ff2326a, 0x8e88, 0x45db, 0x94, 0x81, 0x02, 0x83, 0x80, 0x20, 0xad, 0x02	HII_CONFIG_ROUTING_PROTOCOL.GetAltCfg - GetAltCfg() returns EFI_INVALID_PARAMETER with ConfigResp been NULL.	Call GetAltCfg() with valid parameters except ConfigResp being NULL, The return status should be EFI_INVALID_PARAMETER.
5.18.5.6.2	0xb9b88d34, 0x7479, 0x4807, 0xa4, 0xbf, 0x90, 0x35, 0x87, 0x0a, 0x3c, 0x1a	HII_CONFIG_ROUTING_PROTOCOL.GetAltCfg - GetAltCfg() returns EFI_INVALID_PARAMETER with AltCfgResp been NULL.	Call GetAltCfg() with valid parameters except AltCfgResp being NULL, The return status should be EFI_INVALID_PARAMETER.
5.18.5.6.3	0xfe1e4232, 0x8819, 0x4f52, 0xac, 0xaa, 0xb2, 0x02, 0x72, 0x86, 0xc8, 0xe4	HII_CONFIG_ROUTING_PROTOCOL.GetAltCfg - GetAltCfg() returns EFI_SUCCESS with NULL Guid, Name, DevicePath, except a valid AltCfgId.	Call GetAltCfg() with NULL Guid, Name, DevicePath, except a valid AltCfgId. The return status should be EFI_SUCCESS and AltCfgResp should points to retrieved data.
5.18.5.6.4	0xdf88e78e, 0x8f4d, 0x4027, 0xbb, 0xcd, 0xae, 0x10, 0x68, 0x58, 0xb6, 0x03	HII_CONFIG_ROUTING_PROTOCOL.GetAltCfg - GetAltCfg() returns EFI_SUCCESS with NULL Name, DevicePath, except a valid Guid, AltCfgId.	Call GetAltCfg() with NULL Name, DevicePath, except a valid Guid, AltCfgId. The return status should be EFI_SUCCESS and AltCfgResp should points to retrieved data.
5.18.5.6.5	0x2b56a57a, 0xd906, 0x416c, 0x89, 0x76, 0x43, 0x5f, 0xc7, 0x1c, 0xb7, 0x73	HII_CONFIG_ROUTING_PROTOCOL.GetAltCfg - GetAltCfg() returns EFI_SUCCESS with NULL Guid, DevicePath, except a valid Name, AltCfgId.	Call GetAltCfg() with NULL Guid, DevicePath, except a valid Name, AltCfgId. The return status should be EFI_SUCCESS and AltCfgResp should points to retrieved data.

Number	GUID	Assertion	Test Description
5.18.5.6.6	0x17c575b3, 0x051f, 0x41eb, 0x89, 0xd1, 0x79, 0xb5, 0x8b, 0x0c, 0x92, 0x3c	HII_CONFIG_ROUTING_PROTOCOL.GetAltCfg - GetAltCfg() returns EFI_SUCCESS with NULL DevicePath, except a valid Guid, Name, AltCfgId.	Call GetAltCfg() with NULL DevicePath, except a valid Guid,Name, AltCfgId. The return status should be EFI_SUCCESS and AltCfgResp should points to retrieved data.
5.18.5.6.7	0xb948d2f8, 0x5c45, 0x4b10, 0x97, 0xb4, 0x95, 0x96, 0x97, 0x98, 0xe5, 0x8b	HII_CONFIG_ROUTING_PROTOCOL.GetAltCfg - GetAltCfg() returns EFI_SUCCESS returns EFI_SUCCESS with NULL DevicePath, AltCfgId, except a valid Guid, Name.	Call GetAltCfg() with NULL DevicePath, AltCfgId, except a valid Guid, Name. The return status should be EFI_SUCCESS and AltCfgResp should points to retrieved data.
5.18.5.6.8	0xf732d246, 0x9fa5, 0x4ed3, 0x88, 0x95, 0x28, 0x63, 0xba, 0xf4, 0x68, 0x5d	HII_CONFIG_ROUTING_PROTOCOL.GetAltCfg - GetAltCfg() returns EFI_SUCCESS with valid Name	1.Call GetAltCfg() with NULL GUID DevicePath, AltCfgId, except a valid Name. 2. The return status should be EFI_SUCCESS and AltCfgResp should points to right data.

27.6 EFI_HII_CONFIG_ACCESS_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_HII_CONFIG_ACCESS_PROTOCOL Section.

27.6.1 ExtractConfig()

Number	GUID	Assertion	Test Description
5.18.6.1.1	0xa7173eb5, 0xf76a, 0x4ea1, 0x95, 0x0d, 0x14, 0x91, 0x1e, 0x49, 0x86, 0xc1	`HII_CONFIG_ACCESS_PROTOCOL.ExtractConfig - ExtractConfig()` returns `EFI_INVALID_PARAMETER` with `Request` been <MultiConfigRequest> format.	Call `ExtractConfig()` with valid parameters except with `Request` being <MultiConfigRequest> format., The return status should be `EFI_INVALID_PARAMETER`. And Progress should point to the most recent '&' before the error or beginning of the string.
5.18.6.1.2	0xfa5973e2, 0x0d05, 0x44c2, 0xaf, 0x2d, 0x1b, 0x68, 0x33, 0x42, 0x6d, 0x76	`HII_CONFIG_ACCESS_PROTOCOL.ExtractConfig - ExtractConfig()` returns `EFI_INVALID_PARAMETER` with `Progress` been `NULL.`	Call `ExtractConfig()` with valid parameters except `Progress` being `NULL`, The return status should be `EFI_INVALID_PARAMETER`.
5.18.6.1.3	0x6f6d1dd, 0x49b8, 0x488a, 0xa7, 0x75, 0xde, 0xbc, 0xc7, 0x60, 0xfd, 0x28	`HII_CONFIG_ACCESS_PROTOCOL.ExtractConfig - ExtractConfig()` returns `EFI_INVALID_PARAMETER` with `Results` been `NULL.`	Call `ExtractConfig()` with valid parameters except `Results` being `NULL`, The return status should be `EFI_INVALID_PARAMETER`.
5.18.6.1.4	0x28652613, 0x6bf4, 0x4f42, 0xab, 0xe2, 0x84, 0x4f, 0x2f, 0x77, 0xec, 0x2f	`HII_CONFIG_ACCESS_PROTOCOL.ExtractConfig - ExtractConfig()` returns `EFI_NOT_FOUND` if Routing data doesn't match any known driver or `EFI_INVALID_PARAMETER` if there is an unknown name in `Request.`	Call `ExtractConfig()` with an invalid `Request`. The ConfigHdr of `Request` can't be found in current system. The return status should be `EFI_NOT_FOUND`. `Progress` should point to the error reason. If an unknown name in the `Request`, the return status should be `EFI_INVALID_PARAMETER` and `Progress` should point to the '&' before the name in question.

Number	GUID	Assertion	Test Description
5.18.6.1.5	0x24dcf8bf, 0xbfbf, 0x4588, 0xba, 0x0f, 0x77, 0x1e, 0x24, 0x4e, 0x3e, 0x08	**HII_CONFIG_ACCESS_PROTOCOL.ExtractConfig - ExtractConfig()** returns **EFI_SUCCESS** with valid parameters and and **Progress** points to the **Request**'s **NULL** terminator.	Call **ExtractConfig()** with valid parameters. The return status should be **EFI_SUCCESS** and the **Progress** should point to **Request**'s **NULL** terminator.
5.18.6.1.6	0x961a5268, 0x1998, 0x4a7e, 0x9d, 0x9d, 0xce, 0xdc, 0x67, 0xfb, 0xcc, 0x77	**HII_CONFIG_ACCESS_PROTOCOL.ExtractConfig - ExtractConfig()** returns **EFI_SUCCESS** with valid parameter except **Request** been **NULL.**	Call **ExtractConfig()** with valid parameters except **Request** been **NULL**. The return status should be **EFI_SUCCESS**.
5.18.6.1.7	0xab163674, 0x6c27, 0x4169, 0xa6, 0xa9, 0xe1, 0x9c, 0x88, 0x14, 0x94, 0x96	**HII_CONFIG_ACCESS_PROTOCOL.ExtractConfig - ExtractConfig()** returns **EFI_SUCCESS**. Check if Results is in **<MultiConfigAltResp>** format	Call **ExtractConfig()** with valid parameters. The return status should be **EFI_SUCCESS** and Check if Results is in **<MultiConfigAltResp>** format.

27.6.2 RouteConfig()

Number	GUID	Assertion	Test Description
5.18.6.2.1	0xebba197a, 0x467f, 0x4736, 0x92, 0xf2, 0x11, 0xb1, 0x91, 0x2e, 0xe9, 0x90	HII_CONFIG_ACCESS_PROTOCOL.RouteConfig - RouteConfig() returns EFI_INVALID_PARAMETER with Configuration been NULL.	Call RouteConfig() with valid parameters except with Configuration being NULL, The return status should be EFI_INVALID_PARAMETER.
5.18.6.2.2	0x341fe3e0, 0xf688, 0x45f2, 0x91, 0x56, 0xc7, 0xae, 0x9f, 0x2c, 0xcb, 0xb0	HII_CONFIG_ACCESS_PROTOCOL. RouteConfig - RouteConfig() returns EFI_INVALID_PARAMETER with Progress been NULL.	Call RouteConfig() with valid parameters except Progress being NULL, The return status should be EFI_INVALID_PARAMETER.
5.18.6.2.3	0x1f99ebc8, 0x0253, 0x455f, 0x88, 0xac, 0x9e, 0x2b, 0xa6, 0xdc, 0xd7, 0x29	HII_CONFIG_ACCESS_PROTOCOL. RouteConfig - RouteConfig() returns EFI_NOT_FOUND if no target was found with the routing data.	Call RouteConfig() with no found target for the routing data. The return status should be EFI_NOT_FOUND.
5.18.6.2.4	0x603e52f0, 0x2ce3, 0x4e7a, 0xa7, 0x2e, 0xdf, 0x8c, 0xa3, 0xfd, 0xb2, 0x0d	HII_CONFIG_ACCESS_PROTOCOL. RouteConfig - RouteConfig() returns EFI_SUCCESS with valid parameters and and Progress points to the Configuration's NULL terminator.	Call RouteConfig() with valid parameters. The return status should be EFI_SUCCESS and the Progress should point to Configuration's NULL terminator.

27.7 EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL Section.

27.7.1 SetData()

Number	GUID	Assertion	Test Description
5.18.7.1.1	0xf046a19c, 0xffc1, 0x4fd9, 0x9d, 0x73, 0x92, 0x4f, 0x8c, 0x43, 0xcf, 0xfb	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. SetData() - SetData() returns EFI_NOT_FOUND when an element of the KeywordString was not found. Progress points to the most recent '&' before the first failing string element and ProgressErr should be KEYWORD_HANDLER_KEYWORD_NOT_FOUND.	1. Call SetData() when an element of the KeywordString was not found, the return status should be EFI_NOT_FOUND. Progress points to the most recent '&' before the first failing string element and ProgressErr should be KEYWORD_HANDLER_KEYWORD_NOT_FOUND.
5.18.7.1.2	0x553c956c, 0x78c1, 0x44d4, 0x81, 0x8e, 0x98, 0xdf, 0xd2, 0x25, 0x8, 0xe5	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. SetData() - GetData() returns EFI_SUCCESS, ProgressErr should be KEYWORD_HANDLER_ NO_ERROR. Progress points to the string's NULL terminator.	2. Check the system with GetData(), the storage associated with the earlier keywords is not modified when an EFI_NOT_FOUND error is generated during processing the second or later keyword element.
5.18.7.1.3	0xe334ff21, 0x4005, 0x449a, 0x83, 0x1, 0x97, 0x44, 0xc1, 0xb0, 0xaf, 0xd5	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. SetData() - SetData() returns EFI_SUCCESS when an element of the KeywordString was found. Progress points to the string's NULL terminator and ProgressErr should be KEYWORD_HANDLER_ NO_ERROR.	1. Call SetData() when an element of the KeywordString was found, the return status should be EFI_SUCCESS. Progress points to the string's NULL terminator and ProgressErr should be KEYWORD_HANDLER_ NO_ERROR.

Number	GUID	Assertion	Test Description
5.18.7.1.4	0x8a4618b3, 0xa012, 0x40c4, 0xba, 0x6, 0xa, 0x93, 0x79, 0xb4, 0x64, 0x58	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. SetData() - GetData() returns EFI_SUCCESS, ProgressErr should be KEYWORD_HANDLER_ NO_ERROR. Progress points to the string's NULL terminator.	2. Check the system with GetData(), the storage associated with the earlier keywords should be saved correctly.
5.18.7.1.5	0xfe4f680c, 0xcbe, 0x4f85, 0xb3, 0x20, 0x5e, 0xcc, 0x9d, 0xce, 0xc5, 0x88	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. SetData() - SetData() returns EFI_INVALID_PARAMETER when KeywordString was NULL.	1. Call SetData() when KeywordString was found, the return status should be EFI_INVALID_PARAMETER.
5.18.7.1.6	0xe7966ef2, 0x941e, 0x4a59, 0x8e, 0x15, 0x2f, 0xde, 0x41, 0x9d, 0xfc, 0x91	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. SetData() - SetData() returns EFI_INVALID_PARAMETER when parsing of the KeywordString resulted in an error and Progress points to the most recent '&' before the first failing string element.	1. Call SetData() when parsing of the KeywordString resulted in an error, the return status should be EFI_INVALID_PARAMETER. Progress should point to the most recent '&' before the first failing string element.
5.18.7.1.7	0x1eff122d, 0xa263, 0x43bd, 0x94, 0xfc, 0x82, 0xb, 0x8b, 0xc9, 0xfa, 0x7c	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. SetData() - SetData() returns EFI_NOT_FOUND when an element of the KeywordString was not found and Progress points to the most recent '&' before the first failing string element.	1. Call SetData() when an element of the KeywordString was not found, the return status should be EFI_NOT_FOUND. Progress should point to the most recent '&' before the first failing string element.

Number	GUID	Assertion	Test Description
5.18.7.1.8	0x4bd58084, 0xb158, 0x43fe, 0xbb, 0x87, 0x31, 0x8f, 0xb2, 0x3f, 0x7a, 0xe9	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. SetData() - SetData() returns EFI_ACCESS_DENIED when the ReadOnly element is written and Progress points to the most recent '&' before the first failing string element.	1. Call SetData() when the ReadOnly element is written, the return status should be EFI_ACCESS_DENIED. Progress should point to the most recent '&' before the first failing string element.

27.7.2 GetData()

Number	GUID	Assertion	Test Description
5.18.7.2.1	0x852b267e, 0xcbe, 0x4bd6, 0x85, 0x4d, 0x3b, 0xbd, 0xf0, 0xa0, 0xc, 0x49	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() returns EFI_SUCCESS when KeywordString and NameSpaceId are NULL.	1. Call GetData()when KeywordString and NameSpaceId are NULL, the return status should be EFI_SUCCESS. ProgressErr should be KEYWORD_HANDLER_NO_ERROR.
5.18.7.2.2	0x247b91db, 0xf60b, 0x457f, 0xb9, 0x10, 0xb3, 0xc3, 0x30, 0xa8, 0xaf, 0x88	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() should output the correct result as expected format.	2. The preinstalled Str should be included in the Results outputted from the GetData().
5.18.7.2.3	0xf57e9ce0, 0x827a, 0x4d35, 0x89, 0xb8, 0xde, 0x24, 0x57, 0xe7, 0x94, 0xfb	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() returns EFI_SUCCESS when KeywordString is NULL and NameSpaceId is one valid expression.	1. Call GetData()when KeywordString is NULL and NameSpaceId is one valid expression, the return status should be EFI_SUCCESS. ProgressErr should be KEYWORD_HANDLER_NO_ERROR.
5.18.7.2.4	0x170ab626, 0x648c, 0x4088, 0x8b, 0x5d, 0xf8, 0xf2, 0x9d, 0x65, 0xaf, 0xba	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() should output the correct result as expected format.	2. The preinstalled Str should be included in the Results outputted from the GetData().
5.18.7.2.5	0x60bcfe65, 0xe73a, 0x46dd, 0xa9, 0x42, 0x22, 0xb4, 0xeb, 0x30, 0xb8, 0x7c	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() returns EFI_SUCCESS when KeywordString is the valid expression (with PathHdr) and NameSpaceId is one valid expression.	1. Call GetData() when KeywordString is the valid expression (with PathHdr) and NameSpaceId is one valid expression, the return status should be EFI_SUCCESS. ProgressErr should be KEYWORD_HANDLER_NO_ERROR and Progress points to the string's NULL terminator.

Number	GUID	Assertion	Test Description
5.18.7.2.6	0x7cc0b84, 0x4128, 0x4c66, 0x91, 0x90, 0x76, 0x15, 0x81, 0xb, 0x95, 0x9d	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() returns EFI_SUCCESS when KeywordString is the valid expression (without PathHdr) and NameSpaceId is one valid expression.	2. Call GetData() when KeywordString is the valid expression (without PathHdr) and NameSpaceId is one valid expression, the return status should be EFI_SUCCESS. ProgressErr should be KEYWORD_HANDLER_NO_ERROR and Progress points to the string's NULL terminator.
5.18.7.2.7	0x6114b15, 0xab62, 0x40f5, 0x86, 0xf6, 0x21, 0xd1, 0x81, 0x2b, 0x7f, 0x6c	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() should output the correct result as expected format.	3. The Results outputted with PathHdr should be included in the Results outputted without PathHdr
5.18.7.2.8	0x378ef819, 0x29ee, 0x4875, 0x8c, 0xb2, 0x94, 0x6a, 0x77, 0xb1, 0x48, 0x73	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() returns EFI_INVALID_PARAMETER when Progress, ProgressErr, or Resuts is NULL.	1. Call GetData() when Progress, ProgressErr, or Resuts is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.7.2.9	0xb90fe257, 0xf693, 0x4c3e, 0x89, 0x59, 0x14, 0xb, 0xcf, 0x44, 0x7b, 0x5d	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() returns EFI_INVALID_PARAMETER when Parsing of the KeywordString resulted in an error.	1. Call GetData() when Parsing of the KeywordString resulted in an error, the return status should be EFI_INVALID_PARAMETER. Progress should point to the most recent '&' before the first failing string element and ProgressErr should be KEYWORD_HANDLER_MALFORMED_STRING.
5.18.7.2.10	0x138298f2, 0x7b86, 0x49b7, 0x9c, 0xa7, 0x6d, 0x69, 0xbe, 0x8b, 0x52, 0xfd	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() returns EFI_NOT_FOUND when an element of the KeywordString was not found.	1. Call GetData() when an element of the KeywordString was not found, the return status should be EFI_NOT_FOUND. Progress should point to the most recent '&' before the first failing string element and ProgressErr should be KEYWORD_HANDLER_KEYWORD_NOT_FOUND.

Number	GUID	Assertion	Test Description
5.18.7.2.11	0x48dab3bf, 0xb3dc, 0x4960, 0xa6, 0xf8, 0xb5, 0x1c, 0xd3, 0xfa, 0xfa, 0xe0	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() returns EFI_NOT_FOUND when the NamespaceId specified was not found.	1. Call GetData() when the NamespaceId specified was not found, the return status should be EFI_NOT_FOUND. ProgressErr should be KEYWORD_HANDLER_KEYWORD_NOT_FOUND.
5.18.7.2.12	0xab69961e, 0xd77d, 0x4781, 0x8e, 0xe5, 0xf9, 0x13, 0x55, 0xc7, 0xce, 0x91	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() returns EFI_NOT_FOUND when an element of the KeywordString was not found.	1. Call GetData() when an element of the KeywordString was not found, the return status should be EFI_NOT_FOUND. Progress should point to the most recent '&' before the first failing string element and ProgressErr should be KEYWORD_HANDLER_KEYWORD_NOT_FOUND.
5.18.7.2.13	0xc6b310c5, 0xdddf, 0x4e1d, 0x9d, 0x8c, 0x20, 0x16, 0xe7, 0x66, 0xa6, 0xae	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() should output Results string contains values returned for all keywords processed prior to the keyword generating the error.	2. The returned Results string should contain values for all keywords processed prior to the keyword generating the error.

27.8 EFI_HII_FONT_EX_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_HII_FONT_EX_PROTOCOL Section.

27.8.1 StringToImageEx()

Number	GUID	Assertion	Test Description
5.18.8.1.1	0x81b18c28, 0x7d09, 0x4794, 0xab, 0x4e, 0x92, 0x9b, 0xb7, 0x2f, 0x19, 0x67	EFI_HII_FONT_EX_PROTOCOL . StringToImageEx() - StringToImageEx() returns EFI_INVALID_PARAMETER when String is NULL.	1. Call StringToImageEx() when String is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.8.1.2	0xeba34749, 0x9763, 0x4203, 0x9f, 0xd, 0x26, 0x3a, 0xa4, 0xe9, 0xd6, 0x9a	EFI_HII_FONT_EX_PROTOCOL . StringToImageEx() - StringToImageEx() returns EFI_INVALID_PARAMETER when Blt is NULL.	1. Call StringToImageEx() when Blt is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.8.1.3	0xd6514302, 0x4b34, 0x4bae, 0xa0, 0xcd, 0x37, 0x77, 0xb8, 0x43, 0xc, 0x26	EFI_HII_FONT_EX_PROTOCOL . StringToImageEx() - StringToImageEx() returns EFI_INVALID_PARAMETER with invalid Flags combination.	1. Call StringToImageEx() when Flags is the combination of EFI_HII_OUT_FLAG_CLIP_CLEAN_X and EFI_HII_OUT_FLAG_WRAP, the return status should be EFI_INVALID_PARAMETER.
5.18.8.1.4	0xf711f218, 0x8987, 0x4fa9, 0xb4, 0xb6, 0x64, 0x1e, 0xc1, 0x76, 0xe1, 0xc8	EFI_HII_FONT_EX_PROTOCOL . StringToImageEx() - StringToImageEx() returns EFI_INVALID_PARAMETER with invalid Flags combination.	1. Call StringToImageEx() when Flags is EFI_HII_OUT_FLAG_CLIP_CLEAN_X without EFI_HII_OUT_FLAG_CLIP, the return status should be EFI_INVALID_PARAMETER.
5.18.8.1.5	0x4dd0210d, 0x87b1, 0x4352, 0xa6, 0x16, 0x57, 0x91, 0x78, 0x73, 0xe0, 0xa0	EFI_HII_FONT_EX_PROTOCOL . StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with valid Flags combination.	1. Call StringToImageEx() with the valid Flags combination and use EFI_GRAPUICS_OUTPUT_BLT_PIXEL structure in EFI_IMAGE_OUTPUT structure, the return status should be EFI_SUCCESS.
5.18.8.1.6	0x2af74a94, 0xed7, 0x4b68, 0x9c, 0xdd, 0xfa, 0xdf, 0xfe, 0x6, 0x68, 0x1f	EFI_HII_FONT_EX_PROTOCOL . StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with valid Flags combination.	1. Call StringToImageEx() with the valid Flags combination and use EFI_GRAPUICS_OUTPUT_PROTOCOL in EFI_IMAGE_OUTPUT structure, the return status should be EFI_SUCCESS.
5.18.8.1.7	0x7047fe55, 0x6c8c, 0x4062, 0x8a, 0x24, 0x26, 0xb5, 0x33, 0x88, 0x62, 0x81	EFI_HII_FONT_EX_PROTOCOL . StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with valid parameters for all ASCII visible characters. Each image must equal to sys default glyph.	1. Call StringToImageEx() with the valid parameters and StringInfo is NULL. Compare image output with system default font glyph image.

Number	GUID	Assertion	Test Description
5.18.8.1.8	0xf09da704, 0x352, 0x4afa, 0x90, 0x8f, 0x83, 0x73, 0xf2, 0xe9, 0xe6, 0x2c	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with valid parameters for all ASCII visible characters. Each image must equal to the specific font glyph.	1. Register a specific font package. Call StringToImageEx() with the valid parameters and StringInfo is the specific font. Compare image output with specific font glyph image registered.
5.18.8.1.9	0xbee39111, 0x1e5b, 0x4574, 0xae, 0xeb, 0x2, 0xdd, 0xaa, 0x17, 0x42, 0xbf	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_DIRECT_TO_SCREEN.	1. Call StringToImageEx() with EFI_HII_DIRECT_TO_SCREEN. For the final row, the RowInfoArray.LineHeight and RowInfoArray.BaseLine may describe pixels which are outside the limit specified by Blt.Height (unless EFI_HII_OUT_FLAG_CLIP_CLEAN_Y is specified) even though those pixels were not drawn. 2. The return status should be EFI_SUCCESS.
5.18.8.1.10	0x2c36e6b5, 0x983f, 0x4e05, 0x90, 0xdd, 0xfa, 0x79, 0xfd, 0xdb, 0x15, 0xcd	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_OUT_FLAG_CLIP | EFI_HII_DIRECT_TO_SCREEN.	1. Call StringToImageEx() with EFI_HII_OUT_FLAG_CLIP | EFI_HII_DIRECT_TO_SCREEN. For the final row, the RowInfoArray.LineHeight and RowInfoArray.BaseLine may describe pixels which are outside the limit specified by Blt.Height (unless EFI_HII_OUT_FLAG_CLIP_CLEAN_Y is specified) even though those pixels were not drawn. 2. The return status should be EFI_SUCCESS.
5.18.8.1.11	0x7dd51e66, 0xf38f, 0x4412, 0xa6, 0xd8, 0x32, 0x37, 0x85, 0xb9, 0x8, 0x31	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_X | EFI_HII_DIRECT_TO_SCREEN.	1. Call StringToImageEx() with EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_X | EFI_HII_DIRECT_TO_SCREEN. If a character’s right-most pixel can’t fit, then it will not be drawn at all. 2. The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.18.8.1.12	0x76805500, 0x3e74, 0x44cb, 0x95, 0x9b, 0x63, 0xf7, 0xb7, 0x78, 0x92, 0x17	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_Y | EFI_HII_DIRECT_TO_SCREEN.	1. Call StringToImageEx() with EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_Y | EFI_HII_DIRECT_TO_SCREEN. If a row's bottom-most pixel exceeds screen Height, then it will not be drawn at all. 2. The return status should be EFI_SUCCESS.
5.18.8.1.13	0xe18566cf, 0x619d, 0x454c, 0x85, 0x6b, 0xe, 0x4e, 0xd3, 0x1c, 0x4a, 0xf1	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_IGNORE_IF_NO_GLYPH | EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and String with line break opportunity.	1. Call StringToImageEx() with EFI_HII_IGNORE_IF_NO_GLYPH | EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and String with line break opportunity (SPACE is a line break opportunity). Check display with wrapper at right place. 2. The return status should be EFI_SUCCESS.
5.18.8.1.14	0xacba2f9a, 0x1052, 0x478d, 0x96, 0x99, 0x78, 0xa1, 0x1e, 0x65, 0x5, 0x5d	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and String without line break opportunity.	1. Call StringToImageEx() with EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and String without line break opportunity. String is designed to display as if EFI_HII_OUT_FLAG_CLIP_CLEAN_X is set. 2. The return status should be EFI_SUCCESS.
5.18.8.1.15	0x82482a71, 0x2a32, 0x4104, 0xb7, 0x32, 0x91, 0xa0, 0x95, 0x81, 0x50, 0x49	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_IGNORE_LINE_BREAK | EFI_HII_DIRECT_TO_SCREEN.	1. Call StringToImageEx() with EFI_HII_IGNORE_LINE_BREAK | EFI_HII_DIRECT_TO_SCREEN. If a row's bottom-most pixel can't fit, then it will not be drawn at all. This flag requires that EFI_HII_OUT_FLAG_CLIP be set. 2. The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.18.8.1.16	0xf1c89a03, 0x5b7a, 0x4d1d, 0xbe, 0x9, 0x5c, 0xf7, 0xe5, 0x67, 0xe, 0x77	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_DIRECT_TO_SCREEN.	1. Register a specific font package. 2. Call StringToImageEx() with EFI_HII_DIRECT_TO_SCREEN. 3. Check EFI_HII_DIRECT_TO_SCREEN only case if Blt is not NULL, then EFI_HII_OUT_FLAG_CLIP is implied. String is designed to display with full line. 4. The return status should be EFI_SUCCESS.
5.18.8.1.17	0x2154d7a2, 0x37e2, 0x43a3, 0xb4, 0xaf, 0xb3, 0x74, 0x8a, 0x6c, 0x54, 0xf0	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_OUT_FLAG_CLIP.	1. Register a specific font package. 2. Call StringToImageEx() with EFI_HII_OUT_FLAG_CLIP. 3. For the final row, the RowInfoArray.LineHeight andRowInfoArray.BaseLine may describe pixels which are outside the limit specified by Blt.Height (unless EFI_HII_OUT_FLAG_CLIP_CLEAN_Y is specified) even though those pixels were not drawn. 4. The return status should be EFI_SUCCESS.
5.18.8.1.18	0x6206dfcf, 0x6fb3, 0x4020, 0xba, 0xf3, 0x74, 0xe, 0xed, 0xac, 0x9c, 0xb2	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_X | EFI_HII_DIRECT_TO_SCREEN.	1. Register a specific font package. 2. Call StringToImageEx() with EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_X | EFI_HII_DIRECT_TO_SCREEN. 3. If a character's right-most pixel can't fit, then it will not be drawn at all. 4. The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.18.8.1.19	0x76bd46eb, 0x56a1, 0x4b66, 0xab, 0x63, 0x2e, 0xf1, 0x69, 0x1a, 0xfd, 0x80	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_Y | EFI_HII_DIRECT_TO_SCREEN.	1. Register a specific font package. 2. Call StringToImageEx() with EFI_HII_OUT_FLAG_CLIP | EFI_HII_OUT_FLAG_CLIP_CLEAN_Y | EFI_HII_DIRECT_TO_SCREEN. 3. If a row's bottom-most pixel exceeds screen Height, then it will not be drawn at all. 4. The return status should be EFI_SUCCESS.
5.18.8.1.20	0x9782016a, 0xcd4c, 0x4d39, 0x91, 0xc3, 0x7e, 0xe3, 0xce, 0xfd, 0xcc, 0x2d	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_IGNORE_IF_NO_GLYPH | EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and String with line break opportunity.	1. Register a specific font package. 2. Call StringToImageEx() with EFI_HII_IGNORE_IF_NO_GLYPH | EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and String with line break opportunity (Space is a line-break). 3. Check if the display is right. 4. The return status should be EFI_SUCCESS.
5.18.8.1.21	0x2833962d, 0x3800, 0x45b3, 0x90, 0xf8, 0xfb, 0xe2, 0xee, 0xc6, 0x6e, 0xd9	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and String without line break opportunity.	1. Register a specific font package. 2. Call StringToImageEx() with EFI_HII_OUT_FLAG_WRAP | EFI_HII_DIRECT_TO_SCREEN and String without line break opportunity. 3. String is designed to display as if EFI_HII_OUT_FLAG_CLIP_CLEAN_X is set. 4. The return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.18.8.1.22	0x12eb38a6, 0xfc, 0x4568, 0xa3, 0x44, 0x75, 0x40, 0xd3, 0x89, 0x88, 0xbe	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_IGNORE_LINE_BREAK | EFI_HII_DIRECT_TO_SCREEN.	1. Register a specific font package. 2. Call StringToImageEx() with EFI_HII_IGNORE_LINE_BREAK | EFI_HII_DIRECT_TO_SCREEN. 3. If a row's bottom-most pixel can't fit, then it will not be drawn at all. This flag requires that EFI_HII_OUT_FLAG_CLIP be set. 4. The return status should be EFI_SUCCESS.
5.18.8.1.23	0x9c9802d4, 0x98e5, 0x46b9, 0xab, 0xc7, 0x66, 0x17, 0xb7, 0x80, 0x40, 0x29	EFI_HII_FONT_EX_PROTOCOL. StringToImageEx() - StringToImageEx() returns EFI_SUCCESS with parameters EFI_HII_OUT_FLAG_TRANSPARENT.	1. Register a specific font package. 2. Call StringToImageEx() with EFI_HII_OUT_FLAG_TRANSPARENT. 3. Check the output buffer StringInfo background should be ignored according to UEFI Spec. 4. The return status should be EFI_SUCCESS.

27.8.2 StringIdToImageEx()

Number	GUID	Assertion	Test Description
5.18.8.2.1	0x7baa464a, 0x572c, 0x4fa9, 0x80, 0xa3, 0x99, 0xa0, 0x61, 0xc0, 0x46, 0x4f	EFI_HII_FONT_EX_PROTOCOL. StringIdToImageEx() - StringIdToImageEx() returns EFI_INVALID_PARAMETER when Blt is NULL.	1. Call StringIdToImageEx() when Blt is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.8.2.2	0xa086a16b, 0x6e61, 0x4f06, 0xb5, 0xd, 0xac, 0x6e, 0x80, 0x71, 0x11, 0xe4	EFI_HII_FONT_EX_PROTOCOL. StringIdToImageEx() - StringIdToImageEx() returns EFI_INVALID_PARAMETER when PackageList is NULL.	1. Call StringIdToImageEx() when PackageList is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.8.2.3	0x10931cc4, 0xfa08, 0x4df8, 0xab, 0x6a, 0xb3, 0x8f, 0xa5, 0xc6, 0x84, 0x24	EFI_HII_FONT_EX_PROTOCOL. StringIdToImageEx() - StringIdToImageEx() returns EFI_NOT_FOUND when PackageList is not in Database.	1. Call StringIdToImageEx() when PackageList is not in Database, the return status should be EFI_NOT_FOUND.
5.18.8.2.4	0x7623d5de, 0x71e9, 0x49f6, 0xb7, 0x9f, 0xd2, 0x6f, 0x38, 0x69, 0xae, 0xe9	EFI_HII_FONT_EX_PROTOCOL. StringIdToImageEx() - StringIdToImageEx() returns EFI_NOT_FOUND when StringId is not in PackageList.	1. Call StringIdToImageEx() when StringId is not in PackageList, the return status should be EFI_NOT_FOUND.
5.18.8.2.5	0x36cd9086, 0x8e5e, 0x4a95, 0xb4, 0xdd, 0x56, 0x94, 0x74, 0x5c, 0x21, 0x37	EFI_HII_FONT_EX_PROTOCOL. StringIdToImageEx() - StringIdToImageEx() returns EFI_INVALID_PARAMETER when Flags is the invalid combination.	1. Call StringIdToImageEx() when Flags are EFI_HII_OUT_FLAG_CLIP_CLEAN_X with EFI_HII_OUT_FLAG_WRAP, the return status should be EFI_INVALID_PARAMETER.
5.18.8.2.6	0x16b7317e, 0x1196, 0x4323, 0x9d, 0xeb, 0xe8, 0xc7, 0x44, 0x32, 0x7e, 0x20	EFI_HII_FONT_EX_PROTOCOL. StringIdToImageEx() - StringIdToImageEx() returns EFI_INVALID_PARAMETER when Flags is the invalid combination.	1. Call StringIdToImageEx() when Flags is EFI_HII_OUT_FLAG_CLIP_CLEAN_X without EFI_HII_OUT_FLAG_CLIP, the return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.18.8.2.7	0xc3a512bc, 0x6464, 0x4e74, 0xab, 0x8d, 0x41, 0xd5, 0x42, 0xd6, 0xad, 0x66	EFI_HII_FONT_EX_PROTOCOL. StringIdToImageEx() - StringIdToImageEx() returns EFI_SUCCESS with valid parameters.	1. Call StringIdToImageEx() with valid parameters and use EFI_GRAPUICS_OUTPUT_BLT_PIXEL structure in EFI_IMAGE_OUTPUT structure, the return status should be EFI_SUCCESS.
5.18.8.2.8	0x9c84a237, 0x9ba5, 0x417a, 0x94, 0xcd, 0xf5, 0xed, 0x37, 0xf7, 0xbb, 0x9e	EFI_HII_FONT_EX_PROTOCOL. StringIdToImageEx() - StringIdToImageEx() returns EFI_SUCCESS with valid parameters.	1. Call StringIdToImageEx() with valid parameters and use EFI_GRAPUICS_OUTPUT_PROTOCOL structure in EFI_IMAGE_OUTPUT structure, the return status should be EFI_SUCCESS.

27.8.3 GetGlyphEx()

Number	GUID	Assertion	Test Description
5.18.8.3.1	0x4e1b65f1, 0xa0c1, 0x4f13, 0xb6, 0xfb, 0x2a, 0xdc, 0xaa, 0x21, 0x8d, 0x89	EFI_HII_FONT_EX_PROTOCOL. GetGlyphEx() - GetGlyphEx () returns EFI_INVALID_PARAMETER when Blt is NULL.	1. Call GetGlyphEx() when Blt is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.8.3.2	0x139af9e5, 0x5d3e, 0x46b2, 0x83, 0x9c, 0x52, 0x54, 0x66, 0xf1, 0xe0, 0xe	EFI_HII_FONT_EX_PROTOCOL. GetGlyphEx() - GetGlyphEx () returns EFI_INVALID_PARAMETER when *Blt is not NULL.	1. Call GetGlyphEx() when *Blt is not NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.8.3.3	0xf3fc9dce, 0x7f2c, 0x45d7, 0x87, 0xcf, 0x55, 0x17, 0xea, 0xcf, 0x9d, 0x4d	EFI_HII_FONT_EX_PROTOCOL. GetGlyphEx() - GetGlyphEx () returns EFI_SUCCESS with valid parameters.	1. Call GetGlyphEx() with valid parameters, the return status should be EFI_SUCCESS.

27.8.4 GetFontInfoEx()

Number	GUID	Assertion	Test Description
5.18.8.4.1	0x9511abcb, 0x462e, 0x4b96, 0xb3, 0xf, 0xbf, 0x9b, 0xf5, 0x68, 0x73, 0xeb	EFI_HII_FONT_EX_PROTOCOL. GetFontInfoEx() - GetFontInfoEx() returns EFI_INVALID_PARAMETER with invalid EFI_FONT_INFO_MASK combination.	1. Call GetFontInfoEx() when StringInfoIn->FontInfoMask is the invalid combination, the return status should be EFI_INVALID_PARAMETER.
5.18.8.4.2	0x167059e1, 0x4bf6, 0x4d8c, 0xb0, 0x96, 0x7b, 0xf4, 0x61, 0x7b, 0x75, 0x4b	EFI_HII_FONT_EX_PROTOCOL. GetFontInfoEx() - GetFontInfoEx() returns EFI_SUCCESS with valid parameters.	1. Call GetFontInfoEx() with valid parameters, the return status should be EFI_SUCCESS.
5.18.8.4.3	0x29a5204a, 0x507e, 0x4dc0, 0xa1, 0xb1, 0x90, 0x53, 0xf7, 0x2e, 0xd7, 0x77	EFI_HII_FONT_EX_PROTOCOL. GetFontInfoEx() - GetFontInfoEx() returns EFI_SUCCESS with valid parameters(StringInfoIn is NULL).	1. Call GetFontInfoEx() with valid parameters(StringInfoIn is NULL), the return status should be EFI_SUCCESS.

27.8.5 GetGlyphInfo()

Number	GUID	Assertion	Test Description
5.18.8.5.1	0x298cb0c7, 0x7e78, 0x4e3e, 0x8d, 0x42, 0xc2, 0x2c, 0x16, 0xa0, 0x83, 0x31	EFI_HII_FONT_EX_PROTOCOL. GetGlyphInfo() - GetGlyphInfo() returns EFI_INVALID_PARAMETER when GlyphInfo is NULL.	1. Call GetGlyphInfo() when GlyphInfo is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.8.5.2	0xb20f87ce, 0xbc6b, 0x4e27, 0xb8, 0x2a, 0x61, 0x53, 0x59, 0xab, 0x92, 0xa7	EFI_HII_FONT_EX_PROTOCOL. GetFontInfoEx() - GetFontInfoEx() returns EFI_INVALID_PARAMETER when FontDisplayInfo is NULL.	1. Call GetGlyphInfo() when FontDisplayInfo is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.8.5.3	0x347f2e9e, 0x70c4, 0x4e89, 0xb9, 0x4, 0x7e, 0x5f, 0xbd, 0x78, 0x4d, 0xb3	EFI_HII_FONT_EX_PROTOCOL. GetFontInfoEx() - GetFontInfoEx() returns EFI_SUCCESS with valid parameters.	1. Call GetGlyphInfo() with valid parameters, the return status should be EFI_SUCCESS.

27.9 EFI_HII_IMAGE_EX_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_HII_IMAGE_EX_PROTOCOL Section.

27.9.1 NewImageEx()

Number	GUID	Assertion	Test Description
5.18.9.1.1	0xb604ba95, 0xf054, 0x49fd, 0xba, 0xd1, 0xd4, 0x5e, 0xd4, 0x72, 0x56, 0x74	EFI_HII_IMAGE_EX_PROTOCOL. NewImageEx() - NewImageEx () returns EFI_INVALID_PARAMETER when ImageId is NULL.	1. Call NewImageEx() when ImageId is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.9.1.2	0xd1bb7c92, 0xf5df, 0x4b54, 0xa5, 0x75, 0x51, 0xd2, 0x97, 0xd3, 0xa8, 0xc0	EFI_HII_IMAGE_EX_PROTOCOL. NewImageEx() - NewImageEx () returns EFI_INVALID_PARAMETER when Image is NULL.	1. Call NewImageEx() when Image is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.9.1.3	0x54b5f648, 0xc0de, 0x4f56, 0xb3, 0x3d, 0xa5, 0x11, 0x92, 0xc4, 0x8c, 0x96	EFI_HII_IMAGE_EX_PROTOCOL. NewImageEx() - NewImageEx () returns EFI_NOT_FOUND when PackageList is NULL.	1. Call NewImageEx() when PackageList is NULL, the return status should be EFI_NOT_FOUND.
5.18.9.1.4	0x42b10032, 0x7dd8, 0x438d, 0x97, 0xd1, 0xad, 0x38, 0xda, 0x27, 0x67, 0xc0	EFI_HII_IMAGE_EX_PROTOCOL. NewImageEx() - NewImageEx () returns EFI_SUCCESS with valid parameters.	1. Call NewImageEx() with valid parameters, the return status should be EFI_SUCCESS.

27.9.2 GetImageEx()

Number	GUID	Assertion	Test Description
5.18.9.2.1	0xdf44b77f, 0x2390, 0x47f9, 0x83, 0x11, 0xb, 0xa0, 0x76, 0xeb, 0x5f, 0x58	EFI_HII_IMAGE_EX_PROTOCOL. GetImageEx() - GetImageEx() returns EFI_NOT_FOUND when ImageId is invalid.	1. Call GetImageEx() when ImageId is invalid, the return status should be EFI_NOT_FOUND.
5.18.9.2.2	0x2ef35d72, 0xa2d7, 0x44c7, 0x80, 0x3a, 0x66, 0xa0, 0x62, 0x2c, 0x25, 0x8e	EFI_HII_IMAGE_EX_PROTOCOL. GetImageEx() - GetImageEx() returns EFI_INVALID_PARAMETER when Image is NULL.	1. Call GetImageEx() when Image is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.9.2.3	0x4562d8ad, 0x5441, 0x4a7c, 0x8b, 0xa2, 0x2c, 0x9e, 0x65, 0x31, 0x44, 0x87	EFI_HII_IMAGE_EX_PROTOCOL. GetImageEx() - GetImageEx() returns EFI_NOT_FOUND when PackageList is not in Database.	1. Call GetImageEx() when PackageList is not in Database, the return status should be EFI_NOT_FOUND.
5.18.9.2.4	0x5ce03916, 0x9b93, 0x4f09, 0xb4, 0x94, 0x68, 0x3f, 0x68, 0xe5, 0xbc, 0xa7	EFI_HII_IMAGE_EX_PROTOCOL. GetImageEx() - GetImageEx() returns EFI_NOT_FOUND when PackageList is NULL.	1. Call GetImageEx() when PackageList is NULL, the return status should be EFI_NOT_FOUND.
5.18.9.2.5	0xc30ad068, 0x7fbe, 0x4c44, 0x8a, 0x9b, 0x3f, 0xc2, 0x97, 0x8a, 0x1a, 0x13	EFI_HII_IMAGE_EX_PROTOCOL. GetImageEx() - GetImageEx() returns EFI_SUCCESS with valid parameters.	1. Call GetImageEx() with valid parameters, the return status should be EFI_SUCCESS.

27.9.3 SetImageEx()

Number	GUID	Assertion	Test Description
5.18.9.3.1	0xe88ca946, 0xed6d, 0x415d, 0x85, 0x55, 0x0, 0x27, 0x9f, 0x14, 0xc3, 0xf9	EFI_HII_IMAGE_EX_PROTOCOL. SetImageEx() - SetImageEx() returns EFI_NOT_FOUND when ImageId is invalid.	1. Call SetImageEx() when ImageId is invalid, the return status should be EFI_NOT_FOUND.
5.18.9.3.2	0x5e7cf471, 0x1f23, 0x465d, 0xab, 0x61, 0x91, 0xf9, 0x7c, 0xe6, 0xb8, 0x68	EFI_HII_IMAGE_EX_PROTOCOL. SetImageEx() - SetImageEx() returns EFI_INVALID_PARAMETER when Image is NULL.	1. Call SetImageEx() when Image is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.9.3.3	0xd9ebeb2c, 0x569c, 0x4898, 0x99, 0xf7, 0x20, 0xe4, 0x5d, 0x2f, 0x4a, 0xa9	EFI_HII_IMAGE_EX_PROTOCOL. SetImageEx() - SetImageEx() returns EFI_NOT_FOUND when PackageList is not in Database.	1. Call SetImageEx() when PackageList is not in Database, the return status should be EFI_NOT_FOUND.
5.18.9.3.4	0x2cc6d840, 0x292, 0x4b64, 0x98, 0x6f, 0xb6, 0xe, 0xf0, 0x18, 0xd2, 0x7c	EFI_HII_IMAGE_EX_PROTOCOL. SetImageEx() - SetImageEx() returns EFI_NOT_FOUND when PackageList is NULL.	1. Call SetImageEx() when PackageList is NULL, the return status should be EFI_NOT_FOUND.
5.18.9.3.5	0xbb5d5eb9, 0x70d1, 0x4888, 0x83, 0x35, 0x41, 0x95, 0x5a, 0x43, 0x8c, 0x39	EFI_HII_IMAGE_EX_PROTOCOL. SetImageEx() - SetImageEx() returns EFI_SUCCESS with valid parameters.	1. Call SetImageEx() with valid parameters, the return status should be EFI_SUCCESS.

27.9.4 DrawImageEx()

Number	GUID	Assertion	Test Description
5.18.9.4.1	0x42dd08a5, 0xbd85, 0x4eab, 0xb4, 0x74, 0x9f, 0xe2, 0x55, 0x71, 0x56, 0x8f	EFI_HII_FONT_EX_PROTOCOL. DrawImageEx() - DrawImageEx() returns EFI_INVALID_PARAMETER when Image is NULL.	1. Call DrawImageEx() when Image is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.9.4.2	0xcf06b84d, 0x8d1f, 0x43c1, 0xb5, 0xb2, 0xa3, 0x3a, 0x2, 0xc2, 0xd, 0x50	EFI_HII_FONT_EX_PROTOCOL. DrawImageEx() - DrawImageEx() returns EFI_INVALID_PARAMETER when Flag is EFI_HII_DRAW_FLAG_TRANSPARENT and Blt is NULL.	1. Call DrawImageEx() when Flag is EFI_HII_DRAW_FLAG_TRANSPARENT and Blt is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.9.4.3	0xa20a8ee4, 0x9bed, 0x4538, 0x94, 0x7a, 0xbf, 0xb7, 0x42, 0xa6, 0xaf, 0xd9	EFI_HII_FONT_EX_PROTOCOL. DrawImageEx() - DrawImageEx() returns EFI_INVALID_PARAMETER when Flag is EFI_HII_DIRECT_TO_SCREEN and no screen.	1. Call DrawImageEx() when Flag is EFI_HII_DIRECT_TO_SCREEN and no screen, the return status should be EFI_INVALID_PARAMETER.
5.18.9.4.4	0x8a4f106c, 0xdb5d, 0x4491, 0x96, 0xbd, 0x62, 0x9a, 0xa8, 0xa2, 0xc4, 0x25	EFI_HII_FONT_EX_PROTOCOL. DrawImageEx() - DrawImageEx() returns EFI_INVALID_PARAMETER when Flag is EFI_HII_DRAW_FLAG_CLIP and Blt points to NULL.	1. Call DrawImageEx() when Flag is EFI_HII_DRAW_FLAG_CLIP and Blt points to NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.9.4.5	0x4ed61351, 0xc6de, 0x4910, 0x97, 0x15, 0xcf, 0xc5, 0x5e, 0xe, 0x75, 0x9b	EFI_HII_FONT_EX_PROTOCOL. DrawImageEx() - DrawImageEx() returns EFI_INVALID_PARAMETER when Flag is EFI_HII_DRAW_FLAG_DEFAULT and Blt points to NULL, but Image->Flag is EFI_IMAGE_TRANSPARENT.	1. Call DrawImageEx() when Flag is EFI_HII_DRAW_FLAG_DEFAULT and Blt points to NULL, but Image->Flag is EFI_IMAGE_TRANSPARENT, the return status should be EFI_INVALID_PARAMETER.
5.18.9.4.6	0x3ac875ed, 0x46d4, 0x4d1d, 0xac, 0xfe, 0xdb, 0x37, 0xe5, 0xf1, 0xb7, 0xd0	EFI_HII_FONT_EX_PROTOCOL. DrawImageEx() - DrawImageEx() return EFI_SUCCESS with valid parameters.	1. Call DrawImageEx() when Flag is EFI_HII_DRAW_FLAG_FORCE_OPAQUE, Blt is NULL and other valid parameters, the return status should be EFI_SUCCESS.

Number	GUID	Assertion	Test Description
5.18.9.4.7	0x16a8be, 0x4466, 0x4777, 0xa0, 0xbd, 0xa9, 0x10, 0x1c, 0x54, 0x19, 0xa0	EFI_HII_FONT_EX_PROTOCOL. DrawImageEx() - DrawImageEx() return EFI_SUCCESS with valid parameters.	1. Call DrawImageEx() when Flag is the valid combination, Blt is NULL and other valid parameters, the return status should be EFI_SUCCESS.

27.9.5 DrawImageIdEx()

Number	GUID	Assertion	Test Description
5.18.9.5.1	0x24ddcd2b, 0xa9d8, 0x4ec5, 0xaf, 0xf6, 0x77, 0xf3, 0x69, 0x8c, 0xe, 0x19	EFI_HII_IMAGE_EX_PROTOCOL. DrawImageIdEx() - DrawImageIdEx() returns EFI_NOT_FOUND when PackageList is not in Database.	1. Call DrawImageIdEx() when PackageList is not in Database, the return status should be EFI_NOT_FOUND.
5.18.9.5.2	0x8f114d30, 0x684d, 0x402e, 0xb5, 0x35, 0x74, 0x34, 0x1e, 0xbb, 0x88, 0x5f	EFI_HII_IMAGE_EX_PROTOCOL. DrawImageIdEx() - DrawImageIdEx() returns EFI_NOT_FOUND when PackageList is NULL.	1. Call DrawImageIdEx() when PackageList is NULL, the return status should be EFI_NOT_FOUND.
5.18.9.5.3	0x446d5d03, 0xf2b6, 0x4627, 0xad, 0xd1, 0x75, 0x6d, 0xfe, 0xe9, 0x18, 0x3f	EFI_HII_IMAGE_EX_PROTOCOL. DrawImageIdEx() - DrawImageIdEx() returns EFI_NOT_FOUND when ImageId is invalid.	1. Call DrawImageIdEx() when ImageId is invalid, the return status should be EFI_NOT_FOUND.
5.18.9.5.4	0x6dbc9f6e, 0x2694, 0x44ec, 0x99, 0xe9, 0x2d, 0x67, 0x6a, 0xfe, 0x9f, 0x37	EFI_HII_IMAGE_EX_PROTOCOL. DrawImageIdEx() - DrawImageIdEx() returns EFI_NOT_FOUND when PackageList is NULL.	1. Call DrawImageIdEx() when PackageList is invalid, the return status should be EFI_NOT_FOUND.
5.18.9.5.5	0x8c43a76, 0x7f57, 0x41dd, 0x87, 0x99, 0x13, 0xcf, 0xf2, 0x5, 0x9b, 0x6	EFI_HII_IMAGE_EX_PROTOCOL. DrawImageIdEx() - DrawImageIdEx() returns EFI_INVALID_PARAMETER when Flags is EFI_HII_DRAW_FLAG_FORCE_TRANS and Blt is NULL.	1. Call DrawImageIdEx() when Flags is EFI_HII_DRAW_FLAG_FORCE_TRANS and Blt is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.9.5.6	0x39787a10, 0x1204, 0x41a5, 0xa8, 0xdb, 0xd3, 0xe9, 0x83, 0xc4, 0x47, 0x44	EFI_HII_IMAGE_EX_PROTOCOL. DrawImageIdEx() - DrawImageIdEx() returns EFI_INVALID_PARAMETER when Flags is EFI_HII_DRAW_FLAG_CLIP and Blt points to NULL.	1. Call DrawImageIdEx() when Flags is EFI_HII_DRAW_FLAG_CLIP and Blt points to NULL, the return status should be EFI_INVALID_PARAMETER.

Number	GUID	Assertion	Test Description
5.18.9.5.7	0x82c37f35, 0xbca3, 0x494e, 0x8a, 0xdb, 0xf6, 0xd8, 0xf0, 0x7a, 0xf6, 0xe3	EFI_HII_IMAGE_EX_PROTOCOL. DrawImageIdEx() - DrawImageIdEx() returns EFI_INVALID_PARAMETER when Flags is EFI_HII_DRAW_FLAG_DEFAULT, Blt points to NULL and Image->Flags is EFI_IMAGE_TRANSPARENT.	1. Call DrawImageIdEx() when Flags is EFI_HII_DRAW_FLAG_DEFAULT, Blt points to NULL and Image->Flags is EFI_IMAGE_TRANSPARENT, the return status should be EFI_INVALID_PARAMETER.
5.18.9.5.8	0x1c03d9b0, 0x8d9c, 0x40bf, 0x94, 0xa7, 0xa7, 0x85, 0xa3, 0x52, 0xa2, 0x68	EFI_HII_FONT_EX_PROTOCOL. DrawImageIdEx() - DrawImageIdEx() return EFI_SUCCESS with valid parameters.	1. Call DrawImageIdEx() when Flag is EFI_HII_DRAW_FLAG_FORCE_OPAQUE, Blt is NULL and other valid parameters, the return status should be EFI_SUCCESS.
5.18.9.5.9	0x5ee23086, 0xe0ee, 0x4cc8, 0x85, 0xf2, 0x5a, 0xd3, 0x52, 0xd7, 0x4d, 0xb7	EFI_HII_FONT_EX_PROTOCOL. DrawImageIdEx() - DrawImageIdEx() return EFI_SUCCESS with valid parameters.	1. Call DrawImageIdEx() when Flag is the valid combination, Blt is NULL and other valid parameters, the return status should be EFI_SUCCESS.

27.9.6 GetImageInfo()

Number	GUID	Assertion	Test Description
5.18.9.6.1	0x5c53ff3e, 0xbfb, 0x40e7, 0x9b, 0xa8, 0x1b, 0x6e, 0xda, 0x67, 0xda, 0xc0	EFI_HII_IMAGE_EX_PROTOCOL. GetImageInfo() - GetImageInfo() returns EFI_NOT_FOUND when ImageId is invalid.	1. Call GetImageInfo() when ImageId is invalid, the return status should be EFI_NOT_FOUND.
5.18.9.6.2	0xf61dfb48, 0x1c77, 0x4907, 0x9f, 0xab, 0x43, 0x93, 0x17, 0x8c, 0x99, 0xee	EFI_HII_IMAGE_EX_PROTOCOL. GetImageInfo() - GetImageInfo() returns EFI_INVALID_PARAMETER when Image is NULL.	1. Call GetImageInfo() when Image is NULL, the return status should be EFI_INVALID_PARAMETER.
5.18.9.6.3	0x1663a5c1, 0x7897, 0x48f5, 0x93, 0xe0, 0x8e, 0x67, 0x13, 0xa, 0xc1, 0x5d	EFI_HII_IMAGE_EX_PROTOCOL. GetImageInfo() - GetImageInfo() returns EFI_INVALID_PARAMETER when ImageId is 0.	1. Call GetImageInfo() when ImageId is 0, the return status should be EFI_INVALID_PARAMETER.
5.18.9.6.4	0x9cf6b34c, 0x4d53, 0x464e, 0x99, 0x4e, 0xd0, 0x3, 0xb5, 0x7b, 0x8b, 0x67	EFI_HII_IMAGE_EX_PROTOCOL. GetImageInfo() - GetImageInfo() returns EFI_SUCCESS with valid parameters.	1. Call GetImageInfo() with valid parameters, the return status should be EFI_SUCCESS.

28 Random Number Generator Protocols

28.1 EFI_RNG_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_RNG_PROTOCOL Section.

28.1.1 GetInfo ()

Number	GUID	Assertion	Test Description
5.33.4.1.1	0xb0aeea8a, 0xcd05, 0x4254, 0xb2, 0xcb, 0x30, 0xbb, 0x90, 0x87, 0x73, 0xc6	**EFI_RNG_PROTOCOL.GetInfo() - GetInfo()** returns **EFI_SUCCESS** with valid parameters.	Call **GetInfo()** to get the RNGAlgorithmListSize. Allocate a list buffer with the RNGAlgorithmListSize gotten from step1. 3. Call **GetInfo()** with the new allocated buffer, the return status should be **EFI_SUCCESS**.
5.33.4.1.2	0x50df54e5, 0x1449, 0x4a34, 0x95, 0x6a, 0xb6, 0x61, 0x66, 0xc2, 0xd5, 0x8a	**EFI_RNG_PROTOCOL.GetInfo() - GetInfo()** returns valid algorithm with valid parameters.	Call **GetInfo()** to get the RNGAlgorithmListSize. Allocate a list buffer with the RNGAlgorithmListSize gotten from step1. Call **GetInfo()** with the new allocated buffer, the return status should be **EFI_SUCCESS**. Compare the Algorithm gotten from Step3 with the given algorithms, the result should be success.
5.33.4.1.3	0x0db3b0d2, 0x859f, 0x4682, 0x87, 0x67, 0x62, 0x35, 0x67, 0x91, 0xb7, 0x9d	**EFI_RNG_PROTOCOL.GetInfo() - GetInfo()** returns **EFI_BUFFER_TOO_SMALL** with small RNGAlgorithmListSize and returns valid size	Call **GetInfo()** with small RNGAlgorithmListSize, the return status should be **EFI_BUFFER_TOO_SMALL** and returns valid size

28.1.2 GetRNG()

Number	GUID	Assertion	Test Description
5.33.4.2.1	0x4a54a35e, 0x66ac, 0x4c2e, 0x92, 0xd8, 0x7b, 0x26, 0x3d, 0x8a, 0x77, 0xa8	EFI_RNG_PROTOCOL.GetRNG() - GetRNG() returns EFI_SUCCESS with valid parameters.	Call GetInfo() to get the RNGAlgorithmListSize. Allocate a list buffer with the RNGAlgorithmListSize gotten from step1. Call GetInfo() with the new allocated Buffer. Call GetRNG() with valid parameters, the return status should be EFI_SUCCESS.
5.33.4.2.2	0xe3d11e22, 0xeddb, 0x40c4, 0x8f, 0x6d, 0x25, 0x79, 0x33, 0xea, 0x62, 0xf8	EFI_RNG_PROTOCOL.GetRNG() - GetRNG() returns EFI_SUCCESS with default algorithm.	Call GetRNG() with default algorithm, the return status should be EFI_SUCCESS.
5.33.4.2.3	0xe79e5379, 0xd4dc, 0x4624, 0x88, 0x05, 0x09, 0x46, 0x1c, 0x09, 0x78, 0x28	EFI_RNG_PROTOCOL.GetRNG() - GetRNG() returns EFI_INVALID_PARAMETER when RNGValueLength is 0.	Call GetRNG() when RNGValueLength is 0, the return status should be EFI_INVALID_PARAMETER.
5.33.4.2.4	0x76ac3e4c, 0x5f59, 0x4c21, 0x82, 0x0a, 0xe4, 0x24, 0xc2, 0xef, 0x36, 0x14	EFI_RNG_PROTOCOL.GetRNG() - GetRNG() returns EFI_INVALID_PARAMETER when RNGValue is NULL.	Call GetRNG() when RNGValue is NULL, the return status should be EFI_INVALID_PARAMETER.
5.33.4.2.5	0x27451869, 0x357d, 0x4e92, 0xb8, 0xb0, 0xb8, 0xc5, 0xba, 0xb9, 0xa4, 0xe9	EFI_RNG_PROTOCOL.GetRNG() - GetRNG() returns EFI_INVALID_PARAMETER when RNGAlgorithm is NULL and RNGValueLength is 0.	Call GetRNG() when RNGAlgorithm is NULL and RNGValueLength is 0, the return status should be EFI_INVALID_PARAMETER.

5.33.4.2.6	0x31ce0e8, 0x3604, 0x4489, 0x93, 0x6c, 0x60, 0x8c, 0x9b, 0x2c, 0xf8, 0xf4	**EFI_RNG_PROTOCOL.GetRNG() - GetRNG()** returns **EFI_INVALID_PARAMETER** when RNGValueLength is 0 after the RNGAlgorithm is freed.	Call **GetRNG()** when RNGValue is NULL after the RNGAlgorithm is freed, the return status should be **EFI_INVALID_PARAMETER**.
5.33.4.2.7	0x7a4ea182, 0xa4cd, 0x441d, 0x98, 0xd7, 0x73, 0x65, 0x87, 0x6f, 0xfa, 0x77	**EFI_RNG_PROTOCOL.GetRNG() - GetRNG()** returns **EFI_UNSUPPORTED** when RNGAlgorithm is unsupported.	Call **GetRNG()** when RNGAlgorithm is unsupported, the return status should be **EFI_UNSUPPORTED**.

29 Timestamp Protocols

29.1 EFI_TIMESTAMP_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_TIMESTAMP_PROTOCOL Section.

29.1.1 GetTimestamp()

Number	GUID	Assertion	Test Description
5.34.1.1.1	0xa971e7ad, 0x5889, 0x4af0, 0x8c, 0x7e, 0x05, 0xa6, 0x88, 0xca, 0xf6, 0xd8	`EFI_TIMESTAMP_PROTOCOL.GetTimestamp - GetTimestamp()` returns reasonable value.	Call `GetTimestamp ()` should return a reasonable value.

29.1.2 GetProperties()

Number	GUID	Assertion	Test Description
5.34.1.2.1	0x7530e468, 0xc9d0, 0x4881, 0xa2, 0xe7, 0xb5, 0x9f, 0x80, 0x38, 0x70, 0x26	`EFI_TIMESTAMP_PROTOCOL.GetProperties- GetProperties()` returns `EFI_SUCCESS` with properties being not `NULL`.	Call `GetProperties ()` with properties being not NULL, the return status should be `EFI_SUCCESS`.
5.34.1.2.2	0x2e9847b0, 0x8d24, 0x4c8d, 0xbd, 0xbc, 0x57, 0xc5, 0xdb, 0x10, 0x10, 0x95	`EFI_TIMESTAMP_PROTOCOL.GetProperties- GetProperties()` Properties.EndValue returned from `GetProperties()` should be in 0xFFFF format.	Call `GetProperties ()` with properties being not `NULL`, Properties.EndValue returned from `GetProperties()` should be in 0xFFFF format.

5.34.1.2.3	0x3b1d442f, 0xcc6d, 0x4e89, 0xa3, 0x91, 0x00, 0x40, 0xb2, 0x39, 0xd7, 0xb6	**EFI_TIMESTAMP_PROTOCOL.GetProperties-GetProperties()** returns **EFI_INVALID_PARAMETER** with properties being **NULL**.	Call **GetProperties ()** with properties being **NULL**, the return status should be **EFI_INVALID_PARAMETER**.

30 Protocols String Services Test

30.1 EFI_REGULAR_EXPRESSION_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_REGULAR_EXPRESSION_PROTOCOL Section.

30.1.1 MatchString()

Number	GUID	Assertion	Test Description
5.35.1.1.1	0x9cec70a0, 0xfb56, 0x4b7f, 0x95, 0x31, 0xeb, 0xd0, 0x61, 0xa2, 0xcf, 0x8f	EFI_REGULAR_EXPRESSION_PROTOCOL. MatchString() - MatchString() returns EFI_INVALID_PARAMETER when String is NULL.	1. Call MatchString() when String is NULL, the return status should be EFI_INVALID_PARAMETER.
5.35.1.1.2	0xfdceb7d8, 0x5fb7, 0x43c8, 0x8f, 0xa8, 0xec, 0xf, 0x7f, 0x14, 0x34, 0x29	EFI_REGULAR_EXPRESSION_PROTOCOL. MatchString() - MatchString() returns EFI_INVALID_PARAMETER when Pattern is NULL.	1. Call MatchString() when Pattern is NULL, the return status should be EFI_INVALID_PARAMETER.
5.35.1.1.3	0x76813d40, 0xd2a7, 0x4912, 0x9e, 0xc4, 0x96, 0x6b, 0x14, 0x15, 0x4b, 0x51	EFI_REGULAR_EXPRESSION_PROTOCOL. MatchString() - MatchString() returns EFI_INVALID_PARAMETER when Result is NULL.	1. Call MatchString() when Result is NULL, the return status should be EFI_INVALID_PARAMETER.
5.35.1.1.4	0xea3de64c, 0xe402, 0x43a7, 0xb4, 0x77, 0x66, 0xcd, 0xf5, 0x13, 0x1e, 0x85	EFI_REGULAR_EXPRESSION_PROTOCOL. MatchString() - MatchString() returns EFI_INVALID_PARAMETER when CapturesCount is NULL.	1. Call MatchString() when CapturesCount is NULL, the return status should be EFI_INVALID_PARAMETER.
5.35.1.1.5	0x98dee30e, 0xdc2b, 0x4dc6, {0x83, 0x10, 0xf8, 0x85, 0x17, 0x2f, 0x4c, 0xc8	EFI_REGULAR_EXPRESSION_PROTOCOL. MatchString() - MatchString() returns EFI_UNSUPPORTED with unsupported SyntaxType.	1. Call MatchString() with unsupported SyntaxType, the return status should be EFI_UNSUPPORTED.

5.35.1.1.6	0x94407424, 0xc17e, 0x4a28, 0xb7, 0x84, 0x3f, 0x84, 0x39, 0xcf, 0x30, 0x96	EFI_REGULAR_EXPRESSION_PROTOCOL. MatchString() - MatchString() returns EFI_SUCCESS with all supported SyntaxType.	1. Call MatchString() with all supported SyntaxType, the return status should be EFI_SUCCESS.
5.35.1.1.7	0x3d3be925, 0xfbf3, 0x425c, 0xbd, 0xd, 0x2b, 0x95, 0x2f, 0xf3, 0xbf, 0xe8	EFI_REGULAR_EXPRESSION_PROTOCOL. MatchString() - MatchString() returns EFI_SUCCESS with default SyntaxType.	1. Call MatchString() with default SyntaxType, the return status should be EFI_SUCCESS.

30.1.2 GetInfo()

Number	GUID	Assertion	Test Description
5.35.1.2.1	0x3219e1b1, 0xac3a, 0x4f53, 0x99, 0x11, 0xf3, 0x25, 0x44, 0x5b, 0xa8, 0x26	EFI_REGULAR_EXPRESSION_PROTOCOL. GetInfo() - GetInfo() returns EFI_BUFFER_TOO_SMALL when SyntaxTypeListSize is too small to hold the result.	1. Call GetInfo() when SyntaxTypeListSize is too small to hold the result, the return status should be EFI_BUFFER_TOO_SMALL. The outputted SyntaxTypeListSize should be the multiple of size of EFI_REGEX_SYNTAX_TYPE.
5.35.1.2.2	0x5a216f4d, 0xb4fe, 0x486d, 0x8e, 0x2e, 0x7b, 0xf9, 0x98, 0x47, 0x62, 0xbd	EFI_REGULAR_EXPRESSION_PROTOCOL. GetInfo() - GetInfo() returns EFI_INVALID_PARAMETER when SyntaxTypeListSize is NULL.	1. Call GetInfo() when SyntaxTypeListSize is NULL, the return status should be EFI_INVALID_PARAMETER.
5.35.1.2.3	0x5365a661, 0xdb02, 0x46ed, 0xb8, 0x3e, 0xbc, 0x71, 0x6d, 0x6a, 0x8b, 0xb4	EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL. GetData() - GetData() returns EFI_SUCCESS with valid parameters.	1. Call GetInfo() with valid parameters, the return status should be EFI_SUCCESS. The outputted SyntaxTypeListSize should be same as the input size.

Appendix A
Format of Test Profiles

A.1 EFI Requirements Test Profile

```
File Path: SCT\Dependency\EfiCompliantBBTest\EfiCompliant.Ini
[Platform Specific]
ConsoleDevices            = <yes: if this platform includes
console devices>
GraphicalConsoleDevices   = <yes: if this platform includes
graphical console devices>
PointerDevices            = <yes: if this platform includes a
pointer device as part of its console support>
BootFromDiskDevices       = <yes: if this platform supports to
boot from a disk device>
BootFromNetworkDevices    = <yes: if this platform supports to
boot from a network device>
UartDevices               = <yes: if this platform includes a
byte-stream device such as a UART>
PciBusSupport             = <yes: if this platform includes PCI
bus support>
UsbBusSupport             = <yes: if this platform includes USB
bus support>
ScsiPassThru             = <yes: if this platform includes an I/
O system that uses SCSI command packets>
DebugSupport              = <yes: if this platform supports
debugging capabilities>
PlatformDriverOverride    = <yes: includes the ability to
override the default driver>
```

A.2 EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL Test Profile

```
[PollMem_Func]
DevicePath= <The PCI root bridge device path string>
Address    = <The memory address controlled by this root bridge>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH>
TargetValue= <The target value to be set and polled in
destination address, in hex format>
AlternateValue= <The alternate value to be set in destination
address, in hex format>

[PollIo_Func]
DevicePath= <The PCI root bridge device path string>
Address    = <The Io address controlled by this root bridge>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH>
```

```
TargetValue= <The target value to be set and polled in
destination address, in hex format>
AlternateValue= <The alternate value to be set in destination
address, in hex format>

[MemRead_Func]
DevicePath= <The PCI root bridge device path string>
Address   = <The memory address controlled by this root bridge>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH>
Length    = <The tested address length, in hex format>
DataUnits = <The data unit to be written in to tested area, this
item can be NULL>

[MemWrite_Func]
DevicePath= <The PCI root bridge device path string>
Address   = <The memory address controlled by this root bridge>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH>
Length    = <The tested address length, in hex format>
DataUnits = <The data unit to be written in to tested area, this
item can be NULL>

[IoRead_Func]
DevicePath= <The PCI root bridge device path string>
Address   = <The Io address controlled by this root bridge>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH>
Length    = <The tested address length, in hex format>
DataUnits = <The data unit to be written in to tested area, this
item can be NULL>

[IoWrite_Func]
DevicePath= <The PCI root bridge device path string>
Address   = <The Io address controlled by this root bridge>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH>
Length    = <The tested address length, in hex format>
DataUnits = <The data unit to be written in to tested area, this
item can be NULL>

[PciRead_Func]
DevicePath= <The PCI root bridge device path string>
Address   = <The PCI address controlled by this root bridge>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH>
Length    = <The tested address length, in hex format>
DataUnits = <The data unit to be written in to tested area, this
item can be NULL>

[PciWrite_Func]
DevicePath= <The PCI root bridge device path string>
```

```
Address   = <The PCI address controlled by this root bridge>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH>
Length    = <The tested address length, in hex format>
DataUnits = <The data unit to be written in to tested area, this
item can be NULL>

[CopyMem_Func]
DevicePath= <The PCI root bridge device path string>
Address   = <The memory address controlled by this root bridge>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH>
Length    = <The tested address length, in hex format>
DataUnits = <The data unit to be written in to tested area, this
item can be NULL>

[MemRead_Conf]
DevicePath= <The PCI root bridge device path string>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH
invalid for this system>

[MemWrite_Conf]
DevicePath= <The PCI root bridge device path string>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH
invalid for this system>

[IoRead_Conf]
DevicePath= <The PCI root bridge device path string>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH
invalid for this system>

[IoWrite_Conf]
DevicePath= <The PCI root bridge device path string>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH
invalid for this system>

[PciRead_Conf]
DevicePath= <The PCI root bridge device path string>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH
invalid for this system>

[PciWrite_Conf]
DevicePath= <The PCI root bridge device path string>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH
invalid for this system>

[CopyMem_Conf]
DevicePath= <The PCI root bridge device path string>
RootBridgeIoWidth= <The EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH
```

```
invalid for this system>
```

A.3 EFI_PCI_IO_PROTOCOL Test Profile

```
[PollMem_Func]
DevicePath= <The Pci Device Path String>
BarIndex  = <The BAR Index valid value is 0-5>
AddressOffset= <The Address offset in this BAR, in hex format>
PciIoWidth= <The EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint8>
TargetValue= <The target value to Poll in destination address, in
hex format>
AlternateValue= <The alternate value set in destination address,
in hex format>

[PollIo_Func]
DevicePath= <The Pci Device Path String>
BarIndex  = <The BAR Index valid value is 0-5>
AddressOffset= <The Address offset in this BAR, in hex format>
PciIoWidth= <The EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint8>
TargetValue= <The target value to Poll in destination address, in
hex format>
AlternateValue= <The alternate value set in destination address,
in hex format>

[MemRead_Func]
DevicePath= <The Pci Device Path String>
BarIndex  = <The BAR Index valid value is 0-5>
AddressOffset= <The Address offset in this BAR, in hex format>
Length    = <The Address length to be tested, in hex format>
PciIoWidth= <The EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint8>
DatUnits  = <The data units to be write into the destination
address, can be NULL>

[MemWrite_Func]
DevicePath= <The Pci Device Path String>
BarIndex  = <The BAR Index valid value is 0-5>
AddressOffset= <The Address offset in this BAR, in hex format>
Length    = <The Address length to be tested, in hex format>
PciIoWidth= <The EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint8>
DatUnits  = <The data units to be write into the destination
address, can be NULL>

[IoRead_Func]
```

```
DevicePath= <The Pci Device Path String>
BarIndex  = <The BAR Index valid value is 0-5>
AddressOffset= <The Address offset in this BAR, in hex format>

Length    = <The Address length to be tested, in hex format>
PciIoWidth= <The EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint8>
DatUnits  = <The data units to be write into the destination
address, can be NULL>

[IoWrite_Func]
DevicePath= <The Pci Device Path String>
BarIndex  = <The BAR Index valid value is 0-5>
AddressOffset= <The Address offset in this BAR, in hex format>
Length    = <The Address length to be tested, in hex format>
PciIoWidth= <The EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint8>
DatUnits  = <The data units to be write into the destination
address, can be NULL>

[PciRead_Func]
DevicePath= <The Pci Device Path String>
AddressOffset= <The Address offset in configuration space for
this device, in hex format>
Length    = <The Address length to be tested, in hex format>
PciIoWidth= <The EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint8>
DatUnits  = <The data units to be write into the destination
address, can be NULL>

[PciWrite_Func]
DevicePath= <The Pci Device Path String>
AddressOffset= <The Address offset in configuration space for
this device, in hex format>
Length    = <The Address length to be tested, in hex format>
PciIoWidth= <The EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint8>
DatUnits  = <The data units to be write into the destination
address, can be NULL>

[CopyMem_Func]
DevicePath= <The Pci Device Path String>
SrcBarIndex= <Source BAR index valid value is 0-5>
DestBarIndex= <Destination BAR index valid value is 0-5>
SrcAddressOffset= <The address offset in source BAR resource>
DestAddressOffset= <The address offset in destination BAR
resource>
```

```
Length    = <The Address length to be tested, in hex format>
PciIoWidth= <The EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint8>
DatUnits  = <The data units to be write into the source address,
can be NULL>

[PollMem_Conf]
DevicePath= <The Pci Device Path String>
PciIoWidth= <The invalid EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint64 on IA32 platform>

[PollIo_Conf]
DevicePath= <The Pci Device Path String>
PciIoWidth= <The invalid EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint64 on IA32 platform>

[MemRead_Conf]
DevicePath= <The Pci Device Path String>
PciIoWidth= <The invalid EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint64 on IA32 platform>

[MemWrite_Conf]
DevicePath= <The Pci Device Path String>
PciIoWidth= <The invalid EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint64 on IA32 platform>

[IoRead_Conf]
DevicePath= <The Pci Device Path String>
PciIoWidth= <The invalid EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint64 on IA32 platform>

[IoWrite_Conf]
DevicePath= <The Pci Device Path String>
PciIoWidth= <The invalid EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint64 on IA32 platform>

[PciRead_Conf]
DevicePath= <The Pci Device Path String>
PciIoWidth= <The invalid EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint64 on IA32 platform>

[PciWrite_Conf]
DevicePath= <The Pci Device Path String>
PciIoWidth= <The invalid EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint64 on IA32 platform>

[CopyMem_Conf]
```

```
DevicePath= <The Pci Device Path String>
PciIoWidth= <The invalid EFI_PCI_IO_PROTOCOL_WIDTH. For example
EfiPciIoWidthUint64 on IA32 platform>
```

A.4 EFI_DEVICE_IO_PROTOCOL Test Profile

```
[MemRead_Func]
DevicePath= <The Device IO Protocol instance device path>
ValidBaseAddress= <The Memory address in this Device>
ValidEfiIoWidth= <The valid EFI_IO_WIDTH value>
Length    = <The Data length to be tested>

[MemWrite_Func]
DevicePath= <The Device IO Protocol instance device path>
ValidBaseAddress= <The Memory address in this Device>
ValidEfiIoWidth= <The valid EFI_IO_WIDTH value>
Length    = <The Data length to be tested>

[IoRead_Func]
DevicePath= <The Device IO Protocol instance device path>
ValidBaseAddress= <The Io address in this Device>
ValidEfiIoWidth= <The valid EFI_IO_WIDTH value>
Length    = <The Data length to be tested>

[IoWrite_Func]
DevicePath= <The Device IO Protocol instance device path>
ValidBaseAddress= <The Io address in this Device>
ValidEfiIoWidth= <The valid EFI_IO_WIDTH value>
Length    = <The Data length to be tested>

[PciRead_Func]
DevicePath= <The Device IO Protocol instance device path>
ValidBaseAddress= <The PCI address>
ValidEfiIoWidth= <The valid EFI_IO_WIDTH value>
Length    = <The Data length to be tested>
DataUnits = <The data for this PCI address range>

[PciWrite_Func]
DevicePath= <The Device IO Protocol instance device path>
ValidBaseAddress= <The PCI address >
ValidEfiIoWidth= <The valid EFI_IO_WIDTH value>
Length    = <The Data length to be tested>
DataUnits = <The data to be written for this PCI address range>

[MemRead_Conf]
DevicePath= <The Device IO Protocol instance device path>
ValidBaseAddress= <The Memory address in this device>
InvalidEfiIoWidth= <The EFI_IO_WIDTH invalid for this system>

[MemWrite_Conf]
DevicePath= <The Device IO Protocol instance device path>
```

```
ValidBaseAddress= <The Memory address in this device>
InvalidEfiIoWidth= <The EFI_IO_WIDTH invalid for this system>

[IoRead_Conf]
DevicePath= <The Device IO Protocol instance device path>
ValidBaseAddress= <The Io address in this device>
InvalidEfiIoWidth= <The EFI_IO_WIDTH invalid for this system>

[IoWrite_Conf]
DevicePath= <The Device IO Protocol instance device path>
ValidBaseAddress= <The Io address in this device>
InvalidEfiIoWidth= <The EFI_IO_WIDTH invalid for this system>

[PciRead_Conf]
DevicePath= <The Device IO Protocol instance device path>
ValidBaseAddress= <The Valid PCI address >
InvalidEfiIoWidth = <The EFI_IO_WIDTH invalid for this system>

[PciWrite_Conf]
DevicePath= <The Device IO Protocol instance device path>
ValidBaseAddress= <The Valid PCI address >
InvalidEfiIoWidth = <The EFI_IO_WIDTH invalid for this system>

[AllocateBuffer_Conf]
DevicePath= <The Device IO Protocol instance device path>
InvalidBaseAddress= <The memory address invalid for this system>

[PciDevicePath_Conf]
DevicePath= <The Device IO Protocol instance device path>
InvalidBaseAddress= <The PCI address invalid for this system>
>
```

Appendix B
Deprecated Protocols

This appendix lists the Protocol , GUID, and revision identifier name changes and the deprecated protocols compared to the *EFI Specification 1.10.* The protocols listed are not Runtime, Reentrant or MP Safe. Protocols are listed by EFI 1.10 name.

For protocols in the table whose TPL is not <= TPL_NOTIFY:

This function must be called at a TPL level less then or equal to %%%%.

%%%% is TPL_CALLBACK or TPL_APPLICATION. The <= is done via text.

Table 10. Protocol Name changes

EFI 11.0 Protocol Name	UEFI 2.0 Protocol Name
EFI_LOADED_IMAGE	EFI_LOADED_IMAGE_PROTOCOL
TPL	<= TPL_NOTIFY
New GUID name	EFI_LOADED_IMAGE_PROTOCOL_GUID
EFI_DEVICE_PATH	EFI_DEVICE_PATH_PROTOCOL
TPL	<= TPL_NOTIFY
New GUID name	EFI_DEVICE_PATH_PROTOCOL_GUID
SIMPLE_INPUT_INTERFACE	EFI_SIMPLE_INPUT_PROTOCOL
TPL	<= TPL_APPLICATION
New GUID name	EFI_SIMPLE_INPUT_PROTOCOL_GUID
SIMPLE_TEXT_OUTPUT_INTERFACE	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL
TPL	<=TPL_CALLBACK
New GUID name	EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL_GUID
SERIAL_IO_INTERFACE	EFI_SERIAL_IO_PROTOCOL
TPL	<=TPL_CALLBACK
New GUID name	EFI_SERIAL_IO_PROTOCOL_GUID
EFI_LOAD_FILE_INTERFACE	EFI_LOAD_FILE_PROTOCOL
TPL	<= TPL_NOTIFY
New GUID name	EFI_LOAD_FILE_PROTOCOL_GUID
EFI_FILE_IO_INTERFACE	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL
TPL	<=TPL_CALLBACK
New GUID name	EFI_FILE_SYSTEM_PROTOCOL_GUID
EFI_FILE	EFI_FILE_PROTOCOL
TPL	<= TPL_CALLBACK
New GUID name	EFI_FILE_PROTOCOL_GUID
EFI_DISK_IO	EFI_DISK_IO_PROTOCOL
TPL	<=TPL_CALLBACK

EFI 11.0 Protocol Name	UEFI 2.0 Protocol Name
New GUID name	EFI_DISK_IO_PROTOCOL_GUID
EFI_BLOCK_IO	EFI_BLOCK_IO_PROTOCOL
TPL	<=TPL_CALLBACK
New GUID name	EFI_BLOCK_IO_PROTOCOL_GUID
UNICODE_COLLATION_INTERFACE	EFI_UNICODE_COLLATION_PROTOCOL
TPL	<= TPL_NOTIFY
New GUID name	EFI_UNICODE_COLLATION_PROTOCOL_GUID
EFI_SIMPLE_NETWORK	EFI_SIMPLE_NETWORK_PROTOCOL
TPL	<=TPL_CALLBACK
New GUID name	EFI_SIMPLE_NETWORK_PROTOCOL_GUID
EFI_NETWORK_INTERFACE_IDENTIFIER_INTERFACE	EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL
TPL	<= TPL_NOTIFY
New GUID name	EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL_GUID
EFI_PXE_BASE_CODE	EFI_PXE_BASE_CODE_PROTOCOL
TPL	<= TPL_NOTIFY
New GUID name	EFI_ PXE_BASE_CODE _PROTOCOL_GUID
EFI_PXE_BASE_CODE_CALLBACK	EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL
TPL	<= TPL_NOTIFY
New GUID name	EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL_GUID
EFI_DEVICE_IO_INTERFACE	EFI_DEVICE_IO_PROTOCOL
TPL	<= TPL_NOTIFY
New GUID name	EFI_DEVICE_IO_PROTOCOL_GUID

Table 11. Revision Identifier Name Changes

EFI 11.0 Revision Identifier Name	UEFI 2.0 Revision Identifier Name
EFI_LOADED_IMAGE_INFORMATION_REVISION	EFI_LOADED_IMAGE_PROTOCOL_REVISION
SERIAL_IO_INTERFACE_REVISION	EFI_SERIAL_IO_PROTOCOL_REVISION
EFI_FILE_IO_INTERFACE_REVISION	EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_REVISION
EFI_FILE_REVISION	EFI_FILE_PROTOCOL_REVISION
EFI_DISK_IO_INTERFACE_REVISION	EFI_DISK_IO_PROTOCOL_REVISION
EFI_BLOCK_IO_INTERFACE_REVISION	EFI_BLOCK_IO_PROTOCOL_REVISION
EFI_SIMPLE_NETWORK_INTERFACE_REVISION	EFI_SIMPLE_NETWORK_PROTOCOL_REVISION

EFI 11.0 Revision Identifier Name	UEFI 2.0 Revision Identifier Name
EFI_NETWORK_INTERFACE_IDENTIFIER_INTERFACE _REVISION	EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL _REVISION
EFI_PXE_BASE_CODE_INTERFACE_REVISION	EFI_PXE_BASE_CODE_PROTOCOL_REVISION
EFI_PXE_BASE_CODE_CALLBACK_INTERFACE _REVISION	EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL _REVISION

B.1 Deprecated Protocols

Device I/O Protocol

The support of the Device I/O Protocol (see EFI 1.1 Chapter 18) has been replaced by the use of the **PCI Root Bridge I/O** protocols from the UEFI 2.0 specification and following. Note: certain "legacy" EFI applications such as some of the ones that reside in the EFI Toolkit assume the presence of Device I/O.

UGA I/O + UGA Draw Protocol

The support of the UGA * Protocols (see EFI 1.1 Section 10.7) have been replaced by the use of the **EFI Graphics Output Protocol** described in the UEFI 2.0 specification.

USB Host Controller Protocol (version that existed for EFI 1.1)

The support of the USB Host Controller Protocol (see EFI 1.1 Section 14.1) has been replaced by the use of a UEFI 2.0 instance that covers both USB 1.1 and USB 2.0 support, as described in the UEFI 2.0 specification and following. It replaces the pre-existing protocol definition.

SCSI Passthru Protocol

The support of the SCSI Passthru Protocol (see EFI 1.1 Section 13.1) has been replaced by the use of the **Extended SCSI Passthru Protocol** which is described in the UEFI 2.0 specification.

BIS Protocol

Remains as an optional protocol.

B.2 EFI_UGA_DRAW_PROTOCOL Test

Reference Document:

Specification, Section .

B.2.1 GetMode()

Number	GUID	Assertion	Test Description
5.6.3.1.1	0x7be3c5ea, 0xca81, 0x49e2, 0xba, 0xc6, 0xb9, 0xa6, 0x5b, 0xbf, 0xfc, 0x57	**EFI_UGA_DRAW_PROTOCOL.GetMode – GetMode()** with valid parameter returns **EFI_SUCCESS**	1. Call **GetMode()** with valid parameter to backup current UGA mode. The return code should be **EFI_SUCCESS**
5.6.3.1.2	0x2dcf2f9d, 0xbc9c, 0x4be2, 0x9d, 0x0a, 0x35, 0xb9, 0x9d, 0x13, 0xb1, 0xba	**EFI_UGA_DRAW_PROTOCOL.GetMode – GetMode()** with valid parameter returns **EFI_SUCCESS**	1. Call **SetMode()** to set 800x600x32x60 UGA mode. 2. Call **GetMode()** with valid parameter. The return code should be **EFI_SUCCESS**
5.6.3.1.3	0x53954b07, 0x1ee8, 0x4ab9, 0x9b, 0x5b, 0x28, 0xbe, 0xf2, 0xae, 0x65, 0x8c	**EFI_UGA_DRAW_PROTOCOL.GetMode – GetMode()** with valid parameter returns **EFI_SUCCESS**	1. Call **SetMode()** to set supported UGA mode. 2. Call **GetMode()** with valid parameter. The return code should be **EFI_SUCCESS**
5.6.3.1.4	0xee89abe2, 0xe289, 0x4e5f, 0xbd, 0x0f, 0xee, 0x41, 0x5f, 0x9d, 0x76, 0x06	**EFI_UGA_DRAW_PROTOCOL.GetMode – GetMode()** with a *HorizontalResolution* value of **NULL** returns **EFI_INVALID_PARAMETER**	1. Call **GetMode()** with a *HorizontalResolution* value of **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.6.3.1.5	0x27e72405, 0x627f, 0x4d2d, 0x8d, 0x82, 0x1c, 0xf7, 0x5a, 0x94, 0xb1, 0xe0	**EFI_UGA_DRAW_PROTOCOL.GetMode – GetMode()** with a *VerticalResolution* value of **NULL** returns **EFI_INVALID_PARAMETER**	1. Call **GetMode()** with a *VerticalResolution* value of **NULL**. The return code should be **EFI_INVALID_PARAMETER**.
5.6.3.1.6	0x5426aa3f, 0xcf9b, 0x49a1, 0x8b, 0x83, 0x8b, 0xd7, 0x14, 0x05, 0x68, 0x72	**EFI_UGA_DRAW_PROTOCOL.GetMode – GetMode()** with a *RefreshRate* value of **NULL** returns **EFI_INVALID_PARAMETER**	1. Call **GetMode()** with a *RefreshRate* value of **NULL**. The return code should be **EFI_INVALID_PARAMETER**.

Number	GUID	Assertion	Test Description
5.6.3.1.7	0x36ebe5d4, 0xe938, 0x4859, 0xaa, 0x3e, 0xac, 0xe4, 0x49, 0xba, 0x5f, 0x17	**EFI_UGA_DRAW_PROTOCOL.GetMode – GetMode()** with a *ColorDepth* value of **NULL** returns **EFI_INVALID_PARAMETER**	1. Call **GetMode()** with a *ColorDepth* value of **NULL**. The return code should be **EFI_INVALID_PARAMETER**.

B.2.2 SetMode()

Number	GUID	Assertion	Test Description
5.6.3.2.1	0x6a5e8496, 0x0edf, 0x4616, 0x83, 0x9f, 0xde, 0xb5, 0xf8, 0xbe, 0xc8, 0xfd	EFI_UGA_DRAW_PROTOCOL.SetMode – SetMode() with supported UGA mode clears hardware frame buffer to black.	1. Call SetMode() to set supported UGA mode. 2. Call Blt() with *EfiUgaVideoToBltBuffer* operation to store screen display to buffer. 3. Each pixel in buffer should be (0,0,0).
5.6.3.2.2	0x7ff20bb2, 0xb6e7, 0x47cc, 0x86, 0xc8, 0x81, 0x7d, 0xb0, 0x73, 0x20, 0x41	EFI_UGA_DRAW_PROTOCOL.SetMode – SetMode() with resolution 800*600 color depth 32-bit and 60 refresh rate UGA mode returns EFI_SUCCESS.	1. Call SetMode() to set 800x600x32x60 UGA mode. The return code must be EFI_SUCCESS.
5.6.3.2.3	0xa5caad17, 0x8605, 0x473a, 0xab, 0x08, 0x6b, 0x87, 0x3f, 0x81, 0x2c, 0x14	EFI_UGA_DRAW_PROTOCOL.SetMode – GetMode() returns the values set by SetMode().	1. Call SetMode() to set 800x600x32x60 UGA mode. The return code must be EFI_SUCCESS. 2. Call GetMode() with valid parameter. The return values should equal to the values set by SetMode()
5.6.3.2.4	0x7d0e59bb, 0x54a3, 0x48c8, 0x85, 0xec, 0xad, 0x89, 0xeb, 0xe6, 0x8b, 0x49	EFI_UGA_DRAW_PROTOCOL.SetMode – GetMode() returns the values set by SetMode().	1. Call SetMode() to set supported UGA mode. The return code must be EFI_SUCCESS. 2. Call GetMode() with valid parameter. The return values should equal to the values set by SetMode()
5.6.3.2.5	0x86cc4728, 0x6884, 0x4743, 0x8b, 0x3b, 0x5c, 0x95, 0x5e, 0x9a, 0x77, 0x29	EFI_UGA_DRAW_PROTOCOL.SetMode – SetMode() with valid parameters returns EFI_SUCCESS or EFI_UNSUPPORTED.	1. Call SetMode() to set UGA mode. The return code must be EFI_SUCCESS or EFI_UNSUPPORTED.
5.6.3.2.6	0xe1e7967e, 0xc92a, 0x42dd, 0x93, 0xce, 0xb5, 0x1d, 0x1c, 0xe0, 0x92, 0x17	EFI_UGA_DRAW_PROTOCOL.SetMode – SetMode() with supported UGA mode returns EFI_SUCCESS.	1. Call SetMode() to restore original UGA mode. The return code must be EFI_SUCCESS.

B.2.3 Blt()

Number	GUID	Assertion	Test Description
5.6.3.3.1	0xd0bc9db6, 0xc66e, 0x46ed, 0xae, 0x61, 0x6a, 0x90, 0x28, 0x63, 0x1d, 0x34	**EFI_UGA_DRAW_PROTOCOL.Blt – Blt()** with *EfiUgaVideoFill* operation fills display rectangle with input pixel value.	1. Call **Blt()** with *EfiUgaVideoFill* operation. 2. Call **Blt()** with *EfiUgaVideoToBltBuffer* operation to store whole video display to buffer. 3. Each pixel in the display rectangle (*DestinationX, DestinationY*)(*DestinationX* + *Width*, *DestinationY* + *Height*) should be equal to the input pixel *BltBuffer*(0,0).
5.6.3.3.2	0xb567d336, 0xca3a, 0x474c, 0xaa, 0x84, 0xa7, 0xb4, 0xad, 0x61, 0x57, 0x58	**EFI_UGA_DRAW_PROTOCOL.Blt – Blt()** with *EfiUgaVideoFill* operation returns **EFI_SUCCESS**.	1. Call **Blt()** with *EfiUgaVideoFill* operation. The return code should be **EFI_SUCCESS**.
5.6.3.3.3	0x367d6e99, 0x6a11, 0x4d0f, 0xbf, 0x99, 0x7f, 0xbe, 0x43, 0x8b, 0x31, 0x57	**EFI_UGA_DRAW_PROTOCOL.Blt – Blt()** with *BltEfiUgaVideoToBltBuffer* operation returns **EFI_SUCCESS**.	1. Call **Blt()** with *BltEfiUgaVideoToBltBuffer* operation to store display to *Buffer1*. The return code should be **EFI_SUCCESS**.
5.6.3.3.4	0x85edb629, 0x147d, 0x40b0, 0x94, 0x88, 0x18, 0x02, 0x71, 0x78, 0x09, 0xcf	**EFI_UGA_DRAW_PROTOCOL.Blt – Blt()** with *BltEfiUgaVideoToBltBuffer* operation returns **EFI_SUCCESS**.	1. Call **Blt()** with *BltEfiUgaBltBufferToVideo* operation to copy *Buffer1* contents to video. 1. Call **Blt()** with *BltEfiUgaVideoToBltBuffer* operation to store video display in *Buffer2*. The return code should be **EFI_SUCCESS**.
5.6.3.3.5	0xc776eb3a, 0x6632, 0x425d, 0xb7, 0x04, 0xfa, 0xfb, 0xce, 0x1e, 0x1d, 0x0c	**EFI_UGA_DRAW_PROTOCOL.Blt – Blt()** with *BltEfiUgaBltBufferToVideo* operation returns **EFI_SUCCESS**.	1. Call **Blt()** with *BltEfiUgaBltBufferToVideo* operation to copy *Buffer1* contents to video display. The return code should be **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.6.3.3.6	0x92a04254, 0x6cbe, 0x45be, 0x87, 0xc4, 0x38, 0xd4, 0x66, 0x66, 0x11, 0xe6	**EFI_UGA_DRAW_PROTOCOL.Blt - Blt()** with *BltEfiUgaVideoToBltBuffer* and *BltEfiUgaBltBufferToVideo* operation gets the same content of display rectangle and buffer.	1. Call **Blt()** to output a blue rectangle on screen and call **Blt()** with *BltEfiUgaVideoToBltBuffer* operation to store display to *Buffer1*. 2. Call **Blt()** with *BltEfiUgaBltBufferToVideo* operation to copy *Buffer1* to video. 3. Call **Blt()** with *BltEfiUgaVideoToBltBuffer* to store display to *Buffer2*. 4. Compare *Buffer1* and *Buffer2*. Each pixel should be the same.
5.6.3.3.7	0x9efc6f31, 0x1cb1, 0x458f, 0x9a, 0x15, 0xe3, 0x47, 0xa8, 0x36, 0x8d, 0xd8	**EFI_UGA_DRAW_PROTOCOL.Blt - Blt()** with *EfiUgaVideoToVideo* operation returns **EFI_SUCCESS**.	1. Call **Blt()** to output a blue rectangle on screen and call **Blt()** with *EfiUgaVideoToVideo* operation to copy source display rectangle to destination display destination.
5.6.3.3.8	0x09777d6a, 0x14aa, 0x41eb, 0xb8, 0xbc, 0x0d, 0xcb, 0x90, 0xf6, 0x22, 0xbc	**EFI_UGA_DRAW_PROTOCOL.Blt - Blt()** with *EfiUgaVideoToVideo* operation returns the same contents between source display rectangle and destination display destination.	1. Call **Blt()** to output a blue rectangle on screen and call **Blt()** with *EfiUgaVideoToVideo* operation to copy source display rectangle to destination display destination. 2. Call **Blt()** with *BltEfiUgaVideoToBltBuffer* to store source display rectangle to *Buffer1*. 3. Call **Blt()** with *BltEfiUgaVideoToBltBuffer* to store destination display rectangle to *Buffer2*. 4. Compare *Buffer1* and *Buffer2*. Each pixel should be same.
5.6.3.3.9	0xa077b57a, 0x2d0f, 0x4d26, 0x9e, 0x41, 0x13, 0xb2, 0x6e, 0x28, 0xed, 0xe7	**EFI_UGA_DRAW_PROTOCOL.Blt - Blt()** with invalid *BltOperation* returns **EFI_INVALID_PARAMETER**.	1. Call **Blt()** with invalid *BltOperation*. The return code should be **EFI_INVALID_PARAMETER**.

B.3 EFI_SCSI_PASS_THRU_PROTOCOL Test

Reference Document:

UEFI Specification, EFI_SCSI_PASS_THRU_PROTOCOL Section.

B.3.1 PassThru()

Number	GUID	Assertion	Test Description
5.9.1.1.1	0x23512eed, 0x301c, 0x493d, 0x8a, 0x03, 0xa6, 0xd4, 0x22, 0x1b, 0xee, 0x9c	**EFI_SCSI_PASS_THRU_PROTOCOL.PassThru** - Invoks **PassThru()** with **NULL** *Event* will verify interface correctness by returning **EFI_SUCCESS**.	Call **GetNextDevice()** to get valid *Target* and *Lun*. Use the *Target* and *Lun* gotten before to call **PassThru()** with **NULL** *Event*. The return status should be **EFI_SUCCESS**.
5.9.1.1.2	0x00718d3e, 0x788a, 0x4882, 0x80, 0xf7, 0x71, 0xb4, 0xf0, 0xcf, 0x6b, 0x30	**EFI_SCSI_PASS_THRU_PROTOCOL.PassThru** - Invoks **PassThru()** with *Event* will verify interface correctness by returning **EFI_SUCCESS**.	Call **GetNextDevice()** to get valid *Target* and *Lun*. Use the *Target* and *Lun* gotten before to call **PassThru()** with *Event*. The return status should be **EFI_SUCCESS** and the event should be invoked.
5.9.1.1.3	0x4751f323, 0x0687, 0x47b6, 0xbe, 0x16, 0x57, 0x73, 0xc1, 0xa3, 0x6d, 0x28	**EFI_SCSI_PASS_THRU_PROTOCOL.PassThru** - Calling **PassThru()** with with too long a *TransferLength* returns **EFI_BAD_BUFFER_SIZE**.	Call **PassThru()** with the *TransferLength* larger than the SCSI controller can handle. It should return **EFI_BAD_BUFFER_SIZE** and the *TransferLength* will be updated to the length that SCSI controller can handle.
5.9.1.1.4	0x831dd6e6, 0x1960, 0x4c27, 0xab, 0xef, 0x2c, 0x3c, 0x0d, 0x58, 0x68, 0x7f	**EFI_SCSI_PASS_THRU_PROTOCOL.PassThru** - Calling **PassThru()** with an invalid *Target* returns **EFI_INVALID_PARAMETER**.	Call **PassThru()** with an invalid *Target*. It should return **EFI_INVALID_PARAMETER**.
5.9.1.1.5	0x8dc5b229, 0xb838, 0x4a90, 0xb3, 0x50, 0x81, 0x3c, 0x42, 0xd4, 0x85, 0x44	**EFI_SCSI_PASS_THRU_PROTOCOL.PassThru** - Calling **PassThru()** with an invalid *Lun* returns **EFI_INVALID_PARAMETER**.	Call **PassThru()** with an invalid *Lun*. It should return **EFI_INVALID_PARAMETER**.
5.9.1.1.6	0xf57be290, 0x0aa4, 0x4e8e, 0x8d, 0x09, 0xe2, 0xce, 0xbc, 0x73, 0xc0, 0x77	**EFI_SCSI_PASS_THRU_PROTOCOL.PassThru** - Calling **PassThru()** with an invalid *ScsiRequestPacket* content returns **EFI_INVALID_PARAMETER**.	Call **PassThru()** with an invalid *ScsiRequestPacket* content. It should return **EFI_INVALID_PARAMETER**.

B.3.2 GetNextDevice()

Number	GUID	Assertion	Test Description
5.9.1.2.1	0x4eda0492, 0x1eb2, 0x4022, 0x87, 0x1f, 0xd3, 0x95, 0x58, 0x20, 0x1d, 0x01	**EFI_SCSI_PASS_THRU_PROTOCOL.GetnextDevice – GetnextDevice()** retrieves the list of legal Target IDs and LUNs for SCSI devices on a SCSI channel.	Call **GetNextDevice()** with *Target*'s value of 0xFFFFFFFF to get the first SCSI device present on a SCSI channel. Use the *Target* and **Lun** which were returned to get the next SCSI device until the end. Every call of **GetNextDevice()** should return **EFI_SUCCESS** except the last one. The last call should return **EFI_NOT_FOUND**.
5.9.1.2.2	0x3661f513, 0xd0ea, 0x47f2, 0x8a, 0xb7, 0xaa, 0xb4, 0x6b, 0xcd, 0x93, 0xa0	**EFI_SCSI_PASS_THRU_PROTOCOL.GetnextDevice – GetnextDevice()** uses former *Target* and *Lun* to get next device.	Call **GetNextDevice()** with *Target*=0xFFFFFFFF to get the first device. Then call it again to get the next device. Use the *Target* and *Lun* return from the first call to call the function. It should return **EFI_INVALID_PARAMETER**.
5.9.1.2.3	0xd2d48206, 0xf2dd, 0x40b3, 0xaf, 0x67, 0xe9, 0xae, 0x60, 0xc7, 0x2b, 0x9f	**EFI_SCSI_PASS_THRU_PROTOCOL.GetnextDevice** - Call **GetNextDevice()** with an invalid *Target*.	Call **GetNextDevice()** with an invalid *Target*. It should return **EFI_INVALID_PARAMETER**.
5.9.1.2.4	0xe7e16f25, 0xca2d, 0x4de5, 0x9f, 0xf4, 0xe4, 0xcc, 0xac, 0x9d, 0xf6, 0x90	**EFI_SCSI_PASS_THRU_PROTOCOL.GetnextDevice** - Call **GetNextDevice()** with an invalid **Lun**.	Call **GetNextDevice()** with an invalid *Lun*. It should return **EFI_INVALID_PARAMETER**.

B.3.3 BuildDevicePath()

Number	GUID	Assertion	Test Description
5.9.1.3.1	0x93c4def4, 0x7854, 0x42b3, 0x81, 0xbc, 0xa0, 0x4c, 0x0f, 0xd7, 0xb1, 0x93	**EFI_SCSI_PASS_THRU_PROTOCOL.BuildDevicePath** - Invoks **BuildDevicePath()** will verify interface correctness by returning **EFI_SUCCESS**.	Call **GetNextDevice()** to get the first device's *Target* and *Lun*. Call **BuildDevicePath()** with valid parameter. Free the *DevicePath*. It should return **EFI_SUCCESS**.

Number	GUID	Assertion	Test Description
5.9.1.3.2	0xd4c6c164, 0x0198, 0x47c6, 0xb7, 0xef, 0x01, 0x0c, 0x47, 0x42, 0xc9, 0x88	**EFI_SCSI_PASS_THRU_PROTOCOL.BuildDevicePath** - Calling **BuildDevicePath()** with an invalid *Target* returns **EFI_NOT_FOUND**.	Call **BuildDevicePath()** with an invalid *Target*. It should return **EFI_NOT_FOUND**.
5.9.1.3.3	0xec077c7f, 0x114a, 0x41b1, 0x94, 0x83, 0x5b, 0x38, 0x10, 0xdb, 0xc4, 0x00	**EFI_SCSI_PASS_THRU_PROTOCOL.BuildDevicePath** - Calling **BuildDevicePath()** with an invalid *Lun* returns **EFI_NOT_FOUND**.	Call **BuildDevicePath()** with an invalid *Lun*. It should return **EFI_NOT_FOUND**.
5.9.1.3.4	0x8a1ce910, 0x8a20, 0x4a72, 0xb7, 0x05, 0xb8, 0x09, 0x70, 0xc7, 0xdf, 0xd3	**EFI_SCSI_PASS_THRU_PROTOCOL.BuildDevicePath** - Calling **BuildDevicePath()** with **NULL** *DevicePath* returns **EFI_INVALID_PARAMETER**.	Call **BuildDevicePath()** with *a*. **NULL** *DevicePath*. It should return **EFI_INVALID_PARAMETER**.

B.3.4 GetTargetLun()

Number	GUID	Assertion	Test Description
5.9.1.4.1	0x8d06f9c5, 0xd470, 0x4b31, 0xbe, 0xb9, 0x73, 0x3e, 0x5d, 0x8f, 0xf4, 0xcb	**EFI_SCSI_PASS_THRU_PROTOCOL.GetTargetLun** - Invoks **GetTargetLun()** will verify interface correctness by returning **EFI_SUCCESS**.	Call **GetNextDevice()** and **GetTargetLun()** to get the valid *DevicePath*. Use this *DevicePath* to call **GetTargetLun()**.The return code should be **EFI_SUCCESS**.
5.9.1.4.2	0x462c4098, 0xfd65, 0x4005, 0x8e, 0xdb, 0x7b, 0xb5, 0x95, 0x65, 0xc5, 0x11	**EFI_SCSI_PASS_THRU_PROTOCOL.GetTargetLun** - Invoks **GetTargetLun()** with **NULL** *DevicePath* returns **EFI_INVALID_PARAMETER**.	Call **GetTargetLun()** with **NULL** *DevicePath*. It should return **EFI_INVALID_PARAMETER**.
5.9.1.4.3	0x884c336a, 0xeffd, 0x45b3, 0xb5, 0xcb, 0xc5, 0x50, 0x2a, 0xfa, 0xcf, 0x3f	**EFI_SCSI_PASS_THRU_PROTOCOL.GetTargetLun** - Invoks **GetTargetLun()** with **NULL** *Target* returns **EFI_INVALID_PARAMETER**.	Call **GetTargetLun()** with **NULL** *Target*. It should return **EFI_INVALID_PARAMETER**.
5.9.1.4.4	0x842b366f, 0x035e, 0x46a7, 0x8f, 0x07, 0x45, 0xd8, 0xd1, 0xe1, 0xe1, 0x72	**EFI_SCSI_PASS_THRU_PROTOCOL.GetTargetLun** - Invoks **GetTargetLun()** with **NULL** *Lun* returns **EFI_INVALID_PARAMETER**.	Call **GetTargetLun()** with **NULL** *Lun*. It should return **EFI_INVALID_PARAMETER**.
5.9.1.4.5	0xf29750b2, 0xd353, 0x4baa, 0x8a, 0x44, 0x29, 0xc2, 0x4e, 0xe8, 0x49, 0x43	**EFI_SCSI_PASS_THRU_PROTOCOL.GetTargetLun** - Calling **GetTargetLun()** with unsupported *DevicePath* returns **EFI_UNSUPPORTED**.	Call **GetTargetLun()** with unsupported *DevicePath*. It should return **EFI_UNSUPPORTED**.

B.3.5 ResetChannel()

Number	GUID	Assertion	Test Description
5.9.1.5.1	0x8af96e89, 0x2209, 0x47d9, 0x9b, 0x84, 0xa1, 0xf6, 0xf2, 0xd1, 0x8a, 0x6b	`EFI_SCSI_PASS_THRU_PROTOCOL.ResetChannel` - Invoks `ResetChannel()` will verify interface correctness via return code of `EFI_SUCCESS` or `EFI_UNSUPPORTED`.	Call `ResetChannel()`.The return code should be `EFI_SUCCESS` or `EFI_UNSUPPORTED`.

B.3.6 ResetTarget()

Number	GUID	Assertion	Test Description
5.9.1.6.1	0xbac42d29, 0x75cc, 0x4b9b, 0xa3, 0x16, 0xdf, 0x11, 0xca, 0x7c, 0xf1, 0xe4	`EFI_SCSI_PASS_THRU_PROTOCOL.ResetTarget` - Invoks `ResetTarget()` will verify interface correctness via return code of `EFI_SUCCESS` or `EFI_UNSUPPORTED`.	Call `GetNextDevice()` to get valid *`Target`* and *`Lun`*. Use the *`Target`* and *`Lun`* gotten before to call `ResetTarget()`.The return code should be `EFI_SUCCESS` or `EFI_UNSUPPORTED`.
5.9.1.6.2	0x04296f40, 0xe48b, 0x4b5c, 0xb2, 0xcf, 0x49, 0x25, 0xf0, 0x98, 0x5d, 0x82	`EFI_SCSI_PASS_THRU_PROTOCOL.ResetTarget` - Calling `ResetTarget()` with an invalid *`Target`* returns `EFI_INVALID_PARAMETER`.	Call `GetResetTarget()` with an invalid *`Target`*. It should return `EFI_INVALID_PARAMETER`.
5.9.1.6.3	0xc75f3592, 0xee1a, 0x43a3, 0xaa, 0x9b, 0x08, 0x16, 0x9e, 0xca, 0xa6, 0x93	`EFI_SCSI_PASS_THRU_PROTOCOL.ResetTarget` - Calling `ResetTarget()` with an invalid *`Lun`* returns `EFI_INVALID_PARAMETER`.	Call `GetResetTarget()` with an invalid *`Lun`*. It should return `EFI_INVALID_PARAMETER`.