# Network Protocols UDP and MTFTP

## EFI\_UDP4\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_UDP4\_PROTOCOL Section.

### GetModeData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.1.1.1 | 0xfc4d1b7b, 0x4abd, 0x47d3, 0xbd, 0x64, 0xe0, 0x98, 0x86, 0x29, 0x73, 0xec | EFI\_UDP4\_PROTOCOL.GetModeData() **–** invokes GetModeData() to get all mode data before configuration. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**GetModeData() to get all mode data. The return status should be EFI\_NOT\_STARTED.  3. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.1.2 | 0x15c32ffb, 0x2cdf, 0x4b5b,  0xab, 0x3e, 0x5a, 0xed, 0x7f, 0xc5, 0x25, 0xe7 | EFI\_UDP4\_PROTOCOL.GetModeData() – invokes GetModeData() to get EFI\_UDP4\_PROTOCOL mode data before configuration. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.GetModeData() to get EFI\_UDP4\_PROTOCOL mode data. The return status should be EFI\_NOT\_STARTED.  3. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.1.3 | 0xcdcd0bb9, 0x455a, 0x4525, 0xb8, 0xf2, 0x0e, 0xe0, 0x4b, 0xaa, 0x80, 0x14 | EFI\_UDP4\_PROTOCOL.GetModeData() – invokes GetModeData() to get EFI\_IP4\_PROTOCOL mode data before configuration. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.GetModeData() to get EFI\_IP4\_PROTOCOL mode data. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.1.4 | 0xcc19f3f7, 0x80b9, 0x46e8, 0xb2, 0xaa, 0xb6, 0xdd, 0x81, 0x66, 0xd8, 0x93 | EFI\_UDP4\_PROTOCOL.GetModeData() – invokes GetModeData() to get EFI\_MANAGED\_NETWORK\_PROTOCOL mode data before configuration. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.GetModeData() to get EFI\_MANAGED\_NETWORK\_PROTOCOL mode data. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.1.5 | 0xd291d441, 0x2d3b, 0x4575, 0xa3, 0xf3, 0x05, 0xe1, 0x5a, 0x34, 0x62, 0xc0 | EFI\_UDP4\_PROTOCOL.GetModeData() – invokes GetModeData() to get EFI\_SIMPLE\_NETWORK\_PROTOCOL mode data before configuration. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.GetModeData() to get EFI\_SIMPLE\_NETWORK\_PROTOCOL mode data. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.1.6 | 0xf28448b0, 0xd525, 0x40f7, 0x92, 0xf1, 0xed, 0x6d, 0xaa, 0x59, 0xe4, 0xb4 | EFI\_UDP4\_PROTOCOL.GetModeData() **–** invokes GetModeData() to get all mode data after configuration. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure() to configure the new instance.  3 Call EFI\_UDP4\_PROTOCOL**.**GetModeData() to get all mode data. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.1.7 | 0x182f712c, 0x1b2a, 0x4850, 0xbd, 0x78, 0xa6, 0xe6, 0xb6, 0xf6, 0x73, 0x54 | EFI\_UDP4\_PROTOCOL.GetModeData() – invokes GetModeData() to get EFI\_UDP4\_PROTOCOL mode data after configuration. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure() to configure the new instance.  3 Call EFI\_UDP4\_PROTOCOL.GetModeData() to get EFI\_UDP4\_PROTOCOL mode data. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.1.8 | 0x8aa1ebeb, 0xb735, 0x421e, 0x92, 0x1d, 0xf8, 0x76, 0xd2, 0xae, 0xdf, 0x1c | EFI\_UDP4\_PROTOCOL.GetModeData() – invokes GetModeData() to get EFI\_IP4\_PROTOCOL mode data after configuration. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure() to configure the new instance.  3 Call EFI\_UDP4\_PROTOCOL.GetModeData() to get EFI\_IP4\_PROTOCOL mode data. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.1.9 | 0xba1c7d49, 0x4490, 0x42e1, 0xa8, 0x92, 0xc3, 0x61, 0xef, 0x5d, 0x94, 0x79 | EFI\_UDP4\_PROTOCOL.GetModeData() – invokes GetModeData() to get EFI\_MANAGED\_NETWORK\_PROTOCOL mode data after configuration. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure() to configure the new instance.  3 Call EFI\_UDP4\_PROTOCOL.GetModeData() to get EFI\_MANAGED\_NETWORK\_PROTOCOL mode data. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.1.10 | 0x5df96df3, 0x6404, 0x4486, 0xb6, 0xb7, 0x00, 0xb9, 0x2d, 0x81, 0x21, 0x26 | EFI\_UDP4\_PROTOCOL.GetModeData() – invokes GetModeData() to get EFI\_SIMPLE\_NETWORK\_PROTOCOL mode data after configuration. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure() to configure the new instance.  3 Call EFI\_UDP4\_PROTOCOL.GetModeData() to get EFI\_SIMPLE\_NETWORK\_PROTOCOL mode data. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |

### Configure()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.1.2.1 | 0x13a8fd73, 0x6b66, 0x4418, 0x85, 0x4c, 0xda, 0x63, 0xff, 0x42, 0x75, 0x4f | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() with a StationAddressvalue of a multicast address. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance with a StationAddressvalue of a multicast address. The return status should be EFI\_INVALID\_PARAMETER.  3. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.2 | 0xd8b6f8bd, 0x1ba8, 0x48c1, 0x90, 0x30, 0x5a, 0x37, 0x18, 0x0c, 0x06, 0x01 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() with an invalid SubnetMask value. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance with an invalid SubnetMask value. The return status should be EFI\_INVALID\_PARAMETER.  3. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.3 | 0xb4a98a30, 0x35e9, 0x4460, 0x81, 0x5d, 0x42, 0x33, 0x7c, 0x17, 0x6c, 0x44 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to reconfigure the *AcceptPromiscuous*before the instance has been stopped or reset. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Configure() to reconfigure the AcceptPromiscuous. The return status should be EFI\_ALREADY\_STARTED.  4. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.23 | 0x349fc21a, 0x37db, 0x406e, 0xbd, 0xc8, 0xf6, 0x12, 0x2c, 0xa9, 0xe9, 0xfc | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() with the parameter *RemoteAddress* being a multicast address.The return status should be EFI\_INVALID\_PARAMETER. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()with the parameter *RemoteAddress* being a multicast address.The return status should be EFI\_INVALID\_PARAMETER.  3. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL**.**DestroyChild**()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.4 | 0xa36f507b, 0x7526, 0x441e, 0xaf, 0x48, 0x4a, 0xc4, 0xf4, 0x31, 0xe6, 0xbd | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to reconfigure the *AcceptBroadcast* before the instance has been stopped or reset. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Configure()to reconfigure the AcceptBroadcast. The return status should be EFI\_ALREADY\_STARTED.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.5 | 0xac4cf23e, 0x0c5e, 0x4299, 0xb4, 0x29, 0xc8, 0x83, 0xe7, 0xe6, 0x73, 0xb8 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to reconfigure the *AcceptAnyPort* before the instance has been stopped or reset. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Configure()to reconfigure the AcceptAnyPort. The return status should be EFI\_ALREADY\_STARTED.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.6 | 0xc08bfbab, 0x0cde, 0x4332, 0x86, 0x86, 0x42, 0x52, 0xdc, 0x50, 0x48, 0xcc | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to reconfigure the *AllowDuplicatePort* before the instance has been stopped or reset. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Configure()to reconfigure the AllowDuplicatePort. The return status should be EFI\_ALREADY\_STARTED.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.7 | 0x66544950, 0x16ff, 0x4854, 0x9c, 0x09, 0x45, 0x84, 0x29, 0x2d, 0x7c, 0x51 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to reconfigure the *UseDefaultAddress* before the instance has been stopped or reset. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Configure()to reconfigure the UseDefaultAddress. The return status should be EFI\_ALREADY\_STARTED.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.24 | 0xee87c393, 0xf728, 0x46b9, 0xb1, 0x31, 0x58, 0xc3, 0xdd, 0x5e, 0x18, 0x34 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure()when UdpConfigData*.*AllowDuplicatePort is FALSE and UdpConfigData.StationPort is already used by other instance. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()when UdpConfigData.AllowDuplicatePort is FALSE and UdpConfigData.StationPort is already used by other instance. The return status should be EFI\_ACCESS\_DENIED**.**  3. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.8 | 0xbe8ab604, 0x1c84, 0x4a80, 0xb6, 0x9a, 0x43, 0xfd, 0xf8, 0x94, 0x5e, 0xf2 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to test the function of transmitting a packet. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()to transmit a packet and verify if it is successful.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.9 | 0xddbba5ba, 0x678b, 0x426e, 0x87, 0xa8, 0x8c, 0x1b, 0xde, 0x5b, 0x36, 0x96 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to test that function of receiving a packet. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Receive()to receive a packet and verify if it is successful.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.10 | 0xefe91110, 0x4e6e, 0x4e07, 0xa7, 0xec, 0x09, 0x74, 0xb7, 0xe3, 0x03, 0x87 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to reconfigure the *TypeOfService*before the instance has been stopped or reset. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Configure()to reconfigure the *TypeOfService*. The return status should be EFI\_SUCCESS.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.11 | 0xc6f4f65f, 0x9a98, 0x4d6e, 0xaf, 0xae, 0xe9, 0x87, 0xf9, 0xb4, 0xb4, 0x9c | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to reconfigure the *TimeToLive*before the instance has been stopped or reset. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Configure()to reconfigure the *TimeToLive*. The return status should be EFI\_SUCCESS.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.12 | 0xe6313038, 0x43f2, 0x4cbe, 0xb8, 0x61, 0xa4, 0x1b, 0x6e, 0x3d, 0x58, 0x91 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to reconfigure the *DoNotFragment*before the instance has been stopped or reset. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Configure()to reconfigure the *DoNotFragment*. The return status should be EFI\_SUCCESS.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.13 | 0x2c81abe0, 0xcf2a, 0x42d0, 0xb4, 0xe3, 0x59, 0x9e, 0x9e, 0x2f, 0x60, 0x6a | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to reconfigure the *ReceiveTimeout*before the instance has been stopped or reset. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Configure()to reconfigure the *ReceiveTimeout*. The return status should be EFI\_SUCCESS.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.14 | 0x798d02e5, 0x0810, 0x462c, 0x8f, 0xba, 0xe9, 0x32, 0xfb, 0x9d, 0x84, 0x85 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to reconfigure the *TransmitTimeout*before the instance has been stopped or reset. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Configure()to reconfigure the *TransmitTimeout*. The return status should be EFI\_SUCCESS.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.15 | 0xbe426d4c, 0x8242, 0x4a4e, 0x8d, 0x7d, 0x58, 0xe0, 0x93, 0x92, 0x77, 0x7c | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() with the parameter *AcceptBroadcast* set to **FALSE**. Check that it can not receive broadcast packet. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance with the parameter *AcceptBroadcast* set to **FALSE**. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Receive()and check it can not receive broadcast packet.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.16 | 0xb50d8d35, 0xc0c9, 0x4955, 0x94, 0x13, 0xf7, 0x0a, 0x39, 0x2d, 0xa3, 0x0f | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() with the parameter *AcceptBroadcast* set to **TRUE**. Check that it can receive broadcast packet successfully. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance with the parameter *AcceptBroadcast* set to **TRUE**. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Receive()and check that it can receive broadcast packet successfully.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.17 | 0x4881a297, 0x3afc, 0x4324, 0xa5, 0x8f, 0xcb, 0x02, 0x64, 0xe5, 0xbd, 0x5e | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() with the parameter *AcceptPromiscuous*set to **FALSE**. Check that it can not receive packet to other unicast MACs than its own. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance with the parameter *AcceptPromiscuous*set to **FALSE**. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Receive()and check that it can not receive packet to other unicast MACs than its own.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.18 | 0x066131ca, 0xa6e4, 0x478b, 0x9a, 0xca, 0x05, 0x93, 0xfc, 0xc7, 0xfd, 0x4b | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() with the parameter *AcceptPromiscuous*set to **TRUE**. Check that it can receive packet to other unicast MACs than its own. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance with the parameter *AcceptPromiscuous*set to **TRUE**. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Receive()and check that it can receive packet to other unicast MACs than its own.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.19 | 0x2867badf, 0x1696, 0x40a1, 0xb8, 0x40, 0x00, 0x4c, 0x79, 0xed, 0xc7, 0xf3 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() with the parameter *AcceptAnyPort*set to **FALSE**. Check that it can not receive packet to other port. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance with the parameter *AcceptAnyPort*set to **TRUE**. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Receive()and check that it can not receive packet to other port.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.20 | 0x17d43b3d, 0x9187, 0x4515, 0x83, 0x94, 0x13, 0xdf, 0xf9, 0x35, 0xf4, 0x9e | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() with the parameter *AcceptAnyPort*set to **TRUE**. Check that it can receive packet to other port. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance with the parameter *AcceptAnyPort*set to **TRUE**. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Receive()and check that it can receive packet to other port.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.21 | 0x08c86675, 0x7018, 0x418d, 0xb4, 0x3d, 0x36, 0xdc, 0xc5, 0x8b, 0xdc, 0x88 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to check if the parameter *TypeOfService*can effect the sending out of the packet. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance with the parameter *TypeOfService*set to **1**. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()and check that it can transmit the packet successfully.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.2.22 | 0x4fb07a34, 0xc2ab, 0x40c1, 0x8a, 0x26, 0x42, 0x6d, 0x54, 0x32, 0x3a, 0xa4 | EFI\_UDP4\_PROTOCOL. Configure() **–** invokes Configure() to check if the parameter *TimeToLive*can effect the sending out of the packet . | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Configure()to configure the new EFI\_UDP4\_PROTOCOL instance with the parameter *TimeToLive*set to **111**. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()and check that it can transmit the packet successfully.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |

### Groups()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.1.3.1 | 0x90ff05c9, 0xea78, 0x4359, 0x95, 0xc0, 0x4d, 0x09, 0x7b, 0xa2, 0xcf, 0x14 | EFI\_UDP4\_PROTOCOL.Groups() **–** invokes Groups() with a JoinFlag value of TRUE and a MulticastAddress value of NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Groups() with a *JoinFlag* value of **TRUE** and a *MulticastAddress* value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.3.2 | 0x4e1cabfe, 0x2dda, 0x4e0c, 0xbd, 0xbc, 0x5f, 0xfc, 0x77, 0x42, 0xf8, 0x0f | EFI\_UDP4\_PROTOCOL.Groups() **–** invokes Groups() with a *JoinFlag* value of **TRUE** and a *\*MulticastAddress*value of an invalid multicast address. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Groups() with a *JoinFlag* value of **TRUE** and a *\*MulticastAddress*value of an invalid multicast address. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.3.3 | 0xf1018cf8, 0xd8ba, 0x4fa1, 0x82, 0xec, 0x64, 0x52, 0x06, 0x9a, 0x4a, 0xa7 | EFI\_UDP4\_PROTOCOL.Groups() **–** invokes Groups() when the group address is not in the group table ,while *JoinFlag* is **FALSE**. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Groups() to join a group address into the group table. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_PROTOCOL**.**Groups() to leave an group address which is not in the group table The return status should be **EFI\_NOT\_FOUND**.  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.3.8 | 0x86b5bd38, 0x04ae, 0x4a44, 0xbe, 0x0d, 0x1d, 0x7f, 0x32, 0x0f, 0x46, 0xf8 | EFI\_UDP4\_PROTOCOL.Groups() **–** invokes Groups() when the EFI UDPv4 protocol instance has not been started. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.Groups()**when the EFI UDPv4 protocol instance has not been started.  3. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.3.9 | 0xde218295, 0x6dec, 0x4c7f, 0x8c, 0x02, 0xc9, 0x46, 0xea, 0x64, 0x59, 0xd6 | EFI\_UDP4\_PROTOCOL.Groups() **–** invokes Groups() when the group address is already in the group table when *JoinFlag* is **FALSE.** | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Groups() to join a group address into the group table. The return status should be EFI\_SUCCESS.  3. Call EFI\_UDP4\_PROTOCOL**.**Groups() when the group address is already in the group table when *JoinFlag* is **FALSE.**The return status should be **EFI\_ALREADY\_STARTED**.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.3.4 | 0x101a001f, 0x547e, 0x4e1b, 0xae, 0xf6, 0x7d, 0x35, 0x27, 0xb1, 0x23, 0x6f | EFI\_UDP4\_PROTOCOL.Groups() **–** invokes Groups() to join a group address and call Receive() to check that it can receive UDP packets to the group IP. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Groups() to join a group address into the group table. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_PROTOCOL**.**Receive() to receive the packets and check that it is successful.  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.3.5 | 0x22561bd0, 0x47ba, 0x4240, 0x96, 0x3a, 0x2a, 0xaf, 0x83, 0x5b, 0xda, 0x72 | EFI\_UDP4\_PROTOCOL.Groups() **–** invokes Groups() to join two multicast group address and call Receive() to check if it can receive UDP packets to either of the groups. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Groups() to join two group address into the group table. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_PROTOCOL**.**Receive() to check it can receive UDP packets to either of the groups.  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.3.6 | 0x7fcefed3, 0x6e40, 0x4ed8, 0xa4, 0x41, 0x83, 0x7f, 0x5e, 0x13, 0x06, 0x62 | EFI\_UDP4\_PROTOCOL.Groups() – invokes Groups() to leave a specified group. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Groups() to join a specified group address into the group table. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_PROTOCOL**.**Groups() to leave the group joined in step 3. The return status should be EFI\_SUCCESS.  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.3.7 | 0x06e97222, 0x1858, 0x469a, 0xa8, 0x19, 0x25, 0xd7, 0x1a, 0x15, 0xc3, 0x68 | EFI\_UDP4\_PROTOCOL.Groups() **–** invokes Groups() to leave all multicast groups with a *MulticastAddress* value of NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Groups() to join two group address into the group table. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_PROTOCOL**.**Groups() to leave all multicast groups with a *MulticastAddress* value of NULL. The return status should be EFI\_SUCCESS.  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |

### Routes()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.1.4.1 | 0xedcd02f7, 0x3b78, 0x4186, 0x9d, 0x14, 0x52, 0x92, 0x6b, 0x85, 0x73, 0x08 | EFI\_UDP4\_PROTOCOL.Routes() **–** invokes Routes() with a SubnetAddress value of NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Routes()with a SubnetAddress value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.4.2 | 0xf0bedea5, 0x05bf, 0x4ab9, 0x89, 0xb3, 0xdf, 0xd9, 0x8e, 0x08, 0xe4, 0xdd | EFI\_UDP4\_PROTOCOL.Routes() – invokes Routes() with a SubnetMask value of NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Routes()with a SubnetMaskvalue of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.4.3 | 0x377694cc, 0x9254, 0x4197, 0x92, 0x6c, 0x26, 0x58, 0x5c, 0xde, 0xc9, 0x4c | EFI\_UDP4\_PROTOCOL.Routes() **–** invokes Routes() with a GatewayAddressvalue of NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Routes()with a GatewayAddressvalue of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.4.4 | 0xc694ffe9, 0xef16, 0x47f4, 0x86, 0x89, 0x34, 0x6c, 0x80, 0xb1, 0x59, 0x54 | EFI\_UDP4\_PROTOCOL.Routes() **–** invokes Routes() with a *\**SubnetMaskvalue of an invalid subnet mask. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Routes()with a *\**SubnetMaskvalue of an invalid subnet mask. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.4.5 | 0x601c9a17, 0x1da6, 0x45bc, 0xbb, 0xdc, 0xf8, 0x92, 0xdc, 0xe3, 0x43, 0x04 | EFI\_UDP4\_PROTOCOL.Routes() **–** invokes Routes() with a *\**GatewayAddressvalue of an invalid unicast IP address. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Routes()with a *\**GatewayAddressvalue of an invalid unicast IP address. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.4.6 | 0xab7d87d5, 0x9761, 0x4877, 0x9f, 0x96, 0x42, 0xab, 0x99, 0x66, 0xd5, 0x3f | EFI\_UDP4\_PROTOCOL.Routes() **–** invokes Routes() to delete a route which is not in the routing table. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Routes()to delete a route which is not in the routing table. The return status should be **EFI\_NOT\_FOUND**.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.4.7 | 0x72569926, 0x4edb, 0x4d5b, 0xa2, 0xe5, 0x76, 0x31, 0x2f, 0xd2, 0x76, 0x74 | EFI\_UDP4\_PROTOCOL.Routes() **–** invokes Routes() to add a route that has already defined in the routing table. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Routes()to add a route into the routing table. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_PROTOCOL**.**Routes()to add the route into the routing table which has already defined in step 3. The return status should be **EFI\_ACCESS\_DENIED**.  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.4.10 | 0xe9ff9948, 0x9168, 0x4698, 0xa1, 0x49, 0x44, 0xef, 0x57, 0x33, 0x77, 0x20 | EFI\_UDP4\_PROTOCOL.Routes() **–** invokes Routes()when the EFI UDPv4 Protocol instance has not been started. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  3. Call EFI\_UDP4\_PROTOCOL**.**Routes()when the EFI UDPv4 Protocol instance has not been started. The return status should be **EFI\_NOT\_STARTED**.  3. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.4.8 | 0xae5c33be, 0x930e, 0x401b, 0x8f, 0x4d, 0x32, 0xc8, 0x95, 0xc4, 0x55, 0x48 | EFI\_UDP4\_PROTOCOL.Routes() **–** invokes Routes() to add a route to destination IP and send a packet to it. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Routes()to add a route into the routing table. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_PROTOCOL**.**Transmit()to send a packet to the destination IP and check that it is successful.  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.4.9 | 0xd39800b9, 0xe6e6, 0x4e29, 0xab, 0xd6, 0x17, 0x7a, 0x46, 0x10, 0x51, 0x3d | EFI\_UDP4\_PROTOCOL.Routes() **–** invokes Routes() to delete a route to destination IP and check that packet can not been sent to it. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Routes()to add a route into the routing table. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_PROTOCOL**.**Routes()to delete the route added in the step 3. The return status should be EFI\_SUCCESS.  5. Call EFI\_UDP4\_PROTOCOL**.**Transmit()to send a packet to the destination IP and check that it will be failed.  6. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |

### Transmit()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.1.5.1 | 0xd793cd46, 0x574d, 0x4f5d, 0x92, 0x8a, 0x2b, 0x84, 0x7a, 0xc0, 0x77, 0xd9 | EFI\_UDP4\_PROTOCOL.Transmit() – invokes Transmit() with a Token value of NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()with a Token value of NULL. The return status should be EFI\_INVALID\_PARAMETER**.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.2 | 0xf8ffef65, 0x20fe, 0x4381, 0xa5, 0x46, 0x07, 0x7c, 0x5a, 0x89, 0x7b, 0x6d | EFI\_UDP4\_PROTOCOL.Transmit() – invokes Transmit() with a Token.Event value of NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL.Transmit() with a Token.Event value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.3 | 0x157caa4e, 0xa260, 0x47a2, 0x97, 0x04, 0xd6, 0x62, 0x6c, 0xd9, 0x62, 0xf9 | EFI\_UDP4\_PROTOCOL.Transmit() – invokes Transmit() with a Token.Packet.FragmentCount value of 0. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL.Transmit() with a Token.Packet.FragmentCount value of 0. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.4 | 0xceebb331, 0x26c1, 0x4c6b, 0x91, 0x74, 0xb2, 0xdd, 0xda, 0xb7, 0x3a, 0x7a | EFI\_UDP4\_PROTOCOL.Transmit() – invokes Transmit() with a Token.Packet.TxData value of NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL.Transmit() with a Token.Packet.TxData value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.5 | 0xd381956d, 0x6b86, 0x48a4, 0x82, 0x56, 0x37, 0x5e, 0xa2, 0x46, 0xf6, 0xfa | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit() with a Token**.Packet.DataLength** value other than equal to the sum of fragment lengths. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()with a Token**.Packet.DataLength** value other than equal to the sum of fragment lengths. The return status should be EFI\_INVALID\_PARAMETER**.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.7 | 0x7f9fc4ec, 0x756c, 0x4399, 0xa2, 0x7e, 0x2e, 0x38, 0x3a, 0xff, 0x4e, 0x7b | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit()with the parameter Token.Packet.TxData*.*FragmentTable[].FragmentLenth/FragmetBufferfields being invalid. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()with the parameter Token.Packet.TxData*.*FragmentTable*[].FragmentLenth* fields being zero. The return status should be EFI\_INVALID\_PARAMETER**.**  4. Call EFI\_UDP4\_PROTOCOL**.**Transmit()with the parameter Token.Packet.TxData*.*FragmentTable*[].FragmentBuffer* fields being NULL. The return status should be EFI\_INVALID\_PARAMETER**.**  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.8 | 0x5d755449, 0x3840, 0x4cc8, 0x9c, 0x7f, 0x3a, 0x1a, 0xf3, 0x42, 0xd2, 0x89 | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit()with the parameter Token.Packet.TxData*.*GatewayAddress being not a valid unicast IPv4 address if it is not NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()with the parameter Token.Packet.TxData*.*GatewayAddress being 255.255.255.255. The return status should be EFI\_INVALID\_PARAMETER**.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.9 | 0x411080da, 0x2db4, 0x415e, 0xa0, 0xf5, 0x72, 0xf4, 0x1e, 0x55, 0x38, 0xdb | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit()with the parameter Token.Packet.TxData*.*GatewayAddress being not a valid unicast IPv4 address if it is not NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()with the parameter Token.Packet.TxData*.*GatewayAddress being 172.16.220.255. The return status should be EFI\_INVALID\_PARAMETER**.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.10 | 0x154ee561, 0x041a, 0x4e4b, 0x96, 0x3a, 0xfd, 0xc6, 0x4c, 0x4e, 0x3f, 0x29 | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit()with the parameter Token.Packet.TxData*.*GatewayAddress being not a valid unicast IPv4 address if it is not NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()with the parameter Token.Packet.TxData*.*GatewayAddress being 224.0.0.2. The return status should be EFI\_INVALID\_PARAMETER**.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.11 | 0x0161be6a, 0x75d4, 0x444b, 0xaf, 0x31, 0x78, 0xa4, 0xf0, 0x65, 0xed, 0x43 | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit()with Token.Packet.TxData*.*UdpSessionData being not valid unicast IPv4 addresses if it is not NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()with the Token.Packet.TxData*.*UdpSessionData being 224.0.0.1. The return status should be EFI\_INVALID\_PARAMETER**.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.12 | 0x3315e964, 0xc1bb, 0x4984, 0xb7, 0xc3, 0xff, 0x1a, 0x94, 0xb0, 0xe9, 0xd3 | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit()with Token.Packet.TxData*.*UdpSessionData being not valid unicast IPv4 addresses if it is not NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()with the Token.Packet.TxData*.UdpSessionData* being 172.16.220.0. The return status should be EFI\_INVALID\_PARAMETER**.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.13 | 0x4206d340, 0xe096, 0x4369, 0x96, 0x32, 0x9a, 0x35, 0x27, 0xcf, 0x64, 0xce | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit()when the EFI UDPv4 Protocol instance has not been started. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Transmit()when the EFI UDPv4 Protocol instance has not been started. The return status should be **EFI\_NOT\_STARTED.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.14 | 0xbd543b46, 0xcb6a, 0x4cfb, 0x80, 0x68, 0xe1, 0xaa, 0x28, 0x32, 0x43, 0x75 | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit()when there is no route to the destination network or address. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()when there is no route to the destination network or address. The return status should be **EFI\_NOT\_FOUND.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.15 | 0x0b3c198b, 0xfffd, 0x4dde, 0x9b, 0x1e, 0xbd, 0x5f, 0x8e, 0x70, 0xa0, 0xc2 | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit()when the data length is greater than the maximum UDP packet size. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()when the data length is greater than the maximum UDP packet size. The return status should be **EFI\_BAD\_BUFFER\_SIZE.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.16 | 0xae0d4495, 0xbcda, 0x4de3, 0xa4, 0xbc, 0xab, 0xed, 0xd4, 0x82, 0xdc, 0x92 | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit()when the length of the IP header+UDP header+data length is greater than MTU if *DoNotFragment* is **TRUE** | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit()when the length of the IP header+UDP header+data length is greater than MTU if *DoNotFragment* is **TRUE.** The return status should be **EFI\_BAD\_BUFFER\_SIZE.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.17 | 0xd983be7a, 0x33fd, 0x4308, 0x80, 0x6c, 0x00, 0x58, 0xef, 0xff, 0xe8, 0x17 | EFI\_UDP4\_PROTOCOL.Transmit() – to add a route to destination IP and send a packet to it. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL.Routes()to add a route.  4. Call EFI\_UDP4\_PROTOCOL.Transmit()to transmit packet.  5. Captured packet and verify.  6. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.18 | 0x71158c72, 0xa476, 0x42a8, 0x94, 0x81, 0x6d, 0xa0, 0xb8, 0xb4, 0x2c, 0xef | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit()when the *TxData.*GatewayAddress has been set | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL.Transmit()to transmit packet.  4. Captured packet and verify.  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.5.6 | 0xc0c68374, 0x0d85, 0x4bbb, 0x8b, 0x20, 0xbd, 0x88, 0xb1, 0xb0, 0x7b, 0xd7 | EFI\_UDP4\_PROTOCOL.Transmit() **–** invokes Transmit() with the transmit completion token whose Token.Event was already in the transmit queue. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Transmit() to transmit a packet. The return status should be EFI\_SUCCESS.  4. Call EFI\_UDP4\_PROTOCOL**.**Transmit()with the same Token.Event in step 3. The return status should be **EFI\_ACCESS\_DENIED.**  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |

### Receive()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.1.6.1 | 0x95bf8134, 0x5277, 0x413c, 0xbe, 0x1f, 0xf5, 0x03, 0x2b, 0x08, 0x78, 0x92 | EFI\_UDP4\_PROTOCOL.Receive() – invokes Receive() with a Token value of NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Receive()with a Token value of NULL. The return status should be EFI\_INVALID\_PARAMETER**.**  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.6.2 | 0xa158c0cd, 0x496b, 0x4dfe, 0x9c, 0xe9, 0x93, 0xea, 0x76, 0x40, 0x77, 0x7a | EFI\_UDP4\_PROTOCOL.Receive() – invokes Receive() with a Token.Event value of NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL.Receive() with a Token.Event value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.6.3 | 0xdd8e13d5, 0x7a76, 0x4237, 0x82, 0x14, 0x79, 0x03, 0xda, 0x61, 0x92, 0x4d | EFI\_UDP4\_PROTOCOL.Receive() **–** invokes Receive()when the EFI UDPv4 Protocol instance has not been started. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Receive()when the EFI UDPv4 Protocol instance has not been started. The return status should be **EFI\_NOT\_STARTED.**  3. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.6.4 | 0xe2a9f6b9, 0x7827, 0x474f, 0x97, 0x12, 0xc6, 0x9c, 0xad, 0xb0, 0x1c, 0x49 | EFI\_UDP4\_PROTOCOL.Receive() **–** invokes Receive()when a receive completion token with the same Token.Event was already in the receive queue. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL**.**Receive() at the first time the return status should be EFI\_SUCCESS**.**  3. Call EFI\_UDP4\_PROTOCOL.Receive() again the return status should be EFI\_ACCESS\_DENIED.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.6.5 | 0xa96aa0f5, 0x1c6b, 0x41cf, 0x98, 0x2f, 0xf8, 0x4f, 0x90, 0x43, 0x34, 0xb3 | EFI\_UDP4\_PROTOCOL.Receive() **–** the receiving fails because an ICMP error packet is received. | 1. Create a NETWORK unreachable packet.  2. Call EFI\_UDP4\_PROTOCOL**.**Receive()to receive the packet. The return status should be EFI\_SUCCESS**.**  3. Verify the R\_Token.Status it should be EFI\_NETWORK\_UNREACHABLE. |
| 5.26.1.6.6 | 0x3db8e8ee, 0x6c0b, 0x43d2, 0xa5, 0xfe, 0xb2, 0x34, 0x30, 0x5c, 0x12, 0xf8 | EFI\_UDP4\_PROTOCOL.Receive() **–** the receiving fails because an ICMP error packet is received. | 1. Create a HOST unreachable packet.  2. Call EFI\_UDP4\_PROTOCOL**.**Receive()to receive the packet. The return status should be EFI\_SUCCESS**.**  3. Verify the R\_Token.Status it should be EFI\_HOST\_UNREACHABLE. |
| 5.26.1.6.7 | 0x26f533d1, 0xb63e, 0x4997, 0xbd, 0x2d, 0x68, 0x52, 0xc8, 0x0c, 0xe3, 0x71 | EFI\_UDP4\_PROTOCOL.Receive() **–** the receiving fails because an ICMP error packet is received. | 1. Create a PROTOCOL error packet.  2. Call EFI\_UDP4\_PROTOCOL**.**Receive()to receive the packet. The return status should be EFI\_SUCCESS**.**  3. Verify the R\_Token.Status it should be EFI\_PROTOCOL\_UNREACHABLE. |
| 5.26.1.6.8 | 0xc982e2f7, 0xdf6f, 0x4a7b, 0x9d, 0x4a, 0x25, 0x87, 0x0c, 0x80, 0xb7, 0x9b | EFI\_UDP4\_PROTOCOL.Receive() **–** the receiving fails because an ICMP error packet is received. | 1. Create a PORT unreachable packet.  2. Call EFI\_UDP4\_PROTOCOL**.**Receive()to receive the packet. The return status should be EFI\_SUCCESS**.**  3. Verify the R\_Token.Status it should be EFI\_PORT\_UNREACHABLE. |
| 5.26.1.6.9 | 0x0685647b, 0xeee8, 0x4756, 0xbf, 0xea, 0x72, 0xc6, 0xb5, 0xff, 0x98, 0xb6 | EFI\_UDP4\_PROTOCOL.Receive() **–** the receiving fails because an ICMP error packet is received. | 1. Create a TCMP error packet.  2. Call EFI\_UDP4\_PROTOCOL**.**Receive()to receive the packet. The return status should be EFI\_SUCCESS**.**  3. Verify the R\_Token.Status it should be EFI\_ICMP\_ERROR. |

### Cancel()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.1.7.1 | 0xb4ca8ee0, 0x2b8b, 0x41b3, 0x97, 0x3c, 0x2f, 0x2b, 0x05, 0x07, 0x48, 0x17 | EFI\_UDP4\_PROTOCOL.Cancel() **–** invokes Cancel()to cancel a receive request while it has been completed. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Receive()to receive a packet and check that it is successful.  4. Call EFI\_UDP4\_PROTOCOL**.**Cancel() to cancel the receive request while the token has been completed. The return status should be **EFI\_NOT\_FOUND**.  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.7.2 | 0x46a1ec38, 0x0183, 0x485a, 0xa2, 0xa5, 0x50, 0x4e, 0x3b, 0xdb, 0x1b, 0x53 | EFI\_UDP4\_PROTOCOL.Cancel() **–** invokes Cancel()to cancel a receive request. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Receive()to receive a packet.  4. Call EFI\_UDP4\_PROTOCOL**.**Cancel() to cancel the receive request in step 3. Then check the packet sent to EUT will not be captured.  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |
| 5.26.1.7.3 | 0x6fff20b8, 0x55cd, 0x4610, 0xb3, 0xbe, 0xaa, 0x19, 0x5f, 0x29, 0x10, 0x66 | EFI\_UDP4\_PROTOCOL.Cancel() **–** invokes Cancel()to cancel all pending tokens with the parameter Token set to NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance.  3. Call EFI\_UDP4\_PROTOCOL**.**Receive()to set two requests in the receive queue.  4. Call EFI\_UDP4\_PROTOCOL**.**Cancel() with the parameter Token set to NULL. Then check that no packet sent to EUT will be captured.  5. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |

### Poll()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.1.8.1 | 0x18e54eae, 0x4d67, 0x468c, 0xb6, 0x0d, 0x81, 0x83, 0xd4, 0x07, 0xfe, 0xe8 | EFI\_UDP4\_PROTOCOL.Poll() **–** invokes **Poll()** when the instance has not been started. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_PROTOCOL.Configure() to configure the new EFI\_UDP4\_PROTOCOL instance. Then call EFI\_IP4\_PROTOCOL.Configure() again with an IpConfigData value of NULL.  3. Call EFI\_IP4\_PROTOCOL.**Poll()** for incoming data packets and processing outgoing data packets. The return status should be **EFI\_NOT\_STARTED**.  4. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |

### CreateChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.1.9.1 | 0xf88eaa0c, 0x764e, 0x45e0, 0x95, 0x86, 0xa6, 0x7f, 0x7d, 0x6f, 0xb2, 0x82 | EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() – invokes CreateChild() with a ChildHandle value of NULL. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child with a ChildHandle value of NULL. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.1.9.2 | 0x4dedef14, 0xbcba, 0x4b26, 0xbc, 0xc8, 0xb4, 0x7f, 0x8c, 0x08, 0xc9, 0x9d | EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() – invokes CreateChild() to create three instances. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create three EFI\_UDP4\_PROTOCOL instances and configure them. The return status should be EFI\_SUCCESS.  2. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. |

### DestroyChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.1.10.1 | 0x0ff5f5a1, 0x4d29, 0x40ae, 0xa4, 0xef, 0x02, 0x3b, 0xd3, 0xb8, 0x2e, 0x8c | EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() – invokes DestroyChild() with an invalid ChildHandle value. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()with an invalid ChildHandle value. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.1.10.3 | 0x9d888685, 0xfde7, 0x4832, 0xbc, 0x95, 0x03, 0xd6, 0x44, 0xc6, 0x29, 0xc5 | EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() – invokes DestroyChild() to destroy an existed child twice. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_UDP4\_PROTOCOL child.  2. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created EFI\_UDP4\_PROTOCOL. The return status should be EFI\_SUCCESS.  2. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() again. The return status should be EFI\_UNSUPPORTED. |
| 5.26.1.10.2 | 0x1ff85dcf, 0x885e, 0x42bf, 0x80, 0xd8, 0xf8, 0x4a, 0xaf, 0x11, 0xeb, 0x77 | EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() – invokes DestroyChild() to destroy a child. | 1. Call EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a EFI\_UDP4\_PROTOCOL child. The return status should be EFI\_SUCCESS.  2. Call **EFI\_UDP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created EFI\_UDP4\_PROTOCOL child and clean up the environment. The return status should be EFI\_SUCCESS. |

## EFI\_MTFTP4\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_MTFTP4\_PROTOCOL Section.

### CreateChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.2.1.1 | 0xf44c5295, 0x599e, 0x48bc, 0xbb, 0x67, 0xed, 0x9a, 0x21, 0x5b, 0xa9, 0xb1 | EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - returns EFI\_INVALID\_PARAMETER when creating Child 1 again. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create the same child again.  The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.1.5 | 0x5e30aa7c,0xd5f6,  0x4cac,  0xb2,0x54,  0xbf,0xdf,  0x16,0x3b,  0x34,0xfc | EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() – invokes CreateChild() with ChildHandle being NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild()with ChildHandle being NULL. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.26.2.1.2 | 0xca3fb64a, 0xd149, 0x4f76, 0x91, 0x45, 0xe4, 0xf6, 0xcc, 0xe6, 0x5b, 0x27 | EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - returns EFI\_SUCCESS when creating child1. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle: Handle1.  The return status must be FI\_SUCCESS. |
| 5.26.2.1.3 | 0xb07ae013, 0x0d83, 0x49c3, 0x99, 0x23, 0xef, 0x27, 0x67, 0xd5, 0x48, 0xfe | EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - returns EFI\_SUCCESS when creating child2. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle: Handle1.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle: Handle2. The return status must be EFI\_SUCCESS. |
| 5.26.2.1.4 | 0xd4d966c4, 0xc05a, 0x4995, 0xbf, 0xfb, 0x2c, 0x86, 0x8b, 0x3c, 0x2c, 0x0b | EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - returns EFI\_SUCCESS when creating child3. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle: Handle1.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle: Handle2.  4. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle: Handle3. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() three times to destroy the three newly created EFI\_MTFTP4\_PROTOCOL child handles and clean up the environment. |

### DestroyChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.2.2.1 | 0x3c312328, 0x313d, 0x47f6, 0x80, 0x7c, 0x5b, 0x1e, 0x10, 0xc2, 0xc0, 0x4d | EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - returns EFI\_INVALID\_PARAMETER when destroying a NULL child. | Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy a NULL child.  The return status must be EFI\_INVALID\_PARAMETER. |
| 5.26.2.2.2 | 0xe1c0ee52, 0xd5af, 0x4ec0, 0xa3, 0xf6, 0x31, 0xfb, 0xe0, 0xd4, 0xb7, 0x04 | EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - returns EFI\_INVALID\_PARAMETER when destroying an un-existed child. | Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy an un-existed child.  The return status must be EFI\_INVALID\_PARAMETER. |
| 5.26.2.2.3 | 0x28f8e30c, 0xa5d9, 0x4327, 0x99, 0xfa, 0xac, 0xda, 0xc9, 0x5f, 0xa4, 0xff | EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - returns EFI\_UNSUPPORTED when destroying the same child twice. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle:  2. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the new created child.  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created child again. The return status must be EFI\_UNSUPPORTED. |
| 5.26.2.2.4 | 0xcb939b7a, 0x266a, 0x44f5, 0xa2, 0xe3, 0x57, 0xea, 0xde, 0x7f, 0x44, 0x08 | EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - returns EFI\_SUCCESS with all valid invocations. | 1 .Add an entry in ARP cache.  2. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle: Handle1.  3. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters  4. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  5. If having not captured the packet, OS side set assert fail and call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created child and clean up the environment. The return status must be EFI\_SUCCESS. |
| 5.26.2.2.5 | 0xc9d38d67, 0xadc1, 0x425d, 0xa4, 0xa1, 0x04, 0x18, 0xc6, 0x4b, 0x63, 0x0c | EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - returns EFI\_SUCCESS with all valid invocations. | 1 .Add an entry in ARP cache.  2. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle: Handle1.  3. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters  4. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  5. If having captured the packet, configured OS side will send back a normal OACK packet with active flag set. 6. OS side captures ack packet sent from EUT side and call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the created child and clean up the environment. The return status must be EFI\_SUCCESS. |

### GetModeData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.2.3.1 | 0xdc9ac841, 0x8a0f, 0x4214, 0x91, 0x73, 0x60, 0x65, 0xee, 0x51, 0x8c, 0x52 | EFI\_MTFTP4\_PROTOCOL.GetModeData() - returns EFI\_INVALID\_PARAMETER with a *ModeData* value of NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.GetModeData() with a *ModeData* value of NULL. The return status must be EFI\_INVALID\_PARAMETER.  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.3.2 | 0x40eccfcd, 0xee1c, 0x405f, 0xb0, 0x64, 0x2d, 0xe5, 0x66, 0x7b, 0xfb, 0xee | EFI\_MTFTP4\_PROTOCOL.GetModeData() - returns EFI\_SUCCESS with all valid invocations. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.GetModeData() with all valid parameters. The return status must be EFI\_SUCCESS.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment.. |

### Configure()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.2.4.1 | 0x2c5b72d9, 0x2c30, 0x4249, 0xa2, 0x3a, 0x92, 0x14, 0xfd, 0xea, 0x73, 0x12 | EFI\_MTFTP4\_PROTOCOL.Configure() - returns EFI\_INVALID\_PARAMETER when  MtftpConfigData->*UseDefaultSetting* is **FALSE** and *MtftpConfigData*->*StationIp* is an invalid IPv4 unicast address. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2.Call EFI\_MTFTP4\_PROTOCOL.Configure() with  a MtftpConfigData-> UseDefaultSetting value of **FALSE** and a MtftpConfigData-> StationIp value of unicast address. The return status must be EFI\_INVALID\_PARAMETER**.**  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.4.2 | 0x01ef2cac, 0x1259, 0x41c9, 0xbd, 0x91, 0x49, 0x68, 0xa9, 0xfd, 0xd6, 0x42 | EFI\_MTFTP4\_PROTOCOL.Configure() - returns EFI\_INVALID\_PARAMETER when  MtftpConfigData-> UseDefaultSetting is FALSE and MtftpConfigData-> SubnetMask is invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2.Call EFI\_MTFTP4\_PROTOCOL.Configure() when *MtftpConfigData*->*UseDefaultSetting* is **FALSE** and *MtftpConfigData*->SubnetMask is invalid. The return status must be EFI\_INVALID\_PARAMETER**.**  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.4.3 | 0xbe92bd2e, 0xd085, 0x4da2, 0xaf, 0xbf, 0xec, 0x7b, 0x0d, 0xc7, 0xec, 0xca | EFI\_MTFTP4\_PROTOCOL.Configure() - returns EFI\_INVALID\_PARAMETER when  MtftpConfigData-> UseDefaultSetting is FALSE and MtftpConfigData-> ServerIp is an invalid IPv4 unicast address. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with a  *MtftpConfigData*->*UseDefaultSetting* value of **FALSE** and a *MtftpConfigData*->*ServerIp* value of an invalid IPv4 unicast address. The return status must be EFI\_INVALID\_PARAMETER**.**  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.4.4 | 0x5891d15c, 0x7f5d, 0x4c0d, 0xb0, 0x90, 0x88, 0xcd, 0x44, 0xe1, 0xea, 0x68 | EFI\_MTFTP4\_PROTOCOL.Configure() - returns EFI\_INVALID\_PARAMETER when  MtftpConfigData-> UseDefaultSetting is FALSE and MtftpConfigData-> GatewayIp is an invalid IPv4 unicast address. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with a  *MtftpConfigData*->*UseDefaultSetting* value of **FALSE** and a *MtftpConfigData*->*GatewayIp* value of an invalid IPv4 unicast address. The return status must be EFI\_INVALID\_PARAMETER**.**  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.4.5 | 0xd01d26be, 0x35fb, 0x4a08, 0xb0, 0x22, 0x7b, 0xe2, 0x53, 0xcf, 0x99, 0x02 | EFI\_MTFTP4\_PROTOCOL.Configure() - returns EFI\_INVALID\_PARAMETER when  *MtftpConfigData*->*UseDefaultSetting* is **FALSE** and *MtftpConfigData*->*GatewayIp* is not in the same subnet with station address. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2.Call EFI\_MTFTP4\_PROTOCOL.Configure() when  *MtftpConfigData*->*UseDefaultSetting* is **FALSE** and *MtftpConfigData*->*GatewayIp* is not in the same subnet with station address. The return status must be EFI\_INVALID\_PARAMETER**.**  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.4.6 | 0x37ccae28, 0x4b81, 0x4ba5, 0x8d, 0xe6, 0x79, 0xe7, 0xda, 0xb9, 0x03, 0x04 | **EFI\_MTFTP4\_PROTOCOL.Configure ()** - returns **EFI\_ACCESS\_DENIED** when some operation of this EFI MTFTPv4 Protocol driver instance has not finished yet and the configuration data cannot be changed at this time. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2.Call EFI\_MTFTP4\_PROTOCOL.Configure() when  some operation of this EFI MTFTPv4 Protocol driver instance has not finished yet and the configuration data cannot be changed at this time. The return status must be **EFI\_ACCESS\_DENIED.**  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.4.7 | 0xd31e47ea, 0x5a76, 0x49aa, 0xbd, 0x40, 0x6f, 0xd9, 0x49, 0x88, 0x5f, 0x84 | **EFI\_MTFTP4\_PROTOCOL.Configure ()** - returns EFI\_SUCCESSwhen it is reset by calling Configure () with a MtftpConfigData value of NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call **EFI\_MTFTP4\_PROTOCOL.**Configure()with MtftpConfigData set to NULL. The return status must be EFI\_SUCCESS**.**  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |

### GetInfo()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.2.5.1 | 0x794b1aae, 0x92b4, 0x40de, 0xad, 0xed, 0x43, 0xb3, 0x55, 0x37, 0xd8, 0xa3 | EFI\_MTFTP4\_PROTOCOL.GetInfo() - returns EFI\_INVALID\_PARAMETER with a FileName value of NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with a MtftpConfigData-> UseDefaultSetting value of FALSE.  3. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() with a *FileName* value of NULL. The return stats must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.2 | 0x0733cdb5, 0x4072, 0x4129, 0xa2, 0x06, 0xce, 0x56, 0x6e, 0xf6, 0xd8, 0x61 | EFI\_MTFTP4\_PROTOCOL.GetInfo() - returns EFI\_INVALID\_PARAMETER with an OverrideData.GatewayIp value of invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with a MtftpConfigData-> UseDefaultSetting value of FALSE.  3. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() with an OverrideData.GatewayIp value of invalid. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.3 | 0xa04d3e7c, 0x5e50, 0x4472, 0xa7, 0x70, 0xc1, 0xa9, 0x48, 0xcb, 0xd9, 0x1e | EFI\_MTFTP4\_PROTOCOL.GetInfo() - returns EFI\_INVALID\_PARAMETER with an invalid OverrideData.ServerIp value. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with a MtftpConfigData-> UseDefaultSetting value of FALSE.  2. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() with an OverrideData.ServerIp value of invalid. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.4 | 0x10d2101c, 0x0aa3, 0x4713, 0xb8, 0x2b, 0xe1, 0x43, 0xed, 0xf4, 0x11, 0x26 | EFI\_MTFTP4\_PROTOCOL.GetInfo() - returns EFI\_INVALID\_PARAMETER when OverrideData.GatewayIp and OverrideData.ServerIp are not in the same subnet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with a MtftpConfigData->UseDefaultSetting value of FALSE.  3. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() with OverrideData.GatewayIp and OverrideData.ServerIp are not in the same subnet. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.5 | 0xf85b07f6, 0x9f89, 0x41ad, 0x8d, 0x53, 0x47, 0x53, 0x97, 0xac, 0x98, 0x1a | EFI\_MTFTP4\_PROTOCOL.GetInfo() - returns EFI\_INVALID\_PARAMETER when OptionCount is not 0 and OptionList is NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with a MtftpConfigData->UseDefaultSetting value of FALSE.  3. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() when OptionCount is not 0 and OptionList is NULL. The return status must be EFI\_INVALID\_PARAMETER**.**  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.6 | 0xb9caedcf, 0xf071, 0x421a, 0x9f, 0xb9, 0x7e, 0x24, 0x9d, 0xf4, 0xe3, 0xb2 | EFI\_MTFTP4\_PROTOCOL.GetInfo() - returns EFI\_INVALID\_PARAMETER when PacketLength is NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with a MtftpConfigData->UseDefaultSetting value of FALSE.  3. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() with a PacketLength value of NULL. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.7 | 0x5cb9e305, 0xb4e2, 0x4416, 0xa7, 0x35, 0xe2, 0x72, 0xb6, 0x98, 0xf8, 0x23 | EFI\_MTFTP4\_PROTOCOL.GetInfo() - returns **EFI\_TFTP\_ERROR** with a MTFTPv4 ERROR packet having received in the Buffer. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with MtftpConfigData->UseDefaultSetting is FALSE.  3. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() when OverrideData is NULL and ModeStr is NULL. OS side must capture the packet sent from EUT side.  4. If have captured the packet, configured OS side to send back a MTFTPv4 ERROR packet and OS side should capture another packet sent from EUT side. The return status must be **EFI\_TFTP\_ERROR**.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.8 | 0x30e6a222, 0x2bbc, 0x4ff6, 0xa8, 0xf2, 0xd6, 0x8a, 0xc2, 0x91, 0x98, 0x29 | EFI\_MTFTP4\_PROTOCOL.GetInfo() - returns **EFI\_TIMEOUT** when no packets were received from the MTFTPv4 server. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() when OverrideData is NULL and ModeStr is NULL. In addition, the OS side doesn’t send any packets back. The return status must be **EFI\_TIMEOUT.**  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.9 | 0xc4c5ced1, 0x30a5, 0x4c54, 0xa3, 0xc0, 0x80, 0x2b, 0x35, 0x83, 0xbf, 0x70 | EFI\_MTFTP4\_PROTOCOL.GetInfo() - returns **EFI\_NOT\_STARTED** with the EFI MTFTPv4 Protocol driver having not been started. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.GetInfo()when both OverrideData and ModeStr are NULL. The return status must be EFI\_NOT\_STARTED.  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.10 | 0x32db978c, 0x9d9b, 0x4144, 0x97, 0x9c, 0x27, 0x14, 0x42, 0x9f, 0xe3, 0x47 | EFI\_MTFTP4\_PROTOCOL.GetInfo() - returns EFI\_ACCESS\_DENIED when invoking GetInfo() interface while the previous operation has not been completed yet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() to change the EFI\_MTFTP4\_PROTOCOL State.  4. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() when the previous operation has not been completed yet. The return status must be **EFI\_ACCESS\_DENIED**.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.11 | 0xbf72714a, 0x113f, 0x487e, 0xab, 0x10, 0x08, 0xa7, 0x98, 0xf3, 0x4f, 0xc4 | EFI\_MTFTP4\_PROTOCOL.GetInfo() - returns EFI\_SUCCESS when the server responding a normal OACK packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() with all valid parameters. OS side should capture the packet sent from EUT side.  4. Configure OS side to send back a normal OACK packet and OS side should capture another packet sent from EUT side.  5. The return status of the EFI\_MTFTP4\_PROTOCOL.GetInfo() must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.12 | 0x77dbe1e4, 0x6219, 0x4531, 0xae, 0xbe, 0x58, 0x26, 0x4b, 0x53, 0x7e, 0xd1 | EFI\_MTFTP4\_PROTOCOL.GetInfo() - test the **EFI\_ICMP\_ERROR** conformance of GetInfo() when an ICMP ERROR packet was received and in the buffer. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() with all valid parameters. OS side should capture the packet sent from EUT side.  4. Configure OS side to send back a ICMP error packet. The return status must be **EFI\_ICMP\_ERROR.**  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.13 | 0x70e8d3e9, 0x75a9, 0x4652, 0x82, 0x68, 0xa4, 0x0d, 0xdd, 0x1a, 0x81, 0x5f | EFI\_MTFTP4\_PROTOCOL.GetInfo() - test the **EFI\_UNSUPPORTED** conformance of GetInfo() when one or more options in the optionlist are in the unsupported list of structure EFI\_MTFTP4\_MODE\_DATA. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() with one or more options in the optionlist are in the unsupported list of structure EFI\_MTFTP4\_MODE\_DATA. The return status should be **EFI\_UNSUPPORTED.**  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.5.14 | 0xfaa23d30, 0x1d66, 0x4d8e, 0xbe, 0x21, 0x2d, 0xa7, 0xbc, 0x1c, 0x9d, 0xfd | EFI\_MTFTP4\_PROTOCOL.GetInfo() - test the **EFI\_PROTOCOL\_ERROR** conformance of GetInfo(). The client received an unexpected MTFTPv4 packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.GetInfo() with all valid parameters. OS side should capture the packet sent from EUT side.  4. Configure OS side to send back an unexpected packet and the return status should be **EFI\_PROTOCOL\_ERROR**.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.5.15 | 0xd2c1e819, 0x610b, 0x4cfc, 0x94, 0xf1, 0x33, 0xcd, 0x13, 0xaf, 0x4b, 0xc9 | EFI\_MTFTP4\_PROTOCOL.GetInfo – GetInfo()must return EFI\_NETWORK\_UNREACHABLEwhen receive an ICMP net unreachable packet. | 1. Call  **EFI\_MTFTP4\_SERVICE\_BINDING\_PROT OCOL.CreateChild()** to create a new **EFI\_MTFTP4\_PROTOCOL** child  handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call  EFI\_MTFTP4\_PROTOCOL.GetInfo()with all valid parameters. OS side should capture the packet sent from  EUT side.  4. Configure Host side to send back an  ICMP net unreachable packet and the return status should be EFI\_NETWORK\_UNREACHABLE.  5. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_P ROTOCOL.DestroyChild()to  destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.5.16 | 0x290076e3, 0xdaf2, 0x453d, 0xb2,0x21,0xcd,0x27, 0xce,,0xe7,0x3d,0xbe | EFI\_MTFTP4\_PROTOCOL.GetInfo – GetInfo() must return EFI\_HOST\_UNREACHABLE when receiving an ICMP host unreachable packet. | 1. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_PROT OCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child  handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configur  e() with all valid parameters.  3. Call  EFI\_MTFTP4\_PROTOCOL.GetInfo() with all valid parameters. OS side  should capture the packet sent from  EUT side.  4. Configure Host side to send back an  ICMP host unreachable packet and the return status should be EFI\_HOST\_UNREACHABLE.  5. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_P ROTOCOL.DestroyChild() to  destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.5.17 | 0x706bc816, 0x6353, 0x40ae, 0xa9,0x47,0x9a,0xf0, 0x01,0xa9,0x82,0x8c | EFI\_MTFTP4\_PROTOCOL.GetInfo – GetInfo() must return EFI\_PROTOCOL\_UNREACHABLE when receive an ICMP protocol unreachable packet. | 1. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_PROT OCOL.CreateChild()to create a new EFI\_MTFTP4\_PROTOCOL child  handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configur  e() with all valid parameters.  3. Call  EFI\_MTFTP4\_PROTOCOL.GetInfo() with all valid parameters. OS side  should capture the packet sent from  EUT side.  4. Configure Host side to send back an  ICMP protocol unreachable packet and the return status should be EFI\_PROTOCOL\_UNREACHABLE.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()to  destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.5.18 | 0xa165bd19, 0x951a, 0x4486, 0x88,0x4d,0x1d,0x94,0x30,0xa7,0xbe,0x3c | EFI\_MTFTP4\_PROTOCOL.GetInfo – GetInfo() must return EFI\_PORT\_UNREACHABLE when receive an ICMP port unreachable packet. | 1. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_PROT OCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child  handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configur  e() with all valid parameters.  3. Call  EFI\_MTFTP4\_PROTOCOL.GetInfo() with all valid parameters. OS side  should capture the packet sent from  EUT side.  4. Configure Host side to send back an  ICMP port unreachable packet and the return status should be EFI\_PORT\_UNREACHABLE.  5. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_P ROTOCOL.DestroyChild() to  destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |

### ParseOptions()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.2.6.1 | 0x9bea2f3f, 0x9f02, 0x4eb2, 0x8b, 0x1f, 0x99, 0xd5, 0xcf, 0xc3, 0x57, 0x29 | EFI\_MTFTP4\_PROTOCOL.ParseOptions() - returns EFI\_INVALID\_parameter with a *PacketLength* value of 0. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ParseOptions() with a PacketLength value of 0. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.6.2 | 0x0bc09196, 0xb38a, 0x4fa8, 0xb0, 0x38, 0x4c, 0x4c, 0x8b, 0x3c, 0x69, 0xfa | EFI\_MTFTP4\_PROTOCOL.ParseOptions() - returns EFI\_INVALID\_parameter with a Packet value of NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ParseOptions() with a Packet value of NULL. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.6.3 | 0x72723929, 0x60bd, 0x49c1, 0x99, 0xbd, 0xd1, 0x48, 0x60, 0x33, 0x7a, 0xdc | EFI\_MTFTP4\_PROTOCOL.ParseOptions() - returns EFI\_INVALID\_parameter with a Packet value of an invalid MTFTPv4 Packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ParseOptions() with a Packet value of an invalid MTFTPv4 Packet **-** **Packet.OpCode** is 0x11. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.6.4 | 0xb7ed01b9, 0x7e1b, 0x40ba, 0x8b, 0x6a, 0x52, 0x34, 0xdf, 0x13, 0x53, 0xf0 | EFI\_MTFTP4\_PROTOCOL.ParseOptions() - returns EFI\_INVALID\_parameter with a Packet value of an invalid MTFTPv4 Packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ParseOptions() with a Packet value of an invalid MTFTPv4 Packet - **Packet.OpCode** is 0x01.  The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.6.5 | 0x350c473e, 0x9901, 0x4125, 0xbc, 0xc9, 0x65, 0xbf, 0xa9, 0xf3, 0x16, 0x30 | EFI\_MTFTP4\_PROTOCOL.ParseOptions() - returns EFI\_INVALID\_parameter with a Packet value of an invalid MTFTPv4 Packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ParseOptions() with a Packet value of an invalid MTFTPv4 Packet - **Packet.OpCode** is 0x06 and PacketLength is 1. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.6.6 | 0xcf909489, 0xace2, 0x4fec, 0x8d, 0xc9, 0x66, 0xa0, 0xd9, 0x33, 0xa6, 0x4a | EFI\_MTFTP4\_PROTOCOL.ParseOptions() - returns EFI\_INVALID\_parameter with an OptionCount value of NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ParseOptions() with an OptionCount value of NULL.  The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.6.7 | 0x0131da11, 0x62a1, 0x494f, 0xb1, 0x0a, 0xaf, 0x5d, 0xe2, 0x12, 0xe9, 0x88 | EFI\_MTFTP4\_PROTOCOL.ParseOptions() - returns EFI\_INVALID\_PARAMETER when parsing a non-OACK packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call BS-> CopyMem() to fill the packet needed to be parsed. Set **Packet.OpCode** to be 0x100.  4. Call EFI\_MTFTP4\_PROTOCOL.ParseOptions() with the configured non-OACK packet.  The return status must be EFI\_INVALID\_PARAMETER.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.6.9 | 0x5b7bbe95, 0xdba3, 0x4e9c, 0x89, 0xde, 0x37, 0xf1, 0xf6, 0x42, 0x04, 0x24 | EFI\_MTFTP4\_PROTOCOL.ParseOptions() - test the EFI\_NOT\_FOUND conformance of ParseOptions() with no options were found in the OACK packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  4. Call EFI\_MTFTP4\_PROTOCOL.ParseOptions() with no options were found in the OACK packet..The return status must be EFI\_NOT\_FOUND.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.6.8 | 0x973e370a, 0x5936, 0x4377, 0xb0, 0x6c, 0x82, 0xe6, 0x11, 0x4d, 0xda, 0x6f | EFI\_MTFTP4\_PROTOCOL.ParseOptions() - returns EFI\_SUCCESS when parsing a OACK packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call BS-> CopyMem() to fill the packet needed to be parsed. Set **Packet.OpCode** to be 0x600.  4. Call EFI\_MTFTP4\_PROTOCOL.ParseOptions() with the configured OACK packet. The return status must be EFI\_SUCCESS.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |

### ReadFile()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.2.7.1 | 0x38728e11, 0x6f6f, 0x409a, 0x84, 0x31, 0xf5, 0x1e, 0x60, 0x0f, 0x7d, 0x6f | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TIMEOUT with no packets sent back from the MTFTPv4 server. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.R**eadFi**le() with all valid parameters.  4. If OS side has captured the packet, don’t send back any packets, stall and wait until client timeout.  The return status must be **EFI\_TIMEOUT**.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.2 | 0xcb0105ab, 0x7f16, 0x46a1, 0x87, 0xf2, 0x18, 0x6b, 0x86, 0x74, 0x6a, 0xba | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TIMEOUT when the passive Client having not received any data packets from the server. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. Configure OS side to send back a normal OACK packet with flag set to be passive.  5. Then OS side doesn’t send any data packets back, then stall and wait until client timeout.  The return status must be EFI\_TIMEOUT.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.3 | 0x95384167, 0xa706, 0x4f2c, 0x82, 0x8c, 0x8e, 0x3f, 0x15, 0xee, 0x82, 0x0a | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when TFTPv4 ERROR packet was received. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If having captured the packet, configure OS side to send back a EFI\_MTFTP4\_PROTOCOL Error packet.  5. OS side should capture another packet sent from EUT side. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.4 | 0xf5ac75d7, 0xa32e, 0x4b1f, 0xa8, 0x19, 0x2e, 0xfc, 0x73, 0x24, 0xcc, 0xba | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when the active client receives an MTFTPv4 ERROR packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If having captured the packet, Configure OS side to respond a normal OACK with flag set to be active.  5. If having captured ack, OS side sends back a EFI\_MTFTP4\_PROTOCOL Error packet. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.5 | 0x49f424ed, 0xfdbc, 0x4c82, 0x8d, 0xb8, 0xd5, 0xa2, 0xa4, 0x9b, 0x7e, 0xff | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when the passive client has received a MTFTPv4 ERROR packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If having captured the packet, configure OS side to respond a normal OACK packet with flag set to be passive, and then send back a EFI\_MTFTP4\_PROTOCOL Error packet. The return status must be EFI\_TFTP\_ERROR.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.6 | 0x1392cef9, 0x74e0, 0x4f89, 0xa5, 0x26, 0xa7, 0xa7, 0x77, 0x56, 0x33, 0xd4 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when the server responds with an error OACK packet – active/passive flag error. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to respond with an error OACK packet – active/ passive flag error.  5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.7 | 0x1f4fd053, 0x9e4b, 0x49c4, 0x9a, 0xea, 0x58, 0x75, 0x60, 0xf1, 0xec, 0x7d | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when timeout value in OACK packet is invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to respond with an error OACK packet – timeout value is invalid.  5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.8 | 0x9bbcc0bb, 0x5386, 0x4e5c, 0xa3, 0xac, 0x65, 0xc7, 0x62, 0xf6, 0x93, 0xaa | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when blocksize option value in OACK packet is invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to respond with an error OACK packet – blocksize option value is invalid.  5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.9 | 0x329ae187, 0x6758, 0x42b9, 0x84, 0xae, 0x92, 0x32, 0x42, 0x15, 0xa5, 0xef | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when multicast IP address in OACK packet is invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to respond with an error OACK packet – multicast IP address is invalid.  5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.10 | 0xe491fc10, 0x0c0f, 0x4d45, 0xb5, 0xc3, 0x3c, 0x29, 0x10, 0xdb, 0xe4, 0x70 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when client's listening port in OACK packet is invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to respond with an error OACK packet – client's listening port is 65536.  5. If having captured an ack, OS check whether it is a packet with an error code. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.11 | 0xcff83e43, 0x5d33, 0x4cc0, 0x80, 0xc4, 0x55, 0x96, 0x0e, 0x5f, 0x58, 0xae | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when the format of multicast IP address in OACK packet is invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to respond with an error OACK packet – the format of multicast IP address is invalid.  5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.12 | 0x28754983, 0xac7d, 0x4e7f, 0x9f, 0xad, 0xbf, 0x55, 0x59, 0xff, 0xa7, 0x62 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when the format of multicast option in OACK packet is invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to respond with an error OACK packet – the format of multicast option is invalid.  5. If having captured the ack, OS check whether it is a packet with an error code. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.13 | 0x64fd965d, 0x2acc, 0x4540, 0xbc, 0x57, 0x50, 0xe8, 0xab, 0x02, 0xe8, 0x8a | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when the format of multicast option in OACK packet is invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to respond with error OACK packet – the format of multicast option is invalid.  5. If having captured ack, OS check whether it is a packet with error code. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.14 | 0xd09c7076, 0x316f, 0x4245, 0xac, 0x31, 0x95, 0x82, 0x22, 0xa4, 0x67, 0xd7 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when the format of multicast option in OACK packet is invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to respond with an error OACK packet – the format of multicast option is invalid.  5. If having captured an ack, OS check whether it is a packet with an error code. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.15 | 0x1322cb38, 0x8f90, 0x4fa8, 0xbe, 0xa9, 0x5b, 0x31, 0x8e, 0xb8, 0x24, 0xad | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when the passive client tries to change to be active, but the server responds with an error OACK packet - active/passive flag is error. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to respond OACK with flag set to be passive and then send the file missing several packets.  5. The OS side should capture the ack sent from the passive client to ask for the missing packets.  7. If having captured it, OS sends back OACK with error active/ passive flag. The return status must be EFI\_TFTP\_ERROR.  8. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.16 | 0xff2d0e80, 0xdecd, 0x4a1c, 0xb6, 0x7c, 0xe4, 0xcd, 0x99, 0x9d, 0x69, 0x09 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when the server adds more other options in the OACK packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active while adding more other options in the OACK packet; then OS should capture the ack packet.  5. If having captured ack, OS sends back the only data packet and then receives another ack. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.17 | 0xa7fcbfff, 0x8367, 0x466e, 0x9d, 0x25, 0x5b, 0x80, 0xb8, 0x4f, 0xb5, 0x8f | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR when active client receives OACK, while Token-> OptionCount is 0. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() without any option requested. OS side must capture the packet  sent from EUT side  4. If OS side has captured the packet, configure server to send back OACK with some options and flag set to be active.  5. Then OS should capture ack and sends back the only one data packet. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.18 | 0x00450815, 0x41f5, 0x4da8, 0x90, 0x66, 0x78, 0x80, 0x94, 0x07, 0x34, 0xea | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_TFTP\_ERROR –When the passive client downloads, it misses the first and the last data packet. Then server set the client to be active while changing the transfer channel. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive.  5. Then server sends back the second and the third data packet to the multicast IP address while missing the first and the last data packet.  6. After passive client is timeout, it’ll send ack0 to ask for the missing packets and the server should capture it.  7. If having captured the request, the server sends back OACK with flag set to be active and the client’s listening port also changed. Then the server should capture an error packet. The return status must be EFI\_TFTP\_ERROR.  8. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.19 | 0x9017244c, 0x127a, 0x486e, 0x81, 0x5b, 0x20, 0xe8, 0xa6, 0x55, 0xd4, 0x6f | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_NOT\_STARTED with the EFI MTFTPv4 Protocol driver having not been started. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. The return status must be EFI\_NOT\_STARTED.  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.20 | 0x84b13fab, 0x04f5, 0x474b, 0x89, 0x4c, 0x63, 0xef, 0x9d, 0xcf, 0x78, 0x58 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_INVALID\_PARAMETER when Token is NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() when Token is NULL. The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.21 | 0xd25ff5a4, 0x71e7, 0x4e38, 0xb4, 0x3e, 0x4a, 0xcc, 0xe7, 0x83, 0xfa, 0x77 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_INVALID\_PARAMETER when Token->Filename is NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() when Token->Filename is NULL. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.22 | 0xf370c329, 0xe20b, 0x45a0, 0x9a, 0xb3, 0xd4, 0x13, 0x70, 0x98, 0x00, 0x03 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_INVALID\_PARAMETER when Token-> OptionCount is not 0 and Token-> OptionList is NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() when Token->OptionCount is not 0 and Token->OptionList is NULL. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.23 | 0x2357c86f, 0xf9ba, 0x4f25, 0x9c, 0x77, 0x75, 0x10, 0xab, 0xb5, 0x10, 0x7e | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_INVALID\_PARAMETER when Token->Buffer and Token->CheckPacket are both NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() when Token->Buffer and Token->CheckPacket are both NULL. The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.24 | 0x66019567, 0x321d, 0x41a8, 0xaa, 0xff, 0x60, 0x7f, 0x75, 0xa4, 0x08, 0x42 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_INVALID\_PARAMETER when OverrideData.GatewayIp is invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() when OverrideData.GatewayIp is invalid.  The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.25 | 0x5f64495c, 0xad06, 0x4185, 0x87, 0x55, 0x86, 0xd9, 0x44, 0xf6, 0x39, 0x81 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_INVALID\_PARAMETER when OverrideData.ServerIp is invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() when OverrideData.ServerIp is invalid. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.26 | 0x17fa0734, 0x38f6, 0x4fe5, 0x9f, 0x6a, 0x5d, 0xae, 0x9e, 0xf2, 0xf3, 0xac | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_INVALID\_PARAMETER when OverrideData.GatewayIp is not in the same subnet with *StationIp*. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() when OverrideData**.**GatewayIp is not in the same subnet with *StationIp*. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.27 | 0xa5d93fc4, 0x9b20, 0x45cc, 0xbe, 0x45, 0xcc, 0x60, 0x5e, 0x51, 0xae, 0xf4 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_UNSUPPORTED when options of “restart” and “session” in the Token->OptionList are in the unsupported list of this implementation. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() when options of “restart” and “session” in the Token->OptionList are in the unsupported list of this implementation. The return status must be  EFI\_UNSUPPORTED.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.28 | 0x40f05e07, 0x3a7b, 0x4244, 0x97, 0x4f, 0x96, 0x9a, 0x89, 0x5c, 0xa4, 0x83 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_UNSUPPORTED when option of “pktdelay” in the Token->OptionList are in the unsupported list of this implementation. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() when option of “pktdelay” in the Token->OptionList are in the unsupported list of this implementation.The return status must be EFI\_UNSUPPORTED.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.29 | 0xa8d5abdf, 0x3e19, 0x462e, 0x9f, 0x6d, 0x9f, 0xa6, 0x13, 0xd2, 0x96, 0xd3 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_ACCESS\_DENIED for calling EFI\_MTFTP4\_PROTOCOL.ReadFile() again before the first call ends. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() for the first time with all valid parameters.  4. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() for the second time with the same Token before the first call ends. The return status must be EFI\_ACCESS\_DENIED.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.30 | 0xab02a8d2, 0x2086, 0x4372, 0xb5, 0xc7, 0x06, 0x0e, 0x28, 0x65, 0x1e, 0x8f | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_BUFFER\_TOO\_SMALL whenclient is active and the BufferSize is not larger enough to hold the downloaded data in downloading process. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. Configure OS side to send back a normal OACK packet with multicast option and flag set to be active. In addition, OS side should capture Ack packet sent from EUT side and then responds with data packet whose size is larger than the set BufferSize.  5. Then OS side should capture another packet.  The return status must be  EFI\_BUFFER\_TOO\_SMALL.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.31 | 0xf135f02b, 0x51ca, 0x47b9, 0xab, 0xf4, 0x4b, 0xd9, 0x78, 0x86, 0x68, 0xf8 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_BUFFER\_TOO\_SMALL when client is passive. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. Configure OS side to send back a normal OACK packet with multicasts option and flag set to be passive and wait for the client's processing.  5. OS side sends a data packet whose size is larger than the set BufferSize. The return status must be EFI\_BUFFER\_TOO\_SMALL.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.32 | 0xb8363dd2, 0xedca, 0x49a6, 0xbe, 0x32, 0x90, 0x87, 0xb9, 0x57, 0x6a, 0x1f | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_BUFFER\_TOO\_SMALL when calling ReadFile asynchronously and Client is passive. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() asynchronously with all valid parameters. OS side must capture the packet sent from EUT side.  4. Configure OS side to send back a normal OACK packet with option of multicast and flag set to be passive and wait for the client's processing.  5. OS side sends a data packet whose size is larger than the set BufferSize. The return status must be EFI\_BUFFER\_TOO\_SMALL.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment |
| 5.26.2.7.33 | 0x5ae24123, 0xbb88, 0x42a5, 0xa1, 0xd0, 0xb3, 0x49, 0xfa, 0x20, 0x04, 0x6f | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_BUFFER\_TOO\_SMALL when the client is an active client and the BufferSize is not larger enough to hold the downloaded data in downloading process - return this status until having received the last data block. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. Configure OS side to send back a normal OACK packet with multicast option and flag set to be active. In addition, OS side should capture Ack packet sent from EUT side and then responds with serious data packets whose size are larger than the set BufferSize.  5. Then OS side should capture the ack for the data blocks except the last block.  6. Then OS side should capture the error packet.  The return status must be  EFI\_BUFFER\_TOO\_SMALL.  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.34 | 0xcfdaf47b, 0x8a46, 0x498c, 0x92, 0x0e, 0x96, 0x15, 0xc1, 0x23, 0xbe, 0x57 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_ABORTED when the user aborts the active download process in CheckPacket callback routine in the case of receiving data packets. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with CheckPacket callback set. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture an ack packet sent from client.  5. If having captured it, server sends the only data packet back to the client.  6. Then server should capture another packet and check that if it is an error packet. The return status must be EFI\_ABORTED.  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.35 | 0x731fb0ec, 0xb6b1, 0x4424, 0xb0, 0x61, 0x1b, 0xaa, 0xb3, 0x3f, 0xc0, 0x88 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_ABORTED when the user aborts the active download process in CheckPacket callback routine in the case of receiving OACK packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with CheckPacket callback set. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture another packet and check whether it is an error packet. The return status must be EFI\_ABORTED.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.36 | 0xbd75e9f5, 0x76b3, 0x4e67, 0xb9, 0xbf, 0xcd, 0xfb, 0xed, 0x5c, 0x34, 0xa6 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_ABORTED when the user aborts the passive download process in CheckPacket callback routine in the case of receiving data packets. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with CheckPacket callback set. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive.  5. If having captured it, server sends the only data packet back to the client.  6. Then server should capture another packet and check that if it is an error packet. The return status must be EFI\_ABORTED.  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.37 | 0xc9f2cdc8, 0x38eb, 0x4446, 0x9d, 0xc4, 0x5c, 0x78, 0x4a, 0x69, 0x0b, 0xd1 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_ABORTED when the user aborts the passive download process in CheckPacket callback routine in the case of receiving OACK packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with CheckPacket callback set. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive and then OS should capture another packet and check that if it is an error packet.  The return status must be EFI\_ABORTED.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.38 | 0xc911f1f0, 0x385b, 0x4de3, 0xb3, 0x86, 0xe3, 0x20, 0xec, 0x3c, 0xa8, 0xc2 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_ABORTED when the user aborts the active download process in TimeoutCallback routine in the case of receiving Ack. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with Timeout callback set. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack. The return status must be EFI\_ABORTED.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.39 | 0xfd55be46, 0xb941, 0x4708, 0xbe, 0x69, 0x24, 0x82, 0xca, 0x2c, 0x29, 0x34 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_ABORTED when the user aborts the passive download process in Timeout Callback routine in the case of receiving Ack. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with Timeout callback set. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive and then OS should capture ack. The return status must be EFI\_ABORTED.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.40 | 0x8ad083d8, 0x9757, 0x40ef, 0x99, 0x86, 0x21, 0xee, 0x90, 0x4a, 0xa0, 0x2d | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the server sends back with normal OACK packet whose active flag is set. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet.  5. If having captured ack, OS sends back the only data packet and then receives another ack. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.41 | 0xc3640c29, 0xbfcd, 0x4f0c, 0xae, 0x7e, 0xcc, 0x44, 0x8a, 0xc1, 0x8e, 0x16 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when server send backs with normal OACK packet whose passive flag is set. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive and stall to wait for the client to join in the multicast group.  5. OS sends back the only data packet. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.42 | 0x5e294d5a, 0xf09e, 0x4fdc, 0xa2, 0x2e, 0x9d, 0xcb, 0xfa, 0x44, 0x3d, 0x2b | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the server sends back normal OACK packet after the client resends RRQ several times. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters, client should retry 5 times to send RRQ then OS side should capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet.  5. If having captured ack, OS sends back the only data packet and then receives another ack. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.43 | 0x162e4457, 0x63d9, 0x4402, 0xad, 0xac, 0xaa, 0xdf, 0x3a, 0x61, 0xaf, 0xdc | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the server doesn’t copy the client’s option strings verbatim from the RRQ packet to the OACK packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active while not coping the client’s option strings verbatim from the RRQ packet to the OACK packet; then OS should capture ack packet.  5. If having captured ack, OS sends back the only data packet and then receives another ack. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.44 | 0xe0d3922c, 0x017d, 0x44a2, 0x90, 0x88, 0xad, 0xb6, 0xeb, 0x9f, 0x4c, 0xed | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when client receives an error server source port data packet, it just ignores the packet and continues the data processing. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active; then OS should capture ack packet.  5. If having captured ack, OS sends back an error server source data packet. In addition, client just ignores it.  6. The server sends back another correct source data packet then. The return status must be EFI\_SUCCESS.  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.45 | 0xcc4f141c, 0x9df1, 0x404e, 0x90, 0x27, 0x60, 0xea, 0xbd, 0xa8, 0x08, 0xd8 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS After passive client having received some packets, the server sets it to be active and sends out remaining packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back a normal OACK to set the client passive.  5. If having captured ack0, OS sends back the first and the last data packet.  6. Server resends an empty multicast OACK to set the client active.  7. If having captured ack, OS sends out remain packet. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.46 | 0x2d4d9962, 0x24ac, 0x4f62, 0x9b, 0x66, 0x3c, 0xa5, 0xf3, 0x67, 0xb3, 0xa0 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when server doesn’t support option extension and just sends back the data packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If having captured the packet, OS side sends back the only one data packet then receives another ack. The return status must be EFI\_SUCCESS.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.47 | 0x9e9e85f5, 0x669d, 0x4de3, 0x82, 0xa4, 0xff, 0x96, 0xb9, 0x69, 0x79, 0x05 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the server doesn’t support multicast option and just doesn’t support multicast. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back an OACK without multicast option only and then OS should capture ack packet.  5. If having captured ack, OS sends back the only one data packet then receives another ack. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.48 | 0x0bdc47fc, 0x659e, 0x497f, 0x8d, 0x10, 0x10, 0x52, 0xd3, 0x95, 0x7d, 0x19 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the client continuously joins the group to download file, while the Active flag is set. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Do the step of 4,5,6 for 5 times:  4. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  5. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet.  6. If having captured ack, OS sends back the only data packet and then receives another ack.  The return status must be EFI\_SUCCESS.  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.49 | 0xc965cbdf, 0x1539, 0x4507, 0xb0, 0xd1, 0x4f, 0xcd, 0x17, 0xc4, 0xbb, 0x54 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the server sends back the Data with incorrect sequence of the block numbers. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet.  5. If having captured ack, OS sends back data packets with incorrect sequence of block numbers and then receives another ack. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.50 | 0xcf00a8ae, 0x8676, 0x4ee3, 0xb5, 0xcc, 0x82, 0x22, 0xf9, 0x46, 0x94, 0x03 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the server sends back the Data after some packets' retransmission. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet.  5. If having captured ack, server send backs the Number1 data packet and then receive another ack.  6. Then server doesn’t do anything until having received the fourth ack. Then it sends the rest data packets back. The return status must be EFI\_SUCCESS.  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.51 | 0x21a80b34, 0x73b3, 0x47ba, 0x82, 0x0c, 0x37, 0x34, 0x43, 0x7e, 0xd5, 0xd4 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when client downloads a file with length equal to 1 byte. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet.  5. If having captured ack, OS sends back the only data packet with length equal to 1 byte and then receives another ack. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.52 | 0x9e8004a9, 0xc28c, 0x461b, 0x84, 0xa1, 0x31, 0xca, 0xc6, 0x48, 0x31, 0x28 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when using OverrideData to replace the configuration data and retry counter is set to 0 in override data. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with OverrideData replacing the configuration data and retry counter set to 0 in override data. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active and then OS should capture ack packet.  5. If having captured ack, OS sends back the only data packet with length equal to 1 byte and then receives another ack. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.53 | 0x9bd82567, 0x6249, 0x4635, 0xb0, 0x2d, 0xf8, 0x06, 0x0d, 0x26, 0x68, 0xa6 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when active client receives data packets after server sends back OACK packet twice. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active; then OS should capture ack packet.  5. If having captured ack, OS send backs another OACK with the same option. Then sends back the only data packet and then receives another ack.  The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.54 | 0xcf35445d, 0x0aa1, 0x4485, 0x8e, 0xb6, 0x5f, 0xd8, 0xb4, 0x65, 0x55, 0x84 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the client is passive and it receives unexpected packets (BlockNo is a former number). | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive.  5. Then OS doesn’t sends back all the data packets in sequence. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.55 | 0x62908d19, 0xc308, 0x4f16, 0xa1, 0x70, 0xb6, 0x9a, 0xdf, 0x47, 0xb4, 0x72 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the client is passive and it receives unexpected packets (BlockNo is a further number). | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be passive.  5. Then OS doesn’t sends back all the data packets in sequence. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.56 | 0x58c614fb, 0x51d9, 0x4043, 0xb1, 0x24, 0x95, 0xa3, 0x7c, 0xcd, 0x3d, 0x70 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the server responds data packet with data length larger than blocksize. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back OACK with flag set to be active. In addition, OS should capture an ack.  5. If having captured it, OS sends back the first data packet with length larger than blocksize, then the rest data packet.  6. OS should capture ack. The return status must be EFI\_SUCCESS.  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.57 | 0x23a7aebe, 0x0117, 0x44fc, 0x9d, 0xcc, 0x68, 0x4c, 0xa6, 0x31, 0x2a, 0x20 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the client receives an unexpected ACK packet in the case of downloading file. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back an unexpected ACK and a normal OACK with flag set to be active.  5. Then if OS side has captured the ack, OS side sends back the only data packet.  6. OS should capture another ack. The return status must be EFI\_SUCCESS.  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.58 | 0x9df88b27, 0x0a20, 0x4d91, 0x98, 0x2b, 0x32, 0x26, 0x41, 0x62, 0x39, 0x44 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the active client receives an unexpected OACK packet in the case of downloading file. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be active.  5. If OS side has captured the ack, OS side send backs OACK again.  6. The server should capture another ack. Then the server sends the only data packet back to the client. The return status must be EFI\_SUCCESS.  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.59 | 0xad60cb28, 0x6451, 0x400a, 0xa5, 0x74, 0xf6, 0x35, 0x9f, 0x01, 0x92, 0xd3 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the passive client receives an unexpected OACK packet in the case of downloading file. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive.  5. Then server sends back the first data packet to the multicast IP address and another OACK to the client again.  6. Then the server sends the last data packet back to the multicast IP address. The return status must be EFI\_SUCCESS.  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.60 | 0x2309b8ea, 0x5593, 0x4835, 0xb6, 0x24, 0x65, 0xda, 0xc5, 0x51, 0x04, 0x5d | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the passive client downloads, it misses the last data packet. After client is timeout, server sets client to be passive again and sends out the lost packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive.  5. Then server sends back the first three data packets to the multicast IP address while missing the last data packet.  6. After passive client is timeout, it’ll send ack0 to ask for missing packets and the server should capture it.  7. If having captured the request, the server sends OACK back again and then the last data packet.  The return status must be EFI\_SUCCESS.  8. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.61 | 0xf6c81b41, 0x8edd, 0x46df, 0x8a, 0x82, 0x46, 0x40, 0xd9, 0x8b, 0xda, 0xa5 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the passive client downloads, it misses the last data packet. After client is timeout, server sets client to be passive again and sends out all the data packets. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive.  5. Then server sends back the first three data packets to the multicast IP address while missing the last data packet.  6. After passive client is timeout, it’ll send ack0 and the server should capture it.  7. If having captured the request, the server sends OACK back again and then all the data packets.  The return status must be EFI\_SUCCESS.  8. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.62 | 0x7156d37f, 0xd7ef, 0x47ea, 0xa2, 0xf3, 0x64, 0x3e, 0x7c, 0x44, 0x9f, 0x65 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the passive client downloads, it misses the first and the last packet. After client is timeout, server sets client to be passive again and sends out the lost packet | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive.  5. Then server sends back the second and the third data packet to the multicast IP address while missing the first and the last data packet.  6. After passive client is timeout, it’ll send ack0 and the server should capture it.  7. If having captured the request, the server sends OACK back again and then all the lost packets.  The return status must be EFI\_SUCCESS.  8. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.63 | 0x34753378, 0xb423, 0x40b1, 0x93, 0x7c, 0x4d, 0xaa, 0x5c, 0xa6, 0x63, 0x43 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the passive client downloads, it misses the first and the last packet. After client is timeout, server sets client to be passive again and sends out all the data packets. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive.  5. Then server sends back the second and the third data packet to the multicast IP address while missing the first and the last data packet.  6. After passive client is timeout, it’ll send ack0 and the server should capture it.  7. If having captured the request, the server sends OACK back again and then all the packets.  The return status must be EFI\_SUCCESS.  8. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.64 | 0xd756be67, 0xd667, 0x432f, 0xbb, 0xd6, 0x3a, 0xe1, 0xf5, 0xe6, 0x61, 0xd1 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the passive client downloads, it misses the first and the last packet. After client is timeout, server sets client to be active again and sends out the missing packets. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive.  5. Then server sends back the second and the third data packet to the multicast IP address while missing the first and the last data packet.  6. After passive client is timeout, it’ll send ack0 to ask for the missing packets.  7. If having captured the request, the server sends back OACK with flag set to be active and then the first data packet.  8. The server expects the ack packet to request the last packet. If having captured it, server will send the last data packet. The return status must be EFI\_SUCCESS.  9. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.65 | 0xc0fc889f, 0xc91f, 0x4a41, 0x80, 0x59, 0x0e, 0x22, 0x56, 0x79, 0x0b, 0x53 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the passive client downloads, it misses two blocks of packets. After client is timeout, server sets client to be passive again and sends out the lost packets randomly. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive.  5. Then server sends back the first and the seventh data packet to the multicast IP address while missing the Number2, 3, 4, 5, 6, 8 data packets.  6. After passive client is timeout, it’ll send ack0 and the server should capture it.  7. If having captured the ack0 packet, the server sends back OACK with flag set to be passive. Then it sends out the data packets randomly in the order Number4, 2, 6, 3, 5, 8.  8. The server expects the ack packet. If having captured, server will send the second data packet.  The return status must be EFI\_SUCCESS.  9. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.66 | 0x5a4ed7d1, 0x0e36, 0x4f9c, 0xa7, 0x9c, 0xf2, 0x35, 0x2e, 0xf7, 0x3b, 0x2d | EFI\_MTFTP4\_PROTOCOL.ReadFile() - returns EFI\_SUCCESS when the passive client downloads, it misses the first and the last packets. Then server changes the client to be active and retrieves its unicast transfer model. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side must capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back a normal OACK with flag set to be passive.  5. Then server sends back the second and the third data packets to the multicast IP address while missing the first and the last data packets.  6. After passive client is timeout, it’ll send ack0 and the server should capture it.  7. If having captured the request, the server sends back OACK with transfer mode changed to be unicast and flag set to be active. Then it expects the ack sent from the client and sends out the first data packet.  8. The server should capture the ack and then sends back the second packet.  9. As above, server sends the third and the last data packets. The return status must be EFI\_SUCCESS.  10. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.67 | 0xb441ee5b, 0xbf7f, 0x446f, 0xa2, 0x5c, 0x77, 0x7a, 0x0b, 0xdd, 0xde, 0x78 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - to test the EFI\_ICMP\_ERROR conformance of ReadFile() with an ICMP ERROR packet being received. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() and OS side capture the packet sent from EUT side.  4. If OS side has captured the packet, configure server to send back an ICMP error packet. The return status must be EFI\_ICMP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.68 | 0x6eaabf78, 0x3914, 0x4d08, 0x85, 0x0c, 0xbf, 0x63, 0x6d, 0xe9, 0xf3, 0x55 | EFI\_MTFTP4\_PROTOCOL.ReadFile() - to test the EFI\_INVALID\_PARAMETER conformance of ReadFile() when one or more options in Token.OptionList have wrong format. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadFile() with invalid muticast option value. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.7.69 | 0xd5e062fc, 0x5c0f, 0x470c, 0x8b,0x7a,0x44,0xf7, 0xbc,0xad, 0xc6,0x9c | EFI\_MTFTP4\_PROT  OCOL.ReadFile()- ReadFile()must return EFI\_NETWORK\_UNREACHABLEwhen receive an ICMP network unreachable packet. | 1. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create  a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call  EFI\_MTFTP4\_PROTOCOL.Configur  e() with all valid parameters.  3. Call  EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side  should capture the packet sent from  EUT side.  4. Configure Host side to send back an  ICMP network unreachable packet and the return status should be EFI\_NETWORK\_UNREACHABLE.  5. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to  destroy the newly created  EFI\_MTFTP4\_PROTOCOL child  handle and clean up the environment. |
| 5.26.2.7.70 | 0x6d8a5555, 0xe632, 0x470e, 0x98,0xe5,0x61,0xd2,0x2e,0xc9, 0x0d,0x0d | EFI\_MTFTP4\_PROT  OCOL.ReadFile()  - ReadFile() must return  EFI\_HOST\_UNREACHABLE when receive an ICMP host unreachable packet. | 1. Call  **EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild()** to create  a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call  EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call  EFI\_MTFTP4\_PROTOCOL.ReadFile  () with all valid parameters. OS side  should capture the packet sent from  EUT side.  4. Configure Host side to send back an  ICMP host unreachable packet and the return status should be EFI\_HOST\_UNREACHABLE.  5. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to  destroy the newly created EFI\_MTFTP4\_PROTOCOL child  handle and clean up the environment. |
| 5.26.2.7.71 | 0x732738e8, 0x1ff1, 0x4f3a, 0xa0,0xc8, 0x38,0x81,0x1d,0x15,0x92,0x83 | EFI\_MTFTP4\_PROT  OCOL.ReadFile()  - ReadFile() must return EFI\_PROTOCOL\_UNREACHABLEwhen receive an ICMP protocol unreachable packet. | 1. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create  a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call  EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call  EFI\_MTFTP4\_PROTOCOL.ReadFile()with all valid parameters. OS side  should capture the packet sent from  EUT side.  4. Configure Host side to send back an ICMP protocol unreachable packet and the return status should be EFI\_PROTOCOL\_UNREACHABLE.  5. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to  destroy the newly created  EFI\_MTFTP4\_PROTOCOL child  handle and clean up the environment. |
| 5.26.2.7.72 | 0xd1c4e1e8, 0x1099, 0x4646, 0xb7,0xc9, 0x64,0x7e, 0x65,0xc3, 0x82,0x30 | EFI\_MTFTP4\_PROTOCOL.ReadFile()- ReadFile() must return  **EFI\_PORT\_UNREACHABLE** when receive an ICMP port unreachable packet. | 1. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call  EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call  EFI\_MTFTP4\_PROTOCOL.ReadFile() with all valid parameters. OS side  should capture the packet sent from  EUT side.  4. Configure Host side to send back an  ICMP port unreachable packet and the return status should be EFI\_PORT\_UNREACHABLE.  5. Call  EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created  EFI\_MTFTP4\_PROTOCOL child  handle and clean up the environment. |

### WriteFile()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.2.8.1 | 0x4b00df17, 0xc244, 0x413d, 0x8e, 0xbf, 0xe8, 0x7e, 0x10, 0x9a, 0xa8, 0xd4 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_INVALID\_PARAMETER with a Token value of NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with a Token value of NULL. The return status must be EFI\_NVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.2 | 0xddc80d3b, 0x448d, 0x4ef9, 0xab, 0x74, 0x88, 0x47, 0xa7, 0xc9, 0x7c, 0xa8 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_INVALID\_PARAMETER with a Token->Filename value of NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with Token->Filename value of NULL.  The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.3 | 0x97304d43, 0x1101, 0x4b76, 0x90, 0x70, 0x66, 0x85, 0x62, 0x9e, 0xb3, 0xa3 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_INVALID\_PARAMETER when Token-> OptionCount is not 0 and Token->OptionList is NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() when Token-> OptionCount is not 0 and Token->OptionList is NULL.  The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.4 | 0xf061683f, 0xb39e, 0x42af, 0x92, 0x86, 0x9f, 0x18, 0xcc, 0xc7, 0xc0, 0x8d | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_INVALID\_PARAMETER when both Token->Buffer and Token-> PacketNeeded are NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() when both Token->Buffer and Token-> PacketNeeded are NULL. The return status must be EFI\_NVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.5 | 0xa2d02347, 0x9410, 0x49b3, 0xa9, 0xd2, 0xd7, 0x1a, 0xf4, 0xc5, 0xa7, 0x34 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_INVALID\_PARAMETER with an OverrideData.GatewayIp value of invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with an OverrideData.GatewayIp value of invalid. The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment.. |
| 5.26.2.8.6 | 0xe8f09c7b, 0x2cf3, 0x482e, 0x93, 0xc6, 0x4f, 0x45, 0x85, 0x3a, 0x43, 0x0c | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_INVALID\_PARAMETER with an OverrideData.ServerIp value of invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with an OverrideData**.**ServerIp value of invalid. The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.7 | 0x069921c9, 0x8f37, 0x45b6, 0xa4, 0x98, 0xa3, 0x2f, 0xc9, 0xb5, 0x8d, 0x50 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_INVALID\_PARAMETER when OverrideData.GatewayIp is not in the same subnet with *StationIp*. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() when OverrideData.GatewayIp is not in the same subnet with *StationIp*. The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.8 | 0xb95d36a6, 0x091e, 0x444b, 0x9d, 0xd7, 0x30, 0x4c, 0x9e, 0x59, 0xab, 0x81 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_NOT\_STARTED when the EFI MTFTPv4 Protocol driver having not been started. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with all valid parameters. The return status must be EFI\_NOT\_STARTED.  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.9 | 0x67021dd5, 0xf97d, 0x4783, 0x8d, 0xe2, 0x93, 0x6e, 0x6c, 0x5a, 0xe5, 0xeb | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_ACCESS\_DENIED when calling EFI\_MTFTP4\_PROTOCOL.WriteFile again before the first call ends. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() for the first time with all valid parameters.  4. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() for the second time with the same Token before the first call ends. The return status must be EFI\_ACCESS\_DENIED.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.10 | 0x4a445105, 0xf332, 0x4251, 0xb1, 0x5c, 0x10, 0x5c, 0x27, 0xeb, 0x67, 0x09 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_UNSUPPORTED when one or more options in the Token->OptionList are in the unsupported list for this implementation. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() when one or more options in the Token->OptionList are in the unsupported list for this implementation. The return status must be EFI\_UNSUPPORTED.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.11 | 0x00ee8108, 0xb8ce, 0x4428, 0x9a, 0x58, 0x3c, 0xf3, 0x33, 0x3e, 0xf4, 0x9a | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_TFTP\_ERROR when the client receives an MTFTPv4 ERROR packet during uploading. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with all valid parameters.  4. The server should capture the write request. If having captured the packet, send a normal OACK to the client.  5. Then OS side should capture the data packets. If having captured, OS side sends an error packet back. The return status must be EFI\_TFTP\_ERROR.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.12 | 0x0b05148f, 0x4f07, 0x413d, 0x8e, 0x47, 0x99, 0xbe, 0xac, 0x25, 0xc3, 0x4d | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_ICMP\_ERROR, when server sends back an ICMP error packet, client should terminate the session. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with all valid parameters.  4. The server should capture the write request. If having captured the packet, server responds an ICMP error packet. The return status must be EFI\_ICMP\_ERROR.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.13 | 0x26ac0f66, 0x2fa1, 0x4e91, 0x93, 0x14, 0xfe, 0x0f, 0x86, 0x93, 0x47, 0x4d | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_ABORTED when the user aborts the upload process in CheckPacket callback routine | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with CheckPacket callback set.  4. The server should capture the write request. If having captured the packet, server responds a normal OACK. The return status must be EFI\_ABORTED.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.14 | 0x105a5b0c, 0x72cb, 0x4854, 0x95, 0xdd, 0x86, 0xd7, 0x28, 0x0d, 0xa6, 0x12 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_ABORTED when the user aborts the upload process in TimeoutCallback callback routine | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with TimeoutCallback callback set.  4. The server should capture the write request. If having captured the packet, server responds a normal OACK. The return status must be EFI\_ABORTED.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.15 | 0xcaeef509, 0x3240, 0x4675, 0xa2, 0x50, 0x0b, 0xaf, 0xb5, 0x5a, 0xcb, 0x16 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_ABORTED when the user aborts the upload process in PacketNeeded callback routine | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with PacketNeeded callback set.  4. The server should capture the write request. If having captured the packet, server responds a normal OACK. The return status must be EFI\_ABORTED.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.25 | 0xb76d5034, 0xbee6, 0x468a, 0xa1, 0xf2, 0xc6, 0x9f, 0x20, 0x0d, 0xa6, 0xae | EFI\_MTFTP4\_PROTOCOL.WriteFile() - to test the EFI\_INVALID\_PARAMETER conformance of WriteFile when one or more options in Token.OptionList have wrong format. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with invalid timeout option value. The return status must be EFI\_INVALID\_PARAMETER  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.16 | 0xcc7a5aad, 0xe6ec, 0x4fa7, 0x97, 0x0a, 0xac, 0x30, 0xd6, 0x39, 0x20, 0x16 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_SUCCESS when the user uploads a packet with data less than one block. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with BufferSize set to 100.  4. The server should capture the write request. If having captured the packet, server responds a normal OACK.  5. The server should capture the only data packet sent from the client and respond ACK.  The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.17 | 0x2649936f, 0x161c, 0x40c2, 0xa8, 0x53, 0xc0, 0xa4, 0xa3, 0x2e, 0xf2, 0x62 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_SUCCESS when the user uploads a packet with data length equal to 1 byte. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with BufferSize set to 1.  4. The server should capture the write request. If having captured the packet, server responds a normal OACK.  5. The server should capture the only data packet sent from the client and respond with an ACK packet. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.18 | 0xbcbec9fd, 0x00d8, 0x494d, 0xa4, 0xff, 0x86, 0x98, 0xc4, 0xb0, 0x6a, 0x5a | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_SUCCESS when the user uploads a packet with override configuration data. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with override configuration data.  4. The server should capture the write request. If having captured the packet, server responds a normal OACK.  5. The server should capture the only data packet sent from the client and respond with an ACK packet. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.19 | 0x741101e7, 0x7888, 0x4bd8, 0xa2, 0xcb, 0x1d, 0xec, 0xb1, 0x34, 0x66, 0x31 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_SUCCESS when the server responds with an incorrect ack packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with all valid parameters.  4. The server should capture the write request. If having captured the packet, server responds an OACK.  5. The server should capture the only data packet sent from the client and responds with an incorrent ACK to the incorrect packet number followed by a correct ACK. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.20 | 0xa3c22b82, 0x5f14, 0x4419, 0x8f, 0xc6, 0xd7, 0x89, 0x88, 0xa9, 0x88, 0xe9 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_SUCCESS when the server responds WRQ with an ACK instead of OACK, so client sends data packet to server. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with all valid parameters.  4. The server should capture the write request. If having captured the packet, server responds with an ACK instead of an OACK.  5. The server should capture the only data packet sent from the client and respond with an ACK to this packet. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.21 | 0x20787b06, 0x8766, 0x4ced, 0xb0, 0x25, 0x65, 0xfa, 0xf1, 0xd3, 0x6c, 0x7c | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_SUCCESS when the server replies WRQ with an invalid BlockNo ACK instead of OACK, client should ignore this packet and continue the normal process. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with all valid parameters.  4. The server should capture the write request. If having captured the packet, server responds with an invalid BlockNo ACK instead of an OACK.  5. The server should capture the only data packet sent from the client and responds with an ACK to this packet. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.22 | 0xf549a91c, 0x9d15, 0x45c7, 0xb2, 0xed, 0xa6, 0x7e, 0xff, 0x08, 0xc0, 0xf4 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_SUCCESS when the server replies DATA packet with an error ACK ( BufferLen < sizeof(UINT16)) instead of OACK, client should ignore this packet and continue the normal process. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with all valid parameters.  4. The server should capture the write request. If having captured the packet, server responds with a normal OACK.  5. The server should capture the only data packet sent from the client and replies with an error ACK ( BufferLen < sizeof(UINT16)) and a correct ACK , client should ignore this error ACK and continue the normal process.  The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.23 | 0x9ee2172f, 0xb96e, 0x4d13, 0x9e, 0x6c, 0xbd, 0x27, 0x44, 0x95, 0xee, 0xc6 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_SUCCESS when the client receives an unexpected OACK when waiting for ACK packet during uploading file. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with all valid parameters.  4. The server should capture the write request. If having captured the packet, server responds with ACK.  5. The server should capture the only data packet sent from the client and respond with an unexpected OACK and an ACK to this packet. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.8.24 | 0x434974c8, 0x5f8c, 0x46d8, 0x89, 0x57, 0x4e, 0x03, 0xff, 0xfa, 0xa3, 0xc5 | EFI\_MTFTP4\_PROTOCOL.WriteFile() - returns EFI\_SUCCESS when the client receives an error server source port ACK in the case of waiting for ACK packet. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.WriteFile() with all valid parameters.  4. The server should capture the write request. If having captured the packet, server responds with a normal OACK.  5. The server should capture the only data packet sent from the client and replies with an error server source port ACK and a correct ACK for the packet; client should ignore this error ACK and continue the normal process. The return status must be EFI\_SUCCESS.  6. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |

### ReadDirectory()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.2.9.1 | 0xc9e02ded, 0x0e98, 0x4162, 0x8d, 0x4c, 0x14, 0x58, 0xd0, 0x6a, 0xc7, 0xab | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - returns EFI\_INVALID\_PARAMETER with a Token value of NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() with a Token value of NULL. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.9.2 | 0x120fa0f3, 0xad22, 0x4d39, 0xb9, 0x00, 0xe5, 0x60, 0xdd, 0x8f, 0xe3, 0xb2 | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - returns EFI\_INVALID\_PARAMETER with a Token->Filename value of NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() with a Token->Filename value of NULL.  The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.9.3 | 0xe6339187, 0x07d0, 0x467f, 0x9b, 0x89, 0x5b, 0xf5, 0x6c, 0x2d, 0xf8, 0xe0 | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - returns EFI\_INVALID\_PARAMETER when Token- > OptionCount is not 0 and Token-> OptionList is NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() when Token- >OptionCount is not 0 and Token-> OptionList is NULL.  The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.9.4 | 0xc39cb583, 0x3fa4, 0x4c7f,0x9a, 0x93, 0xa5, 0xf9, 0x30, 0xf0, 0x42, 0x6c | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - returns EFI\_INVALID\_PARAMETER when both Token->Buffer and Token->CheckPacket are NULL. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() when both Token->Buffer and Token->CheckPacket are NULL.  The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.9.5 | 0xaf58aaf5, 0x3cd0, 0x47aa, 0x8b, 0x93, 0x4f, 0x7b, 0x8b, 0xe8, 0x4d, 0xf1 | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - returns EFI\_INVALID\_PARAMETER with an OverrideData.GatewayIp value of invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() with an OverrideData**.GatewayIp** value of invalid. The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.9.6 | 0x7044e68a, 0x6ca9, 0x4b23, 0x9a, 0x50, 0x91, 0x85, 0x34, 0xa3, 0xca, 0xfb | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - returns EFI\_INVALID\_PARAMETER with an OverrideData.ServerIp value of invalid. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() with an OverrideData**.ServerIp** value of invalid. The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.9.7 | 0x8bd21805, 0xec3c, 0x4041, 0xa4, 0xe4, 0x75, 0xf1, 0xa4, 0xec, 0xae, 0x4d | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - returns EFI\_INVALID\_PARAMETER when OverrideData.GatewayIp is not in the same subnet with *StationIp*. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() for OverrideData.GatewayIp is not in the same subnet with*StationIp* .  The return status must be  EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.9.8 | 0x7ecf38c4, 0x4fc5, 0x4663, 0xa4, 0xc4, 0xc0, 0x48, 0x45, 0xfe, 0x59, 0x6b | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - returns EFI\_NOT\_STARTED while the EFI MTFTPv4 Protocol driver having not been started. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() with all valid parameters.  The return status must be EFI\_NOT\_STARTED.  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.9.9 | 0x31599521, 0xb38b, 0x47c8, 0xa6, 0x39, 0xaf, 0x50, 0xe3, 0x30, 0xbe, 0x87 | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - returns EFI\_UNSUPPORTED when one or more options in the a Token->OptionList value of in the unsupported list of this implementation. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() with one or more options in the a Token->OptionList value of in the unsupported list of this implementation.  The return status must be EFI\_UNSUPPORTED.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.9.10 | 0xefc6d249, 0x179f, 0x49a2, 0x96, 0x1c, 0x0d, 0x90, 0xe7, 0x79, 0x4c, 0xcb | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - returns EFI\_ICMP\_ERROR when the server responds with an ICMP error packet, client should terminate the session. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() with all valid parameters.  4. If OS side has captured the request, it sends out an ICMP error packet.  The return status must be EFI\_ICMP\_ERROR.  5. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.9.12 | 0xab9bacfb, 0x79ee, 0x41e5, 0xb9, 0xe9, 0x40, 0x31, 0x7a, 0xf1, 0xcc, 0x64 | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - test the EFI\_INVALID\_PARAMETER conformance of ReadDirectory() when one or more options in Token.OptionList have wrong format. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() with invalid timeout option value. The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |
| 5.26.2.9.11 | 0x968731a3, 0x01e8, 0x44d7, 0xad, 0xba, 0x70, 0x88, 0x80, 0x8c, 0x99, 0xe1 | EFI\_MTFTP4\_PROTOCOL.ReadDirectory() - returns EFI\_SUCCESS - read a list of files on the MTFTPv4 server that are logically (or operationally) related to Token->FileName. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_MTFTP4\_PROTOCOL.ReadDirectory() with all valid parameters.  4. If OS side has captured the request, it sends out a normal OACK.  5. Then OS side should capture the ack from the client and send back the only data packet.  6. Then OS side expects another ack.  The return status must be EFI\_ICMP\_ERROR.  7. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |

### Poll()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.2.10.1 | 0x57e97972, 0xa7a3, 0x4647, 0x95, 0x9a, 0x23, 0x29, 0x5b, 0x81, 0x2c, 0xfe | EFI\_MTFTP4\_PROTOCOL.Poll() - returns EFI\_NOT\_STARTED when the EFI MTFTPv4 Protocol driver having not been started. | 1. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new EFI\_MTFTP4\_PROTOCOL child handle.  2. Call EFI\_MTFTP4\_PROTOCOL.Poll() with all valid parameters. The return status must be EFI\_NOT\_STARTED.  3. Call EFI\_MTFTP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created EFI\_MTFTP4\_PROTOCOL child handle and clean up the environment. |

## EFI\_UDP6\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_UDP6\_PROTOCOL Section.

### CreateChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.3.1.1 | 0x1d3e7323, 0x5a46, 0x4fe3, 0xbf, 0x9d, 0x0a, 0xb8, 0xb1, 0xfd, 0xe7, 0x92 | EFI\_UDP6\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - CreateChild()returns EFI\_INVALID\_PARAMETER with a NULL ChildHandle. | Call CreateChild()with a NULL ChildHandle, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.1.2 | 0x8872614e, 0x51d5, 0x434d, 0xb8, 0x71, 0x20, 0x30, 0x4f, 0xbe, 0x04, 0x92 | EFI\_UDP6\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - CreateChild()returns EFI\_SUCCESS with a valid ChildHandle. | Call CreateChild()with a valid ChildHandle, the return status should be EFI\_SUCCESS. |

### DestoryChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.3.2.1 | 0x25c39b09, 0xba61, 0x49f3, 0xa3, 0x58, 0x98, 0x11, 0x17, 0xd8, 0x14, 0x0e | EFI\_UDP6\_SERVICE\_BINDING\_PROTOCOL.DestoryChild() - DestoryChild() returns EFI\_INVALID\_PARAMETER with a NULL ChildHandle. | Call DestoryChild()with a NULL ChildHandle, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.2.2 | 0x1e938ebd, 0x425a, 0x4eb6, 0xbd, 0x12, 0x9c, 0xa2, 0xdc, 0xc4, 0x0b, 0x4c | EFI\_UDP6\_SERVICE\_BINDING\_PROTOCOL.DestoryChild() - DestoryChild()returns EFI\_SUCCESS with a valid ChildHandle. | Call DestoryChild()with a valid ChildHandle, the return status should be EFI\_SUCCESS. |

### GetModeData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.3.3.1 | 0x920b75d9, 0xba94, 0x4e72, 0xb0, 0x4d, 0x77, 0xe5, 0x81, 0xe7, 0xcf, 0x91 | EFI\_UDP6 PROTOCOL.GetModeData() - GetModeData() returns EFI\_NOT\_STARTED with a not configured ChildHandle | Call GetModeData()with a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED. |
| 5.26.3.3.2 | 0x1a823790, 0xcaec, 0x413d, 0xbc, 0xf3, 0xe7, 0xfa, 0x70, 0xdf, 0x87, 0x6d | EFI\_UDP6 PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters | 5.26.3.3.2 to 5.26.3.3.4 belong to one case  1. Call GetModeData()with valid parameters, the return status should be EFI\_ SUCCESS. |
| 5.26.3.3.3 | 0xdb72ffca, 0xd3d9, 0x4837, 0x8f, 0x39, 0xf9, 0x67, 0x2e, 0x9d, 0x93, 0xab | Validate the IP6ModeData.IsConfigured | 2. The value of IP6ModeData.IsConfigured should be TRUE. |
| 5.26.3.3.4 | 0x923aecf2, 0xcfc6, 0x4497, 0x8c, 0x49, 0xe6, 0x74, 0x1c, 0x60, 0xc7, 0x66 | Validate the Udp6ConfigData | 3. The value of Udp6ConfigData should be same with the assigned configure data. |

### Configure()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.3.4.1 | 0x1c36e6e8, 0xf453, 0x41bb, 0x84, 0x6f, 0x0a, 0x67, 0x91, 0xa6, 0xe5, 0xe7 | EFI\_UDP6 PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER with a UdpConfigData.StationAddress being neither zero nor one of the configured IP addresses in the underlying IPv6 driver | Call Configure()with a UdpConfigData.StationAddress being neither zero nor one of the configured IP addresses in the underlying IPv6 driver, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.4.2 | 0xef302465, 0x7ec6, 0x4652, 0xbb, 0xf0, 0x62, 0x73, 0xa5, 0x5a, 0xd5, 0x52 | EFI\_UDP6 PROTOCOL.Configure() - Configure() returns EFI\_INVALID\_PARAMETER with a UdpConfigData.RemoteAddressbeing an invalid unicast IPv6 address if it is not zero. | Call Configure()with a UdpConfigData RemoteAddressbeing an invalid unicast IPv6 address if it is not zero, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.4.3 | 0xe146a746, 0x2985, 0x4a7b, 0x92, 0xa5, 0x08, 0x44, 0x8d, 0x41, 0x69, 0x03 | EFI\_UDP6 PROTOCOL.Configure() - Configure() returns EFI\_ALREADY\_STARTED with a ChildHandle instance has already been started/configured. | Call Configure()with a ChildHandle instance has already been started/configured, the return status should be EFI\_ALREADY\_STARTED. |
| 5.26.3.4.4 | 0x3522ad76, 0xe7aa, 0x4477, 0x9a, 0x41, 0xb7, 0xdc, 0xd6, 0xff, 0x7f, 0xf2 | EFI\_UDP6 PROTOCOL.Configure() - Configure() returns EFI\_ACCESS\_DENIED with UdpConfigData. AllowDuplicatePort is FALSE and UdpConfigData.StationPort is already used by other instance. | Call Configure() with UdpConfigData. AllowDuplicatePort is FALSE and UdpConfigData.StationPort is already used by other instance, the return status should be EFI\_ACCESS\_DENIED. |
| 5.26.3.4.5 | 0x370fcb11, 0x68de, 0x4c01, 0xb0, 0xce, 0x64, 0x53, 0xb0, 0x94, 0x8f, 0xb5 | EFI\_UDP6 PROTOCOL.Configure() - Configure() returns EFI\_SUCCESS with valid parameters | 5.26.3.4.5 to 5.26.3.4.9 belong to one case.  1. Call Configure()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.26.3.4.6 | 0xd6c84689, 0x0df8, 0x4f69, 0xa6, 0xd0, 0x76, 0x92, 0x89, 0xd0, 0x7d, 0x20 | EFI\_UDP6 PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters | 2. Call GetModeData()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.26.3.4.7 | 0x7c2f3112, 0x80e9, 0x4b59, 0x98, 0xd5, 0x06, 0x25, 0x8e, 0x3e, 0x5f, 0x9f | Validate the IP6ModeData.IsConfigured and Udp6ConfigData | 3. The value of IP6ModeData.IsConfigured should be TRUE. The value of Udp6ConfigData should be same with the assigned configure data. |
| 5.26.3.4.8 | 0xc3fbe729, 0x3f1d, 0x41df, 0x83, 0x66, 0x6f, 0x50, 0x45, 0xf7, 0xce, 0x74 | EFI\_UDP6 PROTOCOL.Configure() - Configure() returns EFI\_SUCCESS with a NULL Udp6ConfigData | 4. Call Configure() with a NULL Udp6ConfigData, the return status should be EFI\_SUCCESS. |
| 5.26.3.4.9 | 0xd5a2273d, 0x33f4, 0x4f98, 0xb0, 0x8e, 0x9a, 0xd4, 0xec, 0x49, 0x9c, 0x76 | EFI\_UDP6 PROTOCOL.GetModeData() - GetModeData() returns EFI\_NOT\_STARTED with valid parameters | 5. Call GetModeData()with valid parameters, the return status should be EFI\_NOT\_STARTED. |

### Groups()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.3.5.1 | 0x1f290403, 0xaa9e, 0x4e3b, 0x94, 0xfb, 0x2d, 0x2b, 0xa0, 0x56, 0x6b, 0x22 | EFI\_UDP6 PROTOCOL.Groups() - Groups() returns EFI\_NOT\_STARTED with a not configured ChildHandle | Call Groups()with a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED. |
| 5.26.3.5.2 | 0xb1fd2421, 0x6e59, 0x4987, 0xb8, 0x28, 0x1c, 0x13, 0xb1, 0xe3, 0x60, 0x37 | EFI\_UDP6 PROTOCOL.Groups() - Groups() returns EFI\_INVALID\_PARAMETER with TRUE JoinFlag and an invalid MulticaseAddress | Call Groups() with TRUE JoinFlag and an invalid MulticaseAddress, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.5.3 | 0xd2d32833, 0x51b6, 0x4c1b, 0x9a, 0x1c, 0x08, 0x11, 0xe6, 0xc1, 0xef, 0x4a | EFI\_UDP6 PROTOCOL.Groups() - Groups() returns EFI\_ALREADY\_STARTED with TRUE JoinFlag and an MulticaseAddress which has already been in the group table. | Call Groups()with TRUE JoinFlag and an MulticaseAddress which has already been in the group table, the return status should be EFI\_ALREADY\_STARTED. |
| 5.26.3.5.4 | 0x68c084c2, 0x55ef, 0x488a, 0x93, 0x24, 0xf9, 0x7b, 0x64, 0xbc, 0xbf, 0x03 | EFI\_UDP6 PROTOCOL.Groups() - Groups() returns EFI\_NOT\_FOUND with FALSE JoinFlag and an MulticaseAddress which is not in the group table. | 5.26.3.5.4 to 5.26.3.5.7 belong to one case.  1. Call Groups()with FALSE JoinFlag and an MulticaseAddress which is not in the group table, the return status should be EFI\_NOT\_FOUND. |
| 5.26.3.5.5 | 0xf16ff0fc, 0x074a, 0x460e, 0xa1, 0x11, 0x5f, 0x9e, 0xd3, 0x35, 0x9c, 0xac | EFI\_UDP6 PROTOCOL.Groups() - Groups() returns EFI\_SUCCESS with TRUE JoinFlag and an MulticaseAddress which is not in the group table. | 2. Call Groups()with TRUE JoinFlag and an MulticaseAddress which is not in the group table, the return status should be EFI\_SUCCESS. |
| 5.26.3.5.6 | 0x60253644, 0x6c0e, 0x4662, 0xbd, 0x4c, 0x63, 0xc8, 0xde, 0xb1, 0x0c, 0x21 | EFI\_UDP6 PROTOCOL.Groups() - Groups() returns EFI\_SUCCESS with FALSE JoinFlag and an MulticaseAddress which has been inserted in the group table. | 3. Call Groups()with FALSE JoinFlag and an MulticaseAddress which has been inserted in the group table, the return status should be EFI\_SUCCESS. |
| 5.26.3.5.7 | 0x5200ac0c, 0x0adb, 0x4a14, 0xa8, 0xbf, 0xbd, 0x42, 0xeb, 0x68, 0x2d, 0x8e | EFI\_UDP6 PROTOCOL.Groups() - Groups() returns EFI\_NOT\_FOUND with FALSE JoinFlag and an MulticaseAddress which has been removed from the group table. | 4. Call Groups()with FALSE JoinFlag and an MulticaseAddress which has been removed from the group table, the return status should be EFI\_NOT\_FOUND. |
| 5.26.3.5.8 | 0x05df343c, 0xaff4, 0x4dc5, 0x8b, 0xa5, 0xd7, 0x76, 0x63, 0x12, 0x89, 0x25 | EFI\_UDP6 PROTOCOL.Groups() - Groups() returns EFI\_SUCCESS with TRUE JoinFlag and an MulticaseAddress which is not in the group table. | 5.26.3.5.8 to 5.26.3.5.11 belong to one case.  1. Call Groups() with TRUE JoinFlag and an MulticaseAddress which is not in the group table, the return status should be EFI\_SUCCESS. |
| 5.26.3.5.9 | 0x24602ea3, 0x6bb2, 0x49cf, 0xac, 0x38, 0xb0, 0x13, 0x85, 0x5c, 0xc8, 0xb9 | EFI\_UDP6 PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters. Check the Ip6ModeData.GroupCount and Ip6ModeData.GroupTable | 2. Call GetModeData()with valid parameters, the return status should be EFI\_SUCCESS. Ip6ModeData.GroupCount and Ip6ModeData.GroupTable should be reasonable. |
| 5.26.3.5.10 | 0x6aabe731, 0x0de1, 0x4643, 0x82, 0x4e, 0x18, 0x0c, 0x65, 0x4a, 0xac, 0x0c | EFI\_UDP6 PROTOCOL.Groups() - Groups() returns EFI\_SUCCESS with FALSE JoinFlag and an MulticaseAddress which has been inserted in the group table. | 3. Call Groups()with FALSE JoinFlag and an MulticaseAddress which has been inserted in the group table, the return status should be EFI\_SUCCESS. |
| 5.26.3.5.11 | 0xe9d7c7e6, 0xfc75, 0x48ef, 0xb9, 0x46, 0x00, 0xda, 0x5d, 0xe4, 0xcd, 0xea | EFI\_UDP6 PROTOCOL.GetModeData() - GetModeData() returns EFI\_SUCCESS with valid parameters. Check the Ip6ModeData.GroupCount | 4. Call GetModeData()with valid parameters, the return status should be EFI\_SUCCESS. Ip6ModeData.GroupCount should be reasonable. |

### Transmit()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.3.6.1 | 0x845b6a05, 0x23f3, 0x4c4f, 0x8d, 0xbc, 0xc0, 0xd3, 0x69, 0x9b, 0x76, 0x46 | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_NOT\_STARTED with a not configured ChildHandle | Call Transmit()with a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED. |
| 5.26.3.6.2 | 0x71c15402, 0x7d5c, 0x4b8c, 0xb9, 0xa5, 0xfd, 0xe5, 0x3e, 0x68, 0xed, 0x22 | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a NULL Token | Call Transmit()with a NULL Token, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.6.3 | 0x12795cad, 0xdbbe, 0x41cd, 0x84, 0x57, 0x5f, 0xae, 0x7d, 0x72, 0x07, 0x2a | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a NULL Token->Event | Call Transmit()with a NULL Token->Event, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.6.4 | 0xbfcd7c31, 0xcb6f, 0x4cfd, 0xb9, 0xe2, 0x01, 0xd7, 0x5c, 0x6b, 0x44, 0xfa | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a NULL Token->Packet.TxData | Call Transmit()with a NULL Token->Packet.TxData, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.6.5 | 0x4c71fbec, 0x6cc6, 0x4cac, 0x89, 0x74, 0x67, 0xb5, 0x27, 0xbe, 0xef, 0xa3 | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with Token->Packet.TxData->FragmentCount is Zero | Call Transmit()with Token->Packet.TxData->FragmentCount is Zero, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.6.6 | 0xe0e3d058, 0xbdc3, 0x4ed2, 0x9c, 0x39, 0xea, 0x10, 0x6b, 0xe5, 0xea, 0x7a | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with Token->Packet.TxData->FragmentTable[0].FragmentLength is Zero | Call Transmit()with Token->Packet.TxData->FragmentTable[0].FragmentLength is Zero, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.6.7 | 0xbacc7fd3, 0x9a5c, 0x4ae6, 0xb6, 0xb3, 0x7f, 0x95, 0xc7, 0xda, 0xc4, 0xa2 | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a NULL Token->Packet.TxData->FragmentTable[0].FragmentBuffer | Call Transmit()with a NULL Token->Packet.TxData->FragmentTable[0].FragmentBuffer, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.6.8 | 0xf062269b, 0x66bb, 0x426a, 0x8e, 0xeb, 0x06, 0xd3, 0x0c, 0xd3, 0x30, 0x16 | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with an invalid Token->Packet.TxData->DataLength | Call Transmit()with an invalid Token->Packet.TxData->DataLength which is not equal to the sum of the fragments length, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.6.9 | 0x5a3af347, 0xdf8a, 0x4a67, 0x80, 0x32, 0xa7, 0xd0, 0xa8, 0xcc, 0x2f, 0x97 | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a non-zero Token->Packet.TxData->Udp6sessionData->DestinationAddress which is not specified in Configure process | Call Transmit()with a non-zero Token->Packet.TxData->Udp6sessionData->DestinationAddress which is not specified in Configure process, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.6.10 | 0x52218200, 0xfffd, 0x4b78, 0x8b, 0x2b, 0xec, 0x17, 0x56, 0x2c, 0x3f, 0xd7 | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a zero Token->Packet.TxData->Udp6sessionData->DestinationAddress when DestinationAddressis unspecified when doing Configure process | Call Transmit()with a zero Token->Packet.TxData->Udp6sessionData->DestinationAddress when DestinationAddressis unspecified when doing Configure process, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.6.11 | 0x97434d51, 0x8e06, 0x49e9, 0x95, 0xd0, 0xfc, 0x3a, 0x03, 0xf9, 0x9c, 0xee | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_INVALID\_PARAMETER with a NULL Token->Packet.TxData->Udp6sessionData and the instance’s UdpConfigData.RemoteAddress is unspecified. | Call Transmit()with a NULL Token->Packet.TxData->Udp6sessionData and the instance’s UdpConfigData.RemoteAddress is unspecified, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.6.12 | 0x31b5da9f, 0xd866, 0x43c7, 0x8c, 0x2b, 0xf8, 0xd9, 0x7c, 0x5b, 0xdb, 0x12 | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_ACCESS\_DENIED with a Token->Event which has already been in the transmit queue. | Call Transmit()with a Token->Event which has already been in the transmit queue, the return status should be EFI\_ACCESS\_DENIED. |
| 5.26.3.6.13 | 0x99e6bfb0, 0x903b, 0x4c6c, 0xa4, 0x6c, 0x9e, 0x51, 0x23, 0xdb, 0xdd, 0x4b | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_BAD\_BUFFER\_SIZE with a Token->Packet.TxData->DataLength which beyond the maximum udp6 packet size. | Call Transmit()with a Token->Packet.TxData->DataLength which beyond the maximum udp6 packet size, the return status should be EFI\_BAD\_BUFFER\_SIZE. |
| 5.26.3.6.14 | 0xaf040d05, 0xf0e3, 0x4348, 0x8f, 0x1d, 0xd9, 0x99, 0x90, 0xc7, 0x3d, 0x06 | EFI\_UDP6 PROTOCOL.Transmit() - Transmit() returns EFI\_SUCCESS with valid parameters. | 5.26.3.6.14 to 5.26.3.6.17 belong to one case.  1. Call Transmit()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.26.3.6.15 | 0x930f3d18, 0x3261, 0x4d17, 0xa3, 0xc0, 0x0d, 0xd1, 0xa6, 0x5d, 0x10, 0xe1 | Token->Event should be signnaled | Token->Event should be signaled. |
| 5.26.3.6.16 | 0x93873bee, 0x2136, 0x432e, 0xb0, 0x8f, 0xd7, 0x9d, 0xd9, 0xf9, 0xcf, 0x04 | Token->Status should be EFI\_SUCCESS | Token->Status should be EFI\_SUCCESS. |
| 5.26.3.6.17 | 0x30ca402a, 0xed8a, 0x4c69, 0x94, 0x7f, 0xa0, 0x4c, 0xd1, 0xbb, 0xaa, 0x58 | The received packet content should be reasonable. | The received packet content should be reasonable. |

### Receive()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.3.7.1 | 0xb5c83b2c, 0x66c1, 0x4ea5, 0xba, 0x41, 0x6c, 0xc4, 0x85, 0xb2, 0x58, 0xaf | EFI\_UDP6 PROTOCOL.Receive() - Receive() returns EFI\_NOT\_STARTED with a not configured ChildHandle | Call Receive()with a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED. |
| 5.26.3.7.2 | 0xc5c9fd31, 0xf095, 0x473f, 0xaf, 0x53, 0x87, 0x16, 0xc8, 0x51, 0x58, 0x9d | EFI\_UDP6 PROTOCOL.Receive() - Receive() returns EFI\_INVALID\_PARAMETER with a NULL Token | Call Receive()with a NULL Token, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.7.3 | 0xa8916a19, 0xecf7, 0x4392, 0xa1, 0x65, 0xc0, 0x6e, 0x1b, 0xff, 0xc1, 0xe6 | EFI\_UDP6 PROTOCOL.Receive() - Receive() returns EFI\_INVALID\_PARAMETER with a NULL Token->Event | Call Receive()with a NULL Token->Event, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.7.14 | 0x17a43441, 0x0701, 0x446b, 0xab, 0x37, 0x4c, 0xd9, 0x23, 0xcf, 0xc1, 0x43 | EFI\_UDP6 PROTOCOL.Receive() – Receive() returns EFI\_ACCESS\_DENIED with a Token->Event which has already been in the transmit queue. | Call Receive()with a Token->Event which has already been in the transmit queue, the return status should be EFI\_ACCESS\_DENIED. |
| 5.26.3.7.5 | 0x3166ca55, 0x6f3f, 0x4748, 0xbc, 0x48, 0xf7, 0xb6, 0x86, 0x35, 0x9d, 0xcc | EFI\_UDP6 PROTOCOL.Receive() – Receive() returns EFI\_SUCCESS with valid parameters. | 5.26.3.7.5 to 5.26.3.7.8 belong to one case.  1. Call Receive()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.26.3.7.6 | 0xb5e37f49, 0xc13a, 0x4c80, 0x9d, 0x37, 0x9b, 0xb6, 0x96, 0xb8, 0x14, 0xe7 | Token->Event should be signaled | Token->Event should be signaled. |
| 5.26.3.7.7 | 0x96a78bb2, 0x8d5d, 0x4ed1, 0x9e, 0xc5, 0xc7, 0x34, 0x28, 0x61, 0x1e, 0x7d | The received packet content should be reasonable. | The received packet content should be reasonable. |
| 5.26.3.7.8 | 0x90b87634, 0x1da5, 0x4f26, 0x8c, 0x78, 0x82, 0xba, 0xd5, 0x4a, 0xc8, 0xfe | Token->Status should be EFI\_SUCCESS | Token->Status should be EFI\_SUCCESS. |

### Cancel()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.3.8.1 | 0xd0aafd24, 0xa340, 0x40f4, 0xba, 0x48, 0xe1, 0x59, 0x86, 0xc7, 0x78, 0x79 | EFI\_UDP6 PROTOCOL.Cancel() – Cancel() returns EFI\_NOT\_STARTED with a not configured ChildHandle | Call Cancel()with a not configured ChildHandle, the return status should be EFI\_NOT\_STARTED. |
| 5.26.3.8.2 | 0x063478c3, 0x207d, 0x4b82, 0x96, 0xf5, 0x0f, 0xbf, 0xee, 0x2f, 0xac, 0x5f | EFI\_UDP6 PROTOCOL.Cancel() – Cancel() returns EFI\_NOT\_FOUND with a Token which hasn’t been inserted into both transmit and receive queue. | Call Cancel()with a Token which hasn’t been inserted into both transmit and receive queue, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.8.3 | 0xed1466df, 0xccc6, 0x412e, 0xbe, 0xda, 0xb9, 0x87, 0xc8, 0x37, 0x2b, 0x6f | EFI\_UDP6 PROTOCOL.Cancel() – Cancel() returns EFI\_NOT\_FOUND with a Token which has been removed into both transmit and receive queue. | Call Cancel()with a Token which has been removed into both transmit and receive queue, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.3.8.4 | 0xebe8e81e, 0x632c, 0x4aa9, 0xa8, 0x50, 0x27, 0xb7, 0x32, 0x63, 0xd0, 0x62 | EFI\_UDP6 PROTOCOL.Cancel() – Cancel() returns EFI\_SUCCESS with valid parameters. | 5.26.3.8.4 to 5.26.3.8.6 belong to one case.  1. Call Cancel()with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.26.3.8.5 | 0x616b87c1, 0xa5f9, 0x4195, 0x81, 0x38, 0x9c, 0xb8, 0xcd, 0x3c, 0x64, 0x50 | Token->Event should be signaled | Token->Event should be signaled. |
| 5.26.3.8.6 | 0x1280bba6, 0x5d60, 0x43ae, 0xba, 0x36, 0x2d, 0xce, 0x08, 0x79, 0x5e, 0x57 | Token->Status should be EFI\_SUCCESS | Token->Status should be EFI\_SUCCESS. |

## EFI\_MTFTP6\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_MTFTP6\_PROTOCOL Section.

### CreateChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.4.1.1 | 0xed279b2f, 0x0fb1, 0x4f84, 0x8c, 0x11, 0x69, 0x36, 0x88, 0x0f, 0x94, 0x48 | EFI\_MTFTP6\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - CreateChild()returns EFI\_INVALID\_PARAMETER with a NULL ChildHandle. | Call CreateChild()with a NULL ChildHandle, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.1.2 | 0x758b358d, 0x4bf0, 0x4bcc, 0x82, 0x6c, 0xe4, 0xad, 0x40, 0xe8, 0x29, 0x6e | EFI\_MTFTP6\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - CreateChild() returns EFI\_SUCCESS with the 1st validChildHandle. | 5.26.4.1.2 to 5.26.4.1.5 belong to one case.  1. Call CreateChild()with the 1st valid ChildHandle, the return status should be EFI\_SUCCESS. |
| 5.26.4.1.3 | 0x5446dbb2, 0xbf0b, 0x4685, 0x88, 0xf4, 0x3b, 0x14, 0x3e, 0x2b, 0xdd, 0x1b | EFI\_MTFTP6\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - CreateChild() returns EFI\_SUCCESS with the 2nd validChildHandle. | 2. Call CreateChild()with the 2nd valid ChildHandle, the return status should be EFI\_SUCCESS. |
| 5.26.4.1.4 | 0x6a61e0bd, 0xd760, 0x4788, 0x85, 0xc9, 0x4b, 0x45, 0xe2, 0x9e, 0x7e, 0x02 | EFI\_MTFTP6\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - DestroyChild() returns EFI\_SUCCESS with the 2nd validChildHandle. | 3. Call DestroyChild()with the 2nd valid ChildHandle, the return status should be EFI\_SUCCESS. |
| 5.26.4.1.5 | 0x0403eeee, 0x34d6, 0x47f4, 0x80, 0xcf, 0x28, 0x44, 0xa1, 0x7e, 0xfb, 0x7a | EFI\_MTFTP6\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - DestroyChild() returns EFI\_SUCCESS with the 1st validChildHandle. | 4. Call DestroyChild()with the 1st valid ChildHandle, the return status should be EFI\_SUCCESS. |

### DestroyChild ()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.4.2.1 | 0xc4bdecde, 0xc89f, 0x4402, 0x9e, 0x9b, 0x2e, 0xac, 0xdd, 0xd6, 0xf7, 0xa6 | EFI\_MTFTP6\_SERVICE\_BINDING\_PROTOCOL. DestroyChild() - DestroyChild() returns EFI\_INVALID\_PARAMETER with a NULL ChildHandle*.* | Call DestroyChild() with a NULL ChildHandle, the return status should be EFI\_INVALID\_PARAMETER. |

### GetModeData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.4.3.1 | 0x2d5eae25, 0x9fda, 0x47c9, 0x80, 0x14, 0xf3, 0x34, 0xf0, 0x1e, 0x67, 0x12 | EFI\_MTFTP6\_PROTOCOL.GetModeData() - GetModeData()returns EFI\_INVALID\_PARAMETER with **NULL** **ModeData** | Call GetModeData()with **NULL** **ModeData,** the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.3.2 | 0x2a17e0f5, 0x6eab, 0x4528, 0xb1, 0xef, 0x4e, 0x99, 0x77, 0xd0, 0xc4, 0xa7 | EFI\_MTFTP6\_PROTOCOL.GetModeData() - GetModeData()returns EFI\_SUCCESS with the valid parameters. | 5.26.4.3.2 to 5.26.4.3.3 belong to one case.  1. Call CreateChild() to create an MTFTP6 instance.  2. Call Configure() to initialize the MTFTP6 instance.  3. Call GetModeData() with the valid parameters, the return status should be EFI\_SUCCESS. |
| 5.26.4.3.3 | 0x51c6056a, 0x9582, 0x444a, 0xba, 0x84, 0x0d, 0xd5, 0xe4, 0x93, 0xb3, 0xa0 | Mtftp6ModeData.ConfigData should be the same as previous set ConfigData. | 4. Mtftp6ModeData.ConfigData should be the same as previous set ConfigData.  5. Call DestroyChild() to destroy the MTFTP6 instance. |

### Configure()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.4.4.1 | 0x2a946231, 0xa817, 0x45ed, 0x88, 0x59, 0x44, 0x42, 0xd5, 0x6d, 0x53, 0x45 | EFI\_MTFTP6\_PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER when StationIP is neither zero nor a configured IP address. | Call Configure() when StationIP is neither zero nor a configured IP address, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.4.2 | 0x02caf586, 0xff1c, 0x41e6, 0xb6, 0x5b, 0x6f, 0xd0, 0x22, 0xa4, 0x60, 0x14 | EFI\_MTFTP6\_PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER when ServerIp is an invalid unicast IPv6 address. | Call Configure() when ServerIp is an invalid unicast IPv6 address, such as ff02::1, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.4.3 | 0x9e06f1d5, 0xb888, 0x4976, 0x9c, 0x39, 0x6a, 0xcf, 0x26, 0x94, 0x2d, 0x76 | EFI\_MTFTP6\_PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER when ServerIp is an invalid unicast IPv6 address. | Call Configure() when ServerIp is an invalid unicast IPv6 address, such as ::, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.4.4 | 0xe5efe42a, 0x6539, 0x487d, 0x89, 0xe3, 0xb2, 0x88, 0x2a, 0xb1, 0xd7, 0xd4 | EFI\_MTFTP6\_PROTOCOL.Configure() - Configure()returns EFI\_ACCESS\_DENIED when StationIp and LocalPort have already been used. | Call Configure() when StationIp and LocalPort have already been used, the return status should be EFI\_ACCESS\_DENIED. |
| 5.26.4.4.5 | 0xcde4ae63, 0x74f6, 0x46fc, 0xa2, 0xae, 0x23, 0x2b, 0x39, 0x3a, 0x02, 0xd3 | EFI\_MTFTP6\_PROTOCOL.Configure() - Configure()returns EFI\_ACCESS\_DENIED when call Configure() again to update the Configure Data without call Configure() with NULL. | Call Configure() again to update the Configure Data without call Configure() with NULL, the return status should be EFI\_ACCESS\_DENIED. |
| 5.26.4.4.6 | 0x90337601, 0x85ca, 0x4152, 0x9f, 0x54, 0xdc, 0xac, 0x13, 0x87, 0x28, 0xdb | EFI\_MTFTP6\_PROTOCOL.Configure() - Configure()returns EFI\_SUCCESS with valid Mtftp6ConfigData. | 5.26.4.4.6 to 5.26.4.4.9 belong to one case.  1. Call CreateChild() to create an MTFTP6 instance.  2. Call Configure() with valid Mtftp6ConfigData, the return status should be EFI\_SUCCESS. |
| 5.26.4.4.7 | 0x85bceaa3, 0x377a, 0x4847, 0x93, 0x4c, 0xa7, 0xea, 0x95, 0x1a, 0x4e, 0x57 | EFI\_MTFTP6\_PROTOCOL.Configure() - Configure()returns EFI\_SUCCESS when Mtftp6ConfigData is NULL. | 3. Call Configure() when Mtftp6ConfigData is NULL, the return status should be EFI\_SUCCESS. |
| 5.26.4.4.8 | 0x62c85a93, 0x029d, 0x4bb2, 0xb5, 0x82, 0x25, 0x63, 0xae, 0x63, 0xba, 0xd7 | EFI\_MTFTP6\_PROTOCOL.Configure() - Configure()returns EFI\_SUCCESS with valid Mtftp6ConfigData in the second time. | 4. Call Configure() with the valid Mtftp6ConfigData in the second time, the return status should be EFI\_SUCCESS. |
| 5.26.4.4.9 | 0xef42aa6a, 0x1c66, 0x4768, 0x8c, 0x4f, 0xc1, 0x18, 0x8a, 0xdc, 0x69, 0xfb | Call GetModeData() with the valid parameters, the Mtftp6ModeData.ConfigData should be the same as previous set ConfigData. | 5. Call GetModeData() with the valid parameters, the Mtftp6ModeData.ConfigData should be the same as previous set ConfigData.  6. Call DestroyChild() to destroy the MTFTP6 instance. |

### GetInfo()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.4.5.1 | 0xed2fb03d, 0x8422, 0x46dc, 0xa4, 0xda, 0x31, 0xbe, 0x84, 0xa2, 0xf5, 0x0d | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo()returns EFI\_NOT\_STARTED when the instance hasn't been configured. | Call GetInfo() when the instance hasn't been configured, the return status should be EFI\_NOT\_STARTED. |
| 5.26.4.5.2 | 0xae921a1d, 0x1d87, 0x40a0, 0x90, 0x09, 0xe1, 0xc3, 0x30, 0x45, 0xd7, 0x7d | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo()returns EFI\_INVALID\_PARAMETER when filename is NULL. | Call GetInfo() when filename is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.5.3 | 0x99321cf6, 0x6591, 0x4b71, 0xbd, 0xbe, 0x5a, 0xcb, 0xbe, 0x97, 0x32, 0x51 | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo()returns EFI\_INVALID\_PARAMETER when OptionCount isn't zero and OptionList is NULL. | Call GetInfo() when OptionCount isn't zero and OptionList is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.5.4 | 0x807e6ac5, 0x5ff8, 0x4e9c, 0x9f, 0xb0, 0x21, 0x9d, 0x3c, 0xf6, 0xae, 0x1c | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo()returns EFI\_INVALID\_PARAMETER when one or more options in OptionList is wrong format. | Call GetInfo() when one or more options in OptionList is wrong format, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.5.5 | 0xdddb451a, 0x2d08, 0x45f2, 0xb4, 0x3c, 0x63, 0xde, 0xfc, 0xcc, 0x29, 0x42 | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo()returns EFI\_INVALID\_PARAMETER when PacketLength is NULL. | Call GetInfo() when PacketLength is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.5.6 | 0x1b915cd6, 0x34eb, 0x4a87, 0x8f, 0x18, 0x63, 0x25, 0x91, 0x1b, 0x80, 0x85 | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo()returns EFI\_INVALID\_PARAMETER when OverrideData.ServerIp is invalid unicast address. | Call GetInfo() when OverrideData.ServerIp is invalid unicast address, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.5.7 | 0x890ecac1, 0xd029, 0x4a8f, 0x99, 0x10, 0x57, 0x73, 0xa0, 0x70, 0xba, 0xff | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo()returns EFI\_INVALID\_PARAMETER when one or more options in OptionList is unsupported. | Call GetInfo() when one or more options in OptionList is unsupported, the return status should be EFI\_UNSUPPORTED. |
| 5.26.4.5.8 | 0xa807dd98, 0x8d94, 0x42cd, 0x9b, 0x38, 0x2c, 0x4d, 0xa1, 0x43, 0xc1, 0xbc | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo()returns EFI\_TFTP\_ERROR when a mtftp6 error packet received from the other side. | Call GetInfo() when a mtftp6 error packet received from the other side, the return status should be EFI\_TFTP\_ERROR. |
| 5.26.4.5.9 | 0x8ea63309, 0x2824, 0x4186, 0x93, 0x39, 0xd4, 0x10, 0x44, 0xef, 0xf2, 0x36 | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo()returns EFI\_TIMEOUT when there is no response from the other side. | Call GetInfo() when no response is sent from the other side, the return status should be EFI\_TIMEOUT. |
| 5.26.4.5.10 | 0x29b90725, 0x6662, 0x43f5, 0xa4, 0xe5, 0xb0, 0xb5, 0xfa, 0x26, 0x55, 0x38 | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo()returns EFI\_PORT\_UNREACHABLE when an ICMP port unreachable error packet was received. | Call GetInfo()when an ICMP port unreachable error packet was received, the return status should be EFI\_PORT\_UNREACHABLE. |
| 5.26.4.5.11 | 0x01b2ee0f, 0xb879, 0x4475, 0x9e, 0x58, 0x7d, 0xff, 0x51, 0x13, 0x88, 0x87 | EFI\_MTFTP6\_PROTOCOL. GetInfo () - GetInfo ()returns EFI\_SUCCESS with valid parameters. | 5.26.4.5.11 to 5.26.4.5.12 belong to one case.  1. Call CreateChild() to create an MTFTP6 instance.  2. Call Configure() with valid Mtftp6ConfigData, the return status should be EFI\_SUCCESS.  3. Call GetInfo() with valid parameters.  4. Host send MTFTP6 OACK packet.  5. Host receive the Ack for OACK  6. The return status of GetInfo() should be EFI\_SUCCESS. |
| 5.26.4.5.12 | 0x9ddd227a, 0x0734, 0x4d6b, 0xaf, 0xa9, 0xdb, 0xc1, 0xad, 0x10, 0xb7, 0xd3 | Call ParseOptions() to parse the Packet, the content of EFI\_MTFTP6\_OPTION should be right. | 7. Call ParseOptions() to parse the Packet, the content of EFI\_MTFTP6\_OPTION should be right. |
| 5.26.4.5.13 | 0x1257a949,  0xb84d, 0x43f6, 0x89,0x2b, 0x48,0x5f, 0x33,0x65, 0x82,0x12 | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo() returns  EFI\_PORT\_UNREACHABlE when an ICMP port  unreachable error packet  was received. | Call GetInfo() when an ICMP port  unreachable error packet was received,  the return status should be  EFI\_PORT\_UNREACHABLE. |
| 5.26.4.5.14 | 0xd3688340, 0x7b29, 0x46cb, 0x98,0x05, 0x76,0xf0, 0xab,0xef, 0x78,0xc0 | EFI\_MTFTP6\_PROTOCOL  .GetInfo() - GetInfo() returns  EFI\_NETWORK\_UNREACHABLE when an ICMP net  unreachable error packet  was received. | Call GetInfo() when an ICMP net  unreachable error packet was received,  the return status should be  EFI\_NETWORK\_UNREACHABLE. |
| 5.26.4.5.15 | 0x8cffd8f0,0xf8e7, 0x4e6c, 0x8e,0x2f, 0xbe,0xf5, 0xff,0xd8, 0xd4,0x8c | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo() returns  EFI\_HOST\_UNREACHABLE when an ICMP host  unreachable error packet  was received. | Call GetInfo() when an ICMP host  unreachable error packet was received,  the return status should be  EFI\_HOST\_UNREACHABLE. |

### ParseOptions()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.4.6.1 | 0x165bba38, 0x2cc8, 0x4c86, 0xb5, 0x9a, 0x82, 0xd5, 0x33, 0xe0, 0x3d, 0x12 | EFI\_MTFTP6\_PROTOCOL.ParseOptions() - ParseOptions()returns EFI\_INVALID\_PARAMETER when PacketLen is zero. | Call ParseOptions() when PacketLen is zero, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.6.2 | 0x46feb505, 0x82fd, 0x4d84, 0x98, 0x9f, 0x2a, 0x24, 0x70, 0xff, 0xf9, 0x1f | EFI\_MTFTP6\_PROTOCOL. ParseOptions() - ParseOptions()returns EFI\_INVALID\_PARAMETER when Packet is NULL. | Call ParseOptions() when Packet is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.6.3 | 0x2c5276ba, 0x0fed, 0x474f, 0x91, 0x79, 0x9d, 0xa1, 0xdb, 0x8e, 0x32, 0x19 | EFI\_MTFTP6\_PROTOCOL. ParseOptions() - ParseOptions()returns EFI\_INVALID\_PARAMETER when Packet isn't a valid Mtftp6 packet. | Call ParseOptions() when Packet isn't a valid Mtftp6 packet, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.6.4 | 0x3bd37c27, 0xeaea, 0x474c, 0x92, 0xf7, 0xd4, 0x90, 0x68, 0x50, 0xb5, 0x54 | EFI\_MTFTP6\_PROTOCOL. ParseOptions() - ParseOptions()returns EFI\_INVALID\_PARAMETER when when OptionCount is NULL. | Call ParseOptions() when OptionCount is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.6.5 | 0xd5918b06, 0x88cd, 0x4321, 0x90, 0x18, 0x3e, 0x3e, 0x1a, 0xd2, 0xcd, 0xa1 | EFI\_MTFTP6\_PROTOCOL. ParseOptions() - ParseOptions()returns EFI\_NOT\_FOUND when no Options is found. | Call ParseOptions() when no Options is found, the return status should be EFI\_NOT\_FOUND. |
| 5.26.4.6.6 | 0xad87d495, 0x9738, 0x4c86, 0x97, 0x2c, 0x63, 0xdb, 0xcd, 0x2b, 0xda, 0x84 | EFI\_MTFTP6\_PROTOCOL. ParseOptions() - ParseOptions()returns EFI\_PROTOCOL\_ERROR when one or more of the option fields are not valid. | Call ParseOptions() when one or more of the option fields are not valid, the return status should be EFI\_PROTOCOL\_ERROR. |

### ReadFile()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.4.7.1 | 0x33346d27, 0x213b, 0x4137, 0xa0, 0x4e, 0xff, 0x79, 0xc3, 0x40, 0x82, 0x2a | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_NOT\_STARTED when the instance hasn’t been configured. | Call ReadFile()when the instance hasn’t been configured. The return status should be EFI\_NOT\_STARTED. |
| 5.26.4.7.2 | 0xfa4a5e44, 0x3823, 0x4273, 0xa8, 0x86, 0x7d, 0x95, 0xb4, 0xd9, 0x0d, 0xa1 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_INVALID\_PARAMETER when Token is NULL. | Call ReadFile() when Token is NULL. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.7.3 | 0x2e09fd86, 0xfe91, 0x4490, 0x9f, 0x33, 0xa9, 0xdf, 0x38, 0x65, 0xf0, 0xdf | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_INVALID\_PARAMETER when Token.Filename is NULL. | Call ReadFile() when Token.Filename is NULL. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.7.4 | 0x197e3225, 0xc6ba, 0x43ee, 0x8d, 0xf7, 0x31, 0x31, 0xbd, 0x71, 0x5e, 0x2e | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_INVALID\_PARAMETER when OptionCount isn't zero and OptionList is NULL. | Call ReadFile() when OptionCount isn't zero and OptionList is NULL. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.7.5 | 0x983411c5, 0x040b, 0x4995, 0xbb, 0x0e, 0x80, 0xb8, 0x69, 0x3b, 0x6b, 0x9e | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_INVALID\_PARAMETER when one or more options in Token.OptionList is wrong format. | Call ReadFile() when one or more options in Token.OptionList is wrong format. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.7.6 | 0x7fff6983, 0x39e5, 0x421f, 0x93, 0xb8, 0x3a, 0x16, 0x4d, 0x3a, 0x95, 0x34 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_INVALID\_PARAMETER when one or more options in Token.Buffer and Token.CheckPacket are both NULL. | Call ReadFile() when one or more options in Token.Buffer and Token.CheckPacket are both NULL. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.7.7 | 0xbdb9aaa3, 0x4efa, 0x41dc, 0x91, 0x22, 0xf9, 0x15, 0x04, 0x4f, 0x34, 0xac | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_INVALID\_PARAMETER when one or more options in Token.OverrideData.ServerIp is not valid unicast IPv6 address. | Call ReadFile() when one or more options in Token.OverrideData.ServerIp is not valid unicast IPv6 address. The return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.7.8 | 0xf410b1c3, 0x5e50, 0x4389, 0x99, 0xac, 0x5f, 0x9f, 0xe7, 0xed, 0x47, 0x9d | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_INVALID\_PARAMETER when one or more options in Token.OptionList is not supported. | Call ReadFile() when one or more options in Token.OptionList is not supported. The return status should be EFI\_UNSUPPORTED. |
| 5.26.4.7.9 | 0xb5b845cf, 0x1ac2, 0x4ba6, 0x88, 0x13, 0x35, 0xdc, 0xe6, 0x07, 0xec, 0x82 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_BUFFER\_TOO\_SMALL when Token.BufferSize isn't large enough to hold the download data in download process. | Call ReadFile() when Token.BufferSize isn't large enough to hold the download data in download process. The return status should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.26.4.7.10 | 0x79f11d98, 0x4a0c, 0x4c2a, 0x8f, 0x48, 0x58, 0xa6, 0x04, 0x6c, 0x11, 0x94 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_ABORTED when current operation is aborted by user. | Call ReadFile() when current operation is aborted by user. The return status should be EFI\_ABORTED. |
| 5.26.4.7.11 | 0x99d1d01e, 0x23f4, 0x4877, 0x98, 0xe9, 0x6e, 0xa3, 0xb9, 0x98, 0x8e, 0x9d | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_TFTP\_ERROR when a mtftp6 error packet received. | Call ReadFile() when a mtftp6 error packet was received. The return status should be EFI\_TFTP\_ERROR. |
| 5.26.4.7.12 | 0x2e222488, 0xcab8, 0x40d5, 0xa6, 0x71, 0xac, 0xa6, 0x6c, 0x76, 0x58, 0x59 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_PORT\_UNREACHABLE when a icmp6 port unreachable error packet was received. | Call ReadFile() when a icmp6 port unreachable error packet was received. The return status should be EFI\_PORT\_UNREACHABLE. |
| 5.26.4.7.13 | 0x36a6ebe2,0xdb79, 0x423f, 0xad,0x53, 0x9b,0xf1, 0x7d,0x1b, 0x4c,0x20 | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo() returns  EFI\_NETWORK\_UNREACHABLEwhen an ICMP net unreachable error packet was received. | Call GetInfo() when an ICMP net  unreachable error packet was received. The return status should be  EFI\_NETWORK\_UNREACHABLE. |
| 5.26.4.7.14 | 0x3215f20a, 0xec4f, 0x4666, 0x8d,0x6b, 0x e7,0x09, 0x21,0x65, 0x7a,0xa2 | EFI\_MTFTP6\_PROTOCOL.GetInfo() - GetInfo() returns  EFI\_HOST\_UNREACHABLE when an ICMP host  unreachable error packet was received. | Call GetInfo() when an ICMP host unreachable error packet was received. The return status should be  EFI\_HOST\_UNREACHABLE. |
| 5.26.4.7.15 | 0x0d5a4c2a, 0xc87e, 0x41e4, 0xa8, 0x6b, 0xce, 0x62, 0x30, 0x7c, 0x84, 0x06 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_TIMEOUT when no response was received. | Call ReadFile() when no response was received. The return status should be EFI\_TIMEOUT. |
| 5.26.4.7.16 | 0xa29fb61f, 0x4f6c, 0x4e15, 0xaf, 0x96, 0xb7, 0x0c, 0xf2, 0x1c, 0xd7, 0x71 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_SUCCESS with valid parameters. | 5.26.4.7.16 to 5.26.4.7.17 belong to one case.  1. Call ReadFile() with valid parameters. The return status should be EFI\_SUCCESS. |
| 5.26.4.7.17 | 0xea84cd69, 0x5550, 0x44a0, 0xbb, 0xe7, 0x0f, 0xc1, 0x5d, 0x08, 0xb8, 0x35 | The Token.Status should be EFI\_SUCCESS. | 2. The Token.Status should be EFI\_SUCCESS. |
| 5.26.4.7.18 | 0x789c0d97, 0x68d8, 0x4a72, 0xa8, 0x7f, 0x66, 0x37, 0xcb, 0x6b, 0xb8, 0xe0 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_SUCCESS with valid parameters. | 5.26.4.7.18 to 5.26.4.7.20 belong to one case.  1. Call ReadFile() with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.26.4.7.19 | 0x9a991ff0, 0x84af, 0x4290, 0x85, 0x3b, 0x02, 0xe0, 0xb7, 0xe4, 0xe0, 0x28 | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.7.20 | 0xd90350a1, 0x7e65, 0x435f, 0xa3, 0x8f, 0x24, 0x27, 0xf1, 0x08, 0x8f, 0x5f | The Token.Status should be EFI\_SUCCESS. | 3. The Token.Status should be EFI\_SUCCESS. |
| 5.26.4.7.21 | 0xdf7f3d8e, 0x492e, 0x46ef, 0xb9, 0x8e, 0x26, 0x06, 0x9e, 0x85, 0x7a, 0x73 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and Token.BufferSize is not large enough. | 5.26.4.7.21 to 5.26.4.7.23 belong to one case.  1. Call ReadFile() with valid parameters, Token.Event is not NULL and Token.BufferSize is not large enough, the return status should be EFI\_SUCCESS. |
| 5.26.4.7.22 | 0x5ff92824, 0x75a9, 0x4e39, 0xa3, 0xa0, 0xc6, 0x2d, 0x42, 0x09, 0x48, 0x78 | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.7.23 | 0x5bd23489, 0xc9df, 0x4ce5, 0x84, 0xe9, 0x51, 0xa1, 0x88, 0xe2, 0x96, 0x3f | The Token.Status should be EFI\_BUFFER\_TOO\_SMALL. | 3. The Token.Status should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.26.4.7.24 | 0x4f23a070, 0xd01c, 0x441c, 0x88, 0x36, 0x26, 0xc4, 0x00, 0x05, 0xda, 0x0b | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and current operation is aborted by user. | 5.26.4.7.24 to 5.26.4.7.26 belong to one case.  1. Call ReadFile() with valid parameters, Token.Event is not NULL and current operation is aborted by user, the return status should be EFI\_SUCCESS. |
| 5.26.4.7.25 | 0xb22cb194, 0xd7db, 0x4141, 0x87, 0x8d, 0xab, 0xb7, 0x76, 0x9a, 0x12, 0xf6 | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.7.26 | 0x8a946d5c, 0xa820, 0x47c1, 0x83, 0xdf, 0x4b, 0x73, 0x9f, 0x52, 0x89, 0x53 | The Token.Status should be EFI\_ABORTED. | 3. The Token.Status should be EFI\_ABORTED. |
| 5.26.4.7.27 | 0x11b9ec6c, 0xff52, 0x4279, 0x9a, 0x07, 0x64, 0x1b, 0xcb, 0xe5, 0x37, 0x73 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and a mtftp6 error packet was received. | 5.26.4.7.27 to 5.26.4.7.29 belong to one case.  1. Call ReadFile() with valid parameters, Token.Event is not NULL and a mtftp6 error packet was received, the return status should be EFI\_SUCCESS. |
| 5.26.4.7.28 | 0x70e67e7f, 0x0a67, 0x4402, 0xa2, 0x63, 0x9a, 0xe9, 0x75, 0xcb, 0x7c, 0x71 | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.7.29 | 0x35b45761, 0x9657, 0x4211, 0xb4, 0xfb, 0xed, 0x68, 0xe1, 0x98, 0xf4, 0x02 | The Token.Status should be EFI\_TFTP\_ERROR. | 3. The Token.Status should be EFI\_TFTP\_ERROR. |
| 5.26.4.7.30 | 0x6aa2ecf0, 0xb01e, 0x4a8e, 0xb3, 0xdc, 0xd0, 0x54, 0xce, 0xb6, 0xa0, 0x83 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and a icmp6 error packet was received. | 5.26.4.7.30 to 5.26.4.7.32 belong to one case.  1. Call ReadFile() with valid parameters, Token.Event is not NULL and a icmp6 error packet was received, the return status should be EFI\_SUCCESS. |
| 5.26.4.7.31 | 0x6794533c, 0xf4f6, 0x4972, 0x8c, 0xf4, 0x3a, 0xc6, 0x20, 0x20, 0x19, 0x4e | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.7.32 | 0xda706911, 0x98fd, 0x49a3, 0xa6, 0x55, 0x74, 0x75, 0xb5, 0x5a, 0x0d, 0x4f | The Token.Status should be EFI\_PORT\_UNREACHABLE. | 3. The Token.Status should be EFI\_PORT\_UNREACHABLE. |
| 5.26.4.7.33 | 0xc80090b9, 0x0876, 0x4959, 0xbd, 0x80, 0x6c, 0x41, 0x40, 0x92, 0xac, 0x48 | EFI\_MTFTP6\_PROTOCOL.ReadFile() - ReadFile()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and a icmp6 error packet was received. | 5.26.4.7.33 to 5.26.4.7.35 belong to one case.  1. Call ReadFile() with valid parameters, Token.Event is not NULL and no response was received, the return status should be EFI\_SUCCESS. |
| 5.26.4.7.34 | 0xa8ce4013, 0x648f, 0x46d5, 0xa4, 0x89, 0xd3, 0x33, 0x1a, 0xee, 0x1d, 0x57 | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.7.35 | 0xbb028e8c, 0xf45a, 0x4052, 0x83, 0x3b, 0x45, 0x81, 0xec, 0x1b, 0x62, 0x0f | The Token.Status should be EFI\_TIMEOUT. | 3. The Token.Status should be EFI\_TIMEOUT. |

### WriteFile()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.4.8.1 | 0x3123cc65, 0x7cea, 0x4b5e, 0x92, 0xd9, 0x7d, 0x8c, 0xe4, 0x95, 0x3f, 0x4f | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_NOT\_STARTED when the instance hasn’t been configured. | Call WriteFile()when the instance hasn’t been configured, the return status should be EFI\_NOT\_STARTED. |
| 5.26.4.8.2 | 0x6738f74e, 0x3f6f, 0x48db, 0xaa, 0x1b, 0x8d, 0x38, 0xf8, 0x19, 0x4a, 0x61 | EFI\_MTFTP6\_PROTOCOL.WriteFile() - Writeile()returns EFI\_INVALID\_PARAMETER when Token is NULL. | Call WriteFile() when Token is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.8.3 | 0x637d7d38, 0x102d, 0x4382, 0x9f, 0x95, 0x0f, 0xc3, 0x97, 0x84, 0x29, 0xcf | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_INVALID\_PARAMETER when Token.Filename is NULL. | Call WriteFile() when Token.Filename is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.8.4 | 0xf39cdb05, 0xd139, 0x4dd7, 0x8c, 0x15, 0x41, 0x2f, 0x8b, 0xde, 0x04, 0xb3 | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_INVALID\_PARAMETER when Token.OptionCount isn't zero and Token.OptionList is NULL. | Call WriteFile() when Token.OptionCount isn't zero and Token.OptionList is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.8.5 | 0x9d3fcbac, 0xbc54, 0x46d9, 0x85, 0x41, 0x64, 0xe3, 0xaa, 0x41, 0x25, 0x58 | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_INVALID\_PARAMETER when one or more options in Token.OptionList is wrong format. | Call WriteFile() when one or more options in Token.OptionList is wrong format, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.8.6 | 0x181a05aa, 0xcd53, 0x4ba1, 0xb8, 0xf3, 0x98, 0x09, 0x9b, 0xd4, 0xa7, 0xe1 | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_INVALID\_PARAMETER when one or more options in Token.Buffer and Token.PacketNeeded are both NULL. | Call WriteFile() when one or more options in Token.Buffer and Token.PacketNeeded are both NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.8.7 | 0xb820e6cb, 0x5290, 0x4748, 0x94, 0x66, 0x8d, 0xc1, 0x99, 0x2e, 0x12, 0xc1 | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_INVALID\_PARAMETER when one or more options in Token.OverrideData.ServerIp is not valid unicast IPv6 address. | Call WriteFile() when one or more options in Token.OverrideData.ServerIp is not valid unicast IPv6 address, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.8.8 | 0x3854186d, 0x550e, 0x4006, 0xbe, 0xff, 0x1c, 0x52, 0x5b, 0xa4, 0x3e, 0xcf | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_INVALID\_PARAMETER when one or more options in Token.OptionList is not supported. | Call WriteFile() when one or more options in Token.OptionList is not supported, the return status should be EFI\_UNSUPPORTED. |
| 5.26.4.8.9 | 0x54ae8e18, 0xd428, 0x48c1, 0xad, 0x32, 0xb5, 0x51, 0xda, 0x1e, 0x90, 0x7f | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_SUCCESS with valid parameters. | 5.26.4.8.9 to 5.26.4.8.10 belong to one case.  1. Call WriteFile() with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.26.4.8.10 | 0xc2e70601, 0xb8d5, 0x4aa6, 0xbb, 0x5c, 0x6e, 0xda, 0x2a, 0xd0, 0xa6, 0x6a | The Token.Status should be EFI\_SUCCESS. | 2. The Token.Status should be EFI\_SUCCESS. |
| 5.26.4.8.11 | 0x9e572894, 0x38da, 0x4039, 0x96, 0xc9, 0xaa, 0xfe, 0xa6, 0x48, 0x60, 0x74 | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_SUCCESS with valid parameters. | 5.26.4.8.11 to 5.26.4.8.13 belong to one case.  1. Call WriteFile() with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.26.4.8.12 | 0xeba41d25, 0x03d7, 0x41d7, 0xa0, 0x58, 0xa8, 0x90, 0xad, 0x68, 0xa7, 0x0b | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.8.13 | 0x39afac2f, 0xb620, 0x45e9, 0x8d, 0x82, 0x7a, 0xec, 0x36, 0x9d, 0x19, 0xfb | The Token.Status should be EFI\_SUCCESS. | 3. The Token.Status should be EFI\_SUCCESS. |
| 5.26.4.8.14 | 0x98410f1a, 0x6f26, 0x45f4, 0x8c, 0x5d, 0x9e, 0x11, 0x19, 0x53, 0xd3, 0xf8 | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and current operation is aborted by user. | 5.26.4.8.14 to 5.26.4.8.16 belong to one case.  1. Call WriteFile() with valid parameters, Token.Event is not NULL and current operation is aborted by user, the return status should be EFI\_SUCCESS. |
| 5.26.4.8.15 | 0xbb6d10b9, 0x4466, 0x4f97, 0x9a, 0x3f, 0xa9, 0xa0, 0x7a, 0x49, 0x88, 0x5d | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.8.16 | 0x87eff284, 0x80a4, 0x48ae, 0xa1, 0x87, 0x54, 0xa5, 0xf6, 0xd8, 0xcc, 0xcf | The Token.Status should be EFI\_ABORTED. | 3. The Token.Status should be EFI\_ABORTED. |
| 5.26.4.8.17 | 0x84cf72a7, 0x0d57, 0x4519, 0x8d, 0x94, 0x5a, 0x63, 0xeb, 0xff, 0x8c, 0x73 | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and a mtftp6 error packet was received. | 5.26.4.8.17 to 5.26.4.8.19 belong to one case.  1. Call WriteFile() with valid parameters, Token.Event is not NULL and a mtftp6 error packet was received, the return status should be EFI\_SUCCESS. |
| 5.26.4.8.18 | 0x35003a00, 0x715d, 0x4f05, 0x80, 0x0d, 0xec, 0xcc, 0x92, 0xed, 0x19, 0x51 | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.8.19 | 0x11ce4fd8, 0x7d75, 0x49ec, 0x8a, 0x2b, 0x98, 0x57, 0xd9, 0xce, 0x5d, 0x66 | The Token.Status should be EFI\_TFTP\_ERROR. | 3. The Token.Status should be EFI\_TFTP\_ERROR. |
| 5.26.4.8.20 | 0xb1d3d500, 0x4afa, 0x465f, 0x8e, 0xac, 0x79, 0x0d, 0xf9, 0xef, 0x3f, 0xea | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and a icmp6 error packet was received. | 5.26.4.8.20 to 5.26.4.8.22 belong to one case.  1. Call WriteFile() with valid parameters, Token.Event is not NULL and a icmp6 error packet was received, the return status should be EFI\_SUCCESS. |
| 5.26.4.8.21 | 0x42093ba6, 0x54ce, 0x408c, 0x82, 0x0f, 0xce, 0xba, 0xf4, 0x56, 0x58, 0xe9 | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.8.22 | 0xc7ba7541, 0x6d62, 0x4143, 0x8e, 0x18, 0x6e, 0xbb, 0xfe, 0x2c, 0x42, 0xd8 | The Token.Status should be EFI\_PORT\_UNREACHABLE. | 3. The Token.Status should be EFI\_PORT\_UNREACHABLE. |
| 5.26.4.8.23 | 0x08fffd13, 0x7cfb, 0x49ec, 0x8b, 0x02, 0x2a, 0x45, 0xb9, 0x78, 0x1d, 0xcd | EFI\_MTFTP6\_PROTOCOL.WriteFile() - WriteFile()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and a icmp6 error packet was received. | 5.26.4.8.23 to 5.26.4.8.25 belong to one case.  1. Call WriteFile() with valid parameters, Token.Event is not NULL and no response was received, the return status should be EFI\_SUCCESS. |
| 5.26.4.8.24 | 0xeb45268b, 0xa856, 0x4ab8, 0xb4, 0xba, 0x38, 0xf9, 0x35, 0xd7, 0x99, 0x2c | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.8.25 | 0xa93f3a80, 0xeb22, 0x4ad8, 0xb5, 0x7c, 0xf5, 0x39, 0x7d, 0xe5, 0x39, 0x33 | The Token.Status should be EFI\_TIMEOUT. | 3. The Token.Status should be EFI\_TIMEOUT. |

### ReadDirectory()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.4.9.1 | 0x1947060b, 0x44a2, 0x4e22, 0x9b, 0xe9, 0x20, 0xf4, 0xa2, 0x9e, 0xb4, 0x2e | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_NOT\_STARTED when the instance hasn’t been configured. | Call ReadDirectory()when the instance hasn’t been configured, the return status should be EFI\_NOT\_STARTED. |
| 5.26.4.9.2 | 0x9d2a2470, 0x98de, 0x425a, 0xbb, 0x3f, 0xab, 0x01, 0x84, 0x13, 0xe6, 0x9c | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_INVALID\_PARAMETER when Token is NULL. | Call ReadDirectory() when Token is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.9.3 | 0x6288a172, 0xbc68, 0x49c1, 0xa8, 0x85, 0x56, 0x5a, 0x9e, 0xf9, 0x5c, 0x46 | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_INVALID\_PARAMETER when Token.Filename is NULL. | Call ReadDirectory() when Token.Filename is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.9.4 | 0xe95a938b, 0x7d16, 0x4b40, 0xbd, 0x23, 0x81, 0x7b, 0xf6, 0xf1, 0xff, 0xb4 | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_INVALID\_PARAMETER when Token.OptionCount isn't zero and Token.OptionList is NULL. | Call ReadDirectory() when Token.OptionCount isn't zero and Token.OptionList is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.9.5 | 0x4ec4e899, 0x9f53, 0x461a, 0xb3, 0xe4, 0x21, 0xe4, 0x1e, 0x66, 0x56, 0x21 | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_INVALID\_PARAMETER when one or more options in Token.OptionList is wrong format. | Call ReadDirectory() when one or more options in Token.OptionList is wrong format, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.9.6 | 0xbb6ac976, 0xbe0b, 0x4329, 0x98, 0xe3, 0x1f, 0x2d, 0xc8, 0x63, 0x8f, 0x9a | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_INVALID\_PARAMETER when one or more options in Token.Buffer and Token.CheckPacket are both NULL. | Call ReadDirectory() when one or more options in Token.Buffer and Token.CheckPacket are both NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.9.7 | 0xa41fa6b3, 0xc128, 0x451f, 0x82, 0xf4, 0xf8, 0x92, 0x8e, 0xa8, 0x61, 0xfd | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_INVALID\_PARAMETER when one or more options in Token.OverrideData.ServerIp is not valid unicast IPv6 address. | Call ReadDirectory() when one or more options in Token.OverrideData.ServerIp is not valid unicast IPv6 address, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.26.4.9.8 | 0xed4d5f77, 0x7856, 0x4c4a, 0x9a, 0x6b, 0x19, 0x66, 0xc7, 0xf6, 0x80, 0xfb | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_INVALID\_PARAMETER when one or more options in Token.OptionList is not supported. | Call ReadDirectory() when one or more options in Token.OptionList is not supported, the return status should be EFI\_UNSUPPORTED. |
| 5.26.4.9.9 | 0xf6439066, 0xb46e, 0x484e, 0x9a, 0x99, 0xfe, 0xa9, 0xaf, 0x72, 0xf9, 0xce | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_ABORTED when current operation is aborted by user. | Call ReadDirectory() when current operation is aborted by user, the return status should be EFI\_ABORTED. |
| 5.26.4.9.10 | 0xd8f2b214, 0xbfc2, 0x4344, 0x85, 0x13, 0xfc, 0x07, 0x04, 0x3f, 0x63, 0x6f | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_TFTP\_ERROR when a mtftp6 error packet was received. | Call ReadDirectory() when a mtftp6 error packet was received, the return status should be EFI\_TFTP\_ERROR. |
| 5.26.4.9.11 | 0x74cbaed3, 0x1521, 0x4677, 0x83, 0xb0, 0xca, 0xac, 0x84, 0x92, 0x27, 0x68 | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_PORT\_UNREACHABLE when a icmp6 port unreachable error packet was received. | Call ReadDirectory() when a icmp6 port unreachable error packet was received, the return status should be EFI\_PORT\_UNREACHABLE. |
| 5.26.4.9.12 | 0x71038101, 0x41ba, 0x416e, 0xa7, 0xb3, 0xd6, 0x12, 0x27, 0x8f, 0xb9, 0xe5 | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_TIMEOUT when no response was received. | Call ReadDirectory() when no response was received, the return status should be EFI\_TIMEOUT. |
| 5.26.4.9.13 | 0x177b35e7, 0x8e93, 0x48c4, 0x8f, 0x19, 0x7e, 0xe7, 0xf9, 0x9d, 0x65, 0x60 | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_SUCCESS with valid parameters. | 5.26.4.9.13 to 5.26.4.9.14 belong to one case.  1. Call ReadDirectory() with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.26.4.9.14 | 0x63dd12dd, 0x62e4, 0x40d3, 0x88, 0x30, 0xac, 0x2e, 0x61, 0xd2, 0x08, 0xb8 | The Token.Status should be EFI\_SUCCESS. | 2. The Token.Status should be EFI\_SUCCESS. |
| 5.26.4.9.15 | 0xbc2d0220, 0xa92b, 0x4281, 0x82, 0xf1, 0x45, 0xf1, 0x98, 0x10, 0x33, 0x05 | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_SUCCESS with valid parameters. | 5.26.4.9.15 to 5.26.4.9.17 belong to one case.  1. Call ReadDirectory() with valid parameters, the return status should be EFI\_SUCCESS. |
| 5.26.4.9.16 | 0xc495566f, 0x31a5, 0x47d3, 0x97, 0x50, 0xdf, 0xe9, 0xed, 0xfd, 0xe7, 0xfc | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.9.17 | 0xb734f8cc, 0x91c2, 0x4ce8, 0xa3, 0x11, 0x70, 0x70, 0xb8, 0x27, 0x6b, 0x03 | The Token.Status should be EFI\_SUCCESS. | 3. The Token.Status should be EFI\_SUCCESS. |
| 5.26.4.9.18 | 0xce7b5436, 0x3e80, 0x46d1, 0xbc, 0x6a, 0x04, 0x05, 0x91, 0xd8, 0xf2, 0x00 | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and current operation is aborted by user. | 5.26.4.9.18 to 5.26.4.9.20 belong to one case.  1. Call ReadDirectory() with valid parameters, Token.Event is not NULL and current operation is aborted by user, the return status should be EFI\_SUCCESS. |
| 5.26.4.9.19 | 0x06cc2106, 0x12e0, 0x4b26, 0x82, 0x84, 0xb6, 0x45, 0x26, 0x56, 0xb7, 0x67 | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.9.20 | 0x2f8d4207, 0xcaa0, 0x4fe8, 0xae, 0x18, 0x09, 0xfc, 0xd8, 0x1f, 0x60, 0xb1 | The Token.Status should be EFI\_ABORTED. | 3. The Token.Status should be EFI\_ABORTED. |
| 5.26.4.9.21 | 0xf19e2441, 0x2e9d, 0x4754, 0xaa, 0x1c, 0x9d, 0xff, 0x5d, 0xac, 0x7b, 0xfb | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and a mtftp6 error packet was received. | 5.26.4.9.21 to 5.26.4.9.23 belong to one case.  1. Call ReadDirectory() with valid parameters, Token.Event is not NULL and a mtftp6 error packet was received, the return status should be EFI\_SUCCESS. |
| 5.26.4.9.22 | 0x6d29ada4, 0xb541, 0x4ed1, 0x9c, 0x54, 0x42, 0x97, 0xc6, 0x99, 0xa9, 0x2f | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.9.23 | 0x7ee4d2f0, 0x43a5, 0x4730, 0x88, 0x00, 0x7f, 0x72, 0xcd, 0x76, 0x6e, 0xb5 | The Token.Status should be EFI\_TFTP\_ERROR. | 3. The Token.Status should be EFI\_TFTP\_ERROR. |
| 5.26.4.9.24 | 0x24b159a5, 0x0d03, 0x408d, 0x84, 0x3d, 0x5b, 0xfd, 0xb8, 0xf7, 0x10, 0xc4 | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and a icmp6 error packet was received. | 5.26.4.9.24 to 5.26.4.9.26 belong to one case.  1. Call ReadDirectory() with valid parameters, Token.Event is not NULL and a icmp6 error packet was received, the return status should be EFI\_SUCCESS. |
| 5.26.4.9.25 | 0x450a81e4, 0xf424, 0x4399, 0x9a, 0xb9, 0xef, 0x2e, 0xa9, 0x4a, 0x7e, 0x46 | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.9.26 | 0xc2b9002f, 0x9183, 0x4a4b, 0xba, 0x26, 0x3f, 0x72, 0x93, 0xed, 0xf0, 0xa4 | The Token.Status should be EFI\_PORT\_UNREACHABLE. | 3. The Token.Status should be EFI\_PORT\_UNREACHABLE. |
| 5.26.4.9.27 | 0x1aef9df8, 0xcf77, 0x449a, 0xa1, 0x6b, 0x0b, 0x8d, 0x8e, 0x4a, 0xf9, 0x06 | EFI\_MTFTP6\_PROTOCOL. ReadDirectory() - ReadDirectory()returns EFI\_SUCCESS with valid parameters, Token.Event is not NULL and a icmp6 error packet was received. | 5.26.4.9.27 to 5.26.4.9.29 belong to one case.  1. Call ReadDirectory() with valid parameters, Token.Event is not NULL and no response was received, the return status should be EFI\_SUCCESS. |
| 5.26.4.9.28 | 0xb5398e7d, 0x02cb, 0x4fd2, 0xa9, 0xdd, 0xd9, 0x75, 0x8f, 0x32, 0xd9, 0x26 | The Token.Event should be signaled. | 2. The Token.Event should be signaled. |
| 5.26.4.9.29 | 0x279e1bfa, 0x5db9, 0x44ba, 0xbb, 0x3c, 0xe4, 0x3b, 0xf1, 0xeb, 0xa8, 0x70 | The Token.Status should be EFI\_TIMEOUT. | 3. The Token.Status should be EFI\_TIMEOUT. |

### Poll()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.26.4.10.1 | 0xdfb24a28, 0xc61c, 0x4ec0, 0x9e, 0x78, 0x3a, 0xcf, 0x85, 0x9f, 0xa8, 0x0e | EFI\_MTFTP6\_PROTOCOL. Poll() – Poll()returns EFI\_NOT\_STARTED when the instance hasn’t been configured. | Call Poll()when the instance hasn’t been configured, the return status should be EFI\_NOT\_STARTED. |