# Services Boot Services

## Event, Timer, and Task Priority Services Test

Reference Document:

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

* 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. |

### 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. |
| 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. |

### 

### 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. |

### 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. |

### 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. |
| 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. |

### 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. |
| 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. |

### 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. |
| 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. |

### 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. |

### 

### 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. |

### 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. |
| 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. |
| 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. |
| 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. |

## Memory Allocation Services Test

Reference Document:

*UEFI Specification*, Memory Allocation Services Section.

* 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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\_PARAMETER |
| 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. |

### 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. |
| 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. |

### 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. |
| 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(). |
| 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. |
| 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(). |
| 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 MapKeys 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. |
| 5.1.2.3.21 | 0x3093039c, 0xdff7, 0x4097, 0x9a, 0x36, 0xd7, 0x96, 0x82, 0x81, 0xc1, 0x46 | BS.GetMemoryMap – GetMemoryMap() returns different MapKeys 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. |
| 5.1.2.3.22 | 0x284e0cc8, 0x913a, 0x4e8b, 0xbd, 0x05, 0xb4, 0xc8, 0xe1, 0x95, 0xc3, 0x69 | BS.GetMemoryMap – GetMemoryMap() returns different MapKeys 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. |

### 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. |
| 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\_PARAMETER. |
| 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\_PARAMETER. |

### 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. |

## Protocol Handler Services Test

Reference Document:

*UEFI Specification*, Protocol Handler Services Section.

* 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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 |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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\_SMALLEFI\_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\_SMALLEFI\_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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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). |
| 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). |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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). |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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). |
| 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). |
| 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). |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |

### 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\_FOUNDwith  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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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\_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. 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. |
| 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\_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. |
| 5.1.3.11.79 | 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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\_PROTOCOLand 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. |
| 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. |
| 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\_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. 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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.InstallMultipleProt  ocolInterfaces –  InstallMultipleProtoco  **lInterfaces()** returns  EFI\_INVALID\_PARAMETER  when handle is **NULL** | 1. Call  InstallMultipleProto  **colInterfaces()** with an  NULL handle. The return  code should be  EFI\_INVALID\_PARAMETE  **R**. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 5.1.3.17.83 | 0xe05ca4d7, 0xa705, 0x4270, 0x99, 0xbb, 0x10, 0x8d, 0x8c, 0x1f, 0xc8, 0x0c | BS.UninstallMultipleProtocolInterfaces – UninstallMultipleProtocolInterfaces() 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.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. The handle should still exist. |
| 5.1.3.17.85 | 0x28749f75, 0xc7c3, 0x4e55, 0xbc, 0xa1, 0xb2, 0xfb, 0x80, 0x77, 0x26, 0x0c | 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. The handle should still exist. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

## Image Services Test

Reference Document:

*UEFI Specification*, Image Services Section.

* 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. |

### 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. |
| 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**. |
| 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**. |
| 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**. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

## EFI\_IMAGE\_ENTRY\_POINT

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

### 

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |

## Misc Boot Services Test

Reference Document:

*UEFI Specification*, Miscellaneous Boot Services Section.

* 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |

### 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. |

### 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. |
| 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(). |
| 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(). |
| 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(). |
| 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. |

### 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |

### 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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**. |
| 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. |
| 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. |
| 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**. |
| 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**. |

### 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. |
| 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. |