# Protocols EFI Driver Model Test

## 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.

## EFI\_PLATFORM\_DRIVER\_OVERRIDE\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_PLATFORM\_DRIVER\_OVERRIDE\_PROTOCOL Section.

### 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. |
| 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. |

### 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. |

### 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 DriverImagePathnot 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. |

## �EFI\_BUS\_SPECIFIC\_DRIVER\_OVERRIDE\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_BUS\_SPECIFIC\_DRIVER\_OVERRIDE\_PROTOCOL Section.

### 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. |

## �EFI\_DRIVER\_CONFIGURATION\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_BUS\_SPECIFIC\_DRIVER\_OVERRIDE\_PROTOCOL Section.

### 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. |

### 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. |

### 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. |

## EFI\_DRIVER\_DIAGNOSTICS\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_DRIVER\_DIAGNOSTICS\_PROTOCOL Section.

### 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 |
| 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. |

## EFI\_DRIVER\_DIAGNOSTICS2\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_DRIVER\_DIAGNOSTICS2\_PROTOCOL Section.

### 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 |
| 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. |
| 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. |

## EFI\_COMPONENT\_NAME\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_COMPONENT\_NAME\_PROTOCOL Section.

### 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. |
| 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. |

### 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. |
| 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. |
| 5.5.7.2.7 | 0x27a4781a, 0xe85a, 0x4714, 0xab, 0x9a, 0x67, 0xc1, 0x01, 0x38, 0x5e, 0x83 | EFI\_COMPONENT\_NAME\_PROTOCOL.GetControllerName – GetGontrollerName() 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 – GetGontrollerName() 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 – GetGontrollerName() 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 – GetGontrollerName() 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. |
| 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. |

## EFI\_COMPONENT\_NAME2\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_COMPONENT\_NAME2\_PROTOCOL Section.

### 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 |
| 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 |

### 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 |
| 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 |

## EFI\_PLATFORM\_TO\_DRIVER\_CONFIGURATION\_PROTOCOL

Reference Document:

*UEFI Specification*, EFI\_PLATFORM\_TO\_DRIVER\_CONFIGURATION\_PROTOCOL Section.

### 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 |

### 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 |

### 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 |

## 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. . |

## EFI\_ADAPTER\_INFORMATION\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_ADAPTER\_INFORMATION\_PROTOCOL Section.

### 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()returnsEFI\_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. |
| 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()returnsEFI\_INVALID\_PARAMETERS  with NULL InformationBlockSize. | Call GetInformation ()with NULL InformationBlockSize, the return status should be EFI\_INVALID\_PARAMETERS. |

### 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. |
| 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. |

### 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. |