# Network Protocols ARP and DHCP

## EFI\_ARP\_PROTOCOL Test

Reference Document:

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

### Add()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.1.1.1 | 0xf6fa3bd8, 0xd8d0, 0x4c54, 0x88, 0xc2, 0x1f, 0xcf, 0x27, 0x62, 0xc5, 0xd4 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_INVALID\_PARAMETER with both the *DenyFlag* and *TargetAddress* value of NULL. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() with both the *DenyFlag* and *TargetAddress* value of NULL.  The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.2 | 0x6404caf6, 0x9020, 0x4272, 0xa2, 0x79, 0x6f, 0x53, 0x8d, 0x42, 0x5c, 0x35 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_INVALID\_PARAMETER with a *DenyFlag* value of FALSE and the TargetHwAddress */*TargetSwAddress value of NULL. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() with a *DenyFlag* value of FALSE and the TargetHwAddress */*TargetSwAddress value of NULL.  The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.3 | 0x138858cd, 0x40fe, 0x4b05, 0xb4, 0x8c, 0xb5, 0x9f, 0xf2, 0xfd, 0xee, 0x5e | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_INVALID\_PARAMETER with a *DenyFlag* value of FALSE and a TargetHwAddress value of NULL. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() with a DenyFlag value of FALSE and a TargetHwAddress value of NULL.  The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.4 | 0x48a946f4, 0x8ff7, 0x4b50, 0xa1, 0xb2, 0xc6, 0x82, 0xcd, 0xa5, 0x78, 0x62 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_INVALID\_PARAMETER with a *DenyFlag* value of FALSE and a TargetSwAddress value of NULL. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() with a *DenyFlag* value of FALSE and a TargetSwAddress value of NULL.  The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.5 | 0x32deb7c7, 0x9e67, 0x459f, 0xbf, 0x4c, 0xbc, 0x80, 0x33, 0x31, 0x36, 0x05 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_INVALID\_PARAMETER with a *DenyFlag* value of TRUE and both TargetHwAddress and TargetSwAddress value of NULL. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() when *DenyFlag* is TRUE and both TargetHwAddress and TargetSwAddress are not NULL.  The return status must be EFI\_INVALID\_PARAMETER.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.6 | 0x87d47f39, 0x8d82, 0x40c4, 0xb9, 0x36, 0x2c, 0xf5, 0x8b, 0xa2, 0xd9, 0x32 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_ACCESS\_DENIED when the ARP cache entry of the same TargetSwAddress already exists and *Overwrite* is FALSE. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() with a *DenyFlag* value of FALSE and with valid TargetSwAddress / TargetHwAddress values.  4. Call EFI\_ARP\_PROTOCOL.Add() with a *DenyFlag* value of TRUE and with the same TargetSwAddress as the one used in the last call while *Overwrite* is FALSE.  The return status must be EFI\_ACCESS\_DENIED.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.7 | 0xecc2942f, 0xd23e, 0x421e, 0x8a, 0x31, 0x3c, 0xe2, 0xdf, 0xee, 0x82, 0xcb | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_ACCESS\_DENIED when the ARP cache entry of the same TargetHwAddress already exists and Overwrite is FALSE. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() with a *DenyFlag* value of FALSE and with valid TargetSwAddress / TargetHwAddress values.  4. Call EFI\_ARP\_PROTOCOL.Add() with a *DenyFlag* value of TRUE and with the same TargetHwAddress as the one used in the last call while Overwrite is FALSE.  The return status must be EFI\_ACCESS\_DENIED.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.8 | 0x31b66402, 0x4c9a, 0x486f, 0x9e, 0x68, 0xf5, 0xb1, 0x8b, 0x7b, 0xb4, 0xbf | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_ACCESS\_DENIED when the ARP cache entry of the same TargetHwAddress already exists and Overwrite is **FALSE**. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() with a DenyFlag value of FALSE and with valid TargetSwAddress/TargetHwAddress .  4. Call EFI\_ARP\_PROTOCOL.Add() with a *DenyFlag* value of **FALSE** and with the same TargetSwAddress/TargetHwAddress as the ones used in the last call while Overwrite is **FALSE**.  The return status must be EFI\_ACCESS\_DENIED.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.9 | 0x14c76af4, 0x29ca, 0x4018, 0x85, 0x6d, 0xfb, 0xfa, 0xfb, 0xae, 0x02, 0xa6 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_NOT\_STARTED when the ARP driver instance has not been configured and TargetHwAddress is valid, while *DenyFlag* is **TRUE**. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Add() with a DenyFlag value of TRUE and with valid TargetHwAddress .  The return status must be EFI\_NOT\_STARTED.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.10 | 0x8f07a21d, 0xfca8, 0x4d4a, 0xa7, 0x18, 0xaf, 0x80, 0x27, 0x46, 0x84, 0x40 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_NOT\_STARTED when the ARP driver instance has not been configured and TargetSwAddress is valid, while *DenyFlag* is **TRUE**. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Add() with a DenyFlag value of TRUE and a valid TargetSwAddress value.  The return status must be EFI\_NOT\_STARTED.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.11 | 0xf7e1b57e, 0x8499, 0x49b7, 0xa1, 0x35, 0xe0, 0x25, 0x7a, 0x68, 0x7c, 0xca | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_NOT\_STARTED when the ARP driver instance has not been configured and TargetSwAddress/TargetHwAddress are valid, while *DenyFlag* is **FALSE**. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Add() with a DenyFlag value of FALSE and with valid TargetSwAddress*/*TargetHwAddress .  The return status must be EFI\_NOT\_STARTED.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.12 | 0x203039cb, 0xbfce, 0x472f, 0x9d, 0x46, 0xfe, 0x53, 0xcd, 0x47, 0x42, 0xb6 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_SUCCESS when Adding normal entry. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() with a DenyFlag value of FALSE and with valid TargetSwAddress*/*TargetHwAddress .  The return status must be EFI\_SUCCESS.  4. Call EFI\_ARP\_PROTOCOL.Request() with the same TargetSwAddress as the one added.  5. Call EFI\_ARP\_PROTOCOL.Request() with the TargetHwAddress added into the entry cache, and compare the TargetHwAddress brought back by it, then verify if they are the same.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.13 | 0x7e93dc4e, 0x2731, 0x41d4, 0x96, 0x89, 0x27, 0x3a, 0xfe, 0xdc, 0x26, 0x40 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_SUCCESS When overwrite is TRUE. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() with a DenyFlag value of FALSE and with valid TargetSwAddress*/*TargetHwAddress .  4. Call EFI\_ARP\_PROTOCOL.Add() with a *DenyFlag* value of **FALSE** and with the same *TargetSwAddress* as the one used in the last call and another different TargetHwAddress , while overwrite is **TRUE**.  The return status must be EFI\_SUCCESS.  5. Call EFI\_ARP\_PROTOCOL.Request() with the same *TargetSwAddress* as the one added.  6. Call EFI\_ARP\_PROTOCOL.Request() with the TargetHwAddress added at the second time, and compare the TargetHwAddress brought back by it, then verify if they are the same.  7. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.14 | 0xa00cc3c8, 0x005c, 0x4aed, 0xa1, 0x5c, 0x3e, 0x91, 0xca, 0x56, 0x33, 0xe5 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_SUCCESS when adding normal entry with Timeout set. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() with a DenyFlag value of FALSE and with valid TargetSwAddress*/*TargetHwAddress .  4. Call EFI\_ARP\_PROTOCOL.Add() to overwrite the exist entry with TimeoutValue set to be 50 seconds.  The return status must be EFI\_SUCCESS.  5. Call EFI\_ARP\_PROTOCOL.Request() with the same *TargetSwAddress* as the same one added.  6. Call EFI\_ARP\_PROTOCOL.Request() with the TargetHwAddress added at the second time, and compare the TargetHwAddress brought back, then verify if they are the same.  7. Stall 30 seconds and then call EFI\_ARP\_PROTOCOL.Request() and verify if the Address is correct again.  8. Stall 20 seconds to let entry timeout, then call EFI\_ARP\_PROTOCOL.Request(), and now the return status must be EFI\_NOT\_READY.  9. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.15 | 0x46eee5b0, 0x7a16, 0x4be3, 0x87, 0x9e, 0xb6, 0x4f, 0xaa, 0xd0, 0xc0, 0x65 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_SUCCESS when adding normal entry after the request call. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress.  The return status must be EFI\_NOT\_READY.  4. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry with the same *TargetSwAddress* as the one used in Request().  The return status must be EFI\_SUCCESS.  5. Call EFI\_ARP\_PROTOCOL.Request() with the TargetHwAddress added, and compare the TargetHwAddress brought back by it, then verify if they are the same.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.16 | 0x01321dca, 0xe8d4, 0x4022, 0xb7, 0xa1, 0xd6, 0x69, 0xca, 0xcb, 0x52, 0x0b | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_SUCCESS when adding denied entry. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a deny entry with the valid *TargetSwAddress*.  The return status must be EFI\_SUCCESS.  4. Call EFI\_ARP\_PROTOCOL.Request() with the same *TargetSwAddress* as the one used in the last call.  The return status must be EFI\_ACCESS\_DENIED.  5. Call EFI\_ARP\_PROTOCOL.Request() with a TargetHwAddress value of “0.0.0.0.0.0”, and compare the TargetHwAddress brought back by it, then verify if they are the same.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.17 | 0x7856bfd5, 0x758a, 0x4bcf, 0x9d, 0xc9, 0x2e, 0x36, 0x9a, 0xea, 0xf7, 0xdf | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_SUCCESS when adding denied entry with a overwrite value of **TRUE**. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a deny entry with the valid TargetHwAddress (0:2:3:4:5:6) and overwrite value of **TRUE**.  The return status must be EFI\_SUCCESS.  4. Call EFI\_ARP\_PROTOCOL.Request() with valid *TargetSwAddress* (172.16.210.161). The OS side should capture the request packet sent from the EUT side.  5. If captured, the OS side configures the ARP reply packet with source IP “172.16.210.161”, source Mac “0:2:3:4:5:6”. Then send the packet back to EUT side.  6. Then the OS side configures another ARP reply packet with source IP “172.16.210.161”, source Mac “0:2:3:4:5:7”. Then sends the second packet back to EUT side.  The return status must be **EFI\_ACCESS\_DENIED**.  7. Call EFI\_ARP\_PROTOCOL.Request() with a TargetHwAddress value of “0:2:3:4:5:7” and compare the TargetHwAddress brought back by it, then verify if they are the same.  8. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.18 | 0xefcdb906, 0xa43a, 0x437f, 0x81, 0x35, 0xe0, 0xef, 0xea, 0xd3, 0xdc, 0x0a | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_SUCCESS – Add denied entry with overwrite is **TRUE**. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry with the valid *TargetSwAddress*”172.16.210.161” and TargetHwAddress “0:2:3:4:5:6”.  4. Call EFI\_ARP\_PROTOCOL.Add() to overwrite the existed entry with a deny entry and the TargetHwAddress is still “0:2: 3:4:5:6”.  The return status must be EFI\_SUCCESS.  5. Call EFI\_ARP\_PROTOCOL.Request() with valid *TargetSwAddress* (172.16.210.161). The OS side should capture the request packet sent from the EUT side.  6. If having captured, the OS side configures the ARP reply packet with sender IP “172.16.210.161”, sender Mac “0:2:3:4:5:6”. Then send the packet back to EUT side.  7. Then the OS side configures another ARP reply packet with sender IP “172.16.210.161”, sender Mac “0:2:3:4:5:7”. Then send the second packet back to EUT side.  The return status must be **EFI\_ACCESS\_DENIED**.  8. Compare the TargetHwAddress brought back by EFI\_ARP\_PROTOCOL.Request() with “0:2:3:4:5:7” and verify if they are same.  8. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.19 | 0xccf3f6de, 0x5d43, 0x4dfa, 0xbe, 0x65, 0xe8, 0xc5, 0x3d, 0xe0, 0xdf, 0x95 | EFI\_ARP\_PROTOCOL.Add() – returns EFI\_SUCCESS when adding denied entry with overwrite value of **TRUE**. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry with the valid *TargetSwAddress*”172.16.210.161” and TargetHwAddress “0:2:3:4:5:6”.  4. Call EFI\_ARP\_PROTOCOL.Add() to overwrite the existed entry with a deny entry and the TargetHwAddress is still “0:2: 3:4:5:6”.  The return status must be EFI\_SUCCESS.  5. Call EFI\_ARP\_PROTOCOL.Request() with valid *TargetSwAddress* (172.16.210.161). The OS side should capture the request packet sent from the EUT side.  6. If having captured, the OS side configures the ARP reply packet with sender IP “172.16.210.161”, sender Mac “0:2:3:4:5:6”. Then send the packet back to EUT side.  7. Then the OS side configures another ARP reply packet with sender IP “172.16.210.161”, sender Mac “0:2:3:4:5:7”. Then send the second packet back to EUT side.  The return status must be **EFI\_ACCESS\_DENIED**.  8. Compare the TargetHwAddress brought back by EFI\_ARP\_PROTOCOL.Request() with “0:2:3:4:5:7” and verify if they are same.  8. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment.  9. Call EFI\_ARP\_PROTOCOL.Add() to overwrite the exist entry with a deny entry whose *TargetSwAddress* is “172.16.210.161”.  The return status must be EFI\_SUCCESS. |
| 5.24.1.1.19 (continued) |  |  | 10. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress (172.16.210.161).  The return status must be  EFI\_ACCESS\_DENIED. |
| 5.24.1.1.20 | 0xb294d2a8, 0xb3f7, 0x4ec0, 0xa1, 0x4c, 0x74, 0xa9, 0x6d, 0xcc, 0x56, 0xb7 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_SUCCESS when adding denied entry with Timeout set. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a deny entry whose *TargetSwAddress* is ”172.16.210.161” and a Timeout value of set to be 50.  The return status must be EFI\_SUCCESS.  4. Call EFI\_ARP\_PROTOCOL.Request() with valid *TargetSwAddress* (172.16.210.161).  The return status must be  **EFI\_ACCESS\_DENIED** and the return TargetHwAddress must be “0:0:0:0:0:0”.  5. Stall 30 seconds, call EFI\_ARP\_PROTOCOL.Request() again with valid *TargetSwAddress*”172.16.210.161”.  The return status must be **EFI\_ACCESS\_DENIED** and the return TargetHwAddress must be “0:0:0:0:0:0”.  6. Stall 20 seconds, call EFI\_ARP\_PROTOCOL.Request() again with valid *TargetSwAddress*”172.16.210.161”.This time the return status must be EFI\_NOT\_READY.  7. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.1.21 | 0x48d3af46, 0x09db, 0x4c34, 0xb9, 0x1e, 0xb0, 0x48, 0xe0, 0x1a, 0x9d, 0x17 | EFI\_ARP\_PROTOCOL.Add() - returns EFI\_SUCCESS when adding denied entry. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid *TargetSwAddress* (172.16.210.161).  The return status must be  EFI\_NOT\_READY.  4. Call EFI\_ARP\_PROTOCOL.Add() to add a deny entry whose *TargetSwAddress* is ”172.16.210.161”.  The return status must be EFI\_SUCCESS.  5. Verify if the return TargetHwAddress is “0:0:0:0:0:0”.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |

### Cancel()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.1.2.1 | 0x56539533, 0xee7d, 0x4e57, 0xaf, 0x89, 0x2a, 0xa7, 0x3d, 0x82, 0x36, 0x61 | EFI\_ARP\_PROTOCOL.Cancel() - returns EFI\_INVALID\_PARAMETER with a TargetSwAddress value of invalid. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid *TargetSwAddress* (172.16.210.161).  The return status must be  EFI\_NOT\_READY.  4. Call EFI\_ARP\_PROTOCOL.Cancel() with a TargetSwAddress value of NULL and a ResolvedEvent value other than NULL.  The return status must be EFI\_INVALID\_PARAMETER.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.2.2 | 0xe9118c8c, 0x1e0e, 0x451b, 0x8f, 0x4f, 0xd6, 0x37, 0x8b, 0x82, 0xf3, 0x6a | EFI\_ARP\_PROTOCOL.Cancel() - returns EFI\_INVALID\_PARAMETER with an invalid *Event* value. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress (172.16.210.161).  The return status must be  EFI\_NOT\_READY.  4. Call EFI\_ARP\_PROTOCOL.Cancel() with an Event value of NULL and a TargetSwAddress value of not NULL.  The return status must be EFI\_INVALID\_PARAMETER.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.2.3 | 0x8b6cee26, 0x52c3, 0x45fe, 0xae, 0x7e, 0xfa, 0xa6, 0xd9, 0xc1, 0x80, 0xc7 | EFI\_ARP\_PROTOCOL.Cancel() - returns EFI\_NOT\_FOUND with Event not found. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress (172.16.210.161).  The return status must be  EFI\_NOT\_READY.  4. Call EFI\_ARP\_PROTOCOL.Cancel() with valid TargetSwAddress while *Event* is not issued by the EFI\_ARP\_PROTOCOL.Request().  The return status must be EFI\_NOT\_FOUND.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.2.4 | 0x09e570d8, 0xdc54, 0x4458, 0xb9, 0xa3, 0x58, 0x4f, 0xeb, 0x64, 0xc0, 0xdb | EFI\_ARP\_PROTOCOL.Cancel() - returns EFI\_NOT\_FOUND with TargetSwAddress not found. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress (172.16.210.161).  The return status must be  EFI\_NOT\_READY.  4. Call EFI\_ARP\_PROTOCOL.Cancel() with a TargetSwAddress value of “172.16.210.160” which is not issued by the EFI\_ARP\_PROTOCOL.Request().  The return status must be EFI\_NOT\_FOUND.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.2.5 | 0xbecb34c1, 0xbfed, 0x43c1, 0x81, 0xfe, 0xc5, 0x9f, 0x8d, 0xf4, 0xf2, 0x5a | EFI\_ARP\_PROTOCOL.Cancel() - returns EFI\_NOT\_FOUND with TargetSwAddress not found. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress (172.16.210.161).  The return status must be  EFI\_NOT\_READY.  4. Call EFI\_ARP\_PROTOCOL.Cancel() with a TargetSwAddress value of “172.16.210.160” which is not issued by the EFI\_ARP\_PROTOCOL.Request().  The return status must be EFI\_NOT\_FOUND.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.2.6 | 0x9511bd75, 0x971b, 0x4e14, 0xb2, 0xd1, 0x44, 0x9b, 0x2e, 0x0a, 0x90, 0x78 | EFI\_ARP\_PROTOCOL.Cancel() - returns EFI\_NOT\_FOUND with both the TargetSwAddress and Event value of NULL. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress (172.16.210.161).  The return status must be  EFI\_NOT\_READY.  4. Call EFI\_ARP\_PROTOCOL.Cancel()with both the TargetSwAddress and Event value of NULL.  The return status must be EFI\_NOT\_FOUND.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.2.7 | 0xd45a3a11, 0xf14c, 0x4dc2, 0x8d, 0x91, 0xfe, 0x0b, 0xa7, 0x14, 0xac, 0x97 | EFI\_ARP\_PROTOCOL.Cancel() - returns EFI\_NOT\_STARTED when the ARP driver instance has not been configured. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress (172.16.210.161).  The return status must be  EFI\_NOT\_READY.  4. Call EFI\_ARP\_PROTOCOL.Configure() with a *ConfigData* value of NULL to reset the ARP driver instance.  5. Call EFI\_ARP\_PROTOCOL.Cancel() with valid parameters which Request () had issued.  The return status must be EFI\_NOT\_STARTED.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.2.8 | 0x1b5f4fbb, 0x0d7d, 0x4b4c, 0xad, 0x29, 0x7b, 0x8b, 0xa5, 0x3e, 0xab, 0xc6 | EFI\_ARP\_PROTOCOL.Cancel() - returns EFI\_SUCCESS when canceling request. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with all valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress (172.16.210.161).  The return status must be  EFI\_NOT\_READY.  4. Call EFI\_ARP\_PROTOCOL.Cancel() with parameters issued by EFI\_ARP\_PROTOCOL.Request().  The return status must be EFI\_SUCCESS.  5. Then the OS side shouldn’t capture any packet sent from the EUT side.  6. Call EFI\_ARP\_PROTOCOL.Request() again, the return status should be EFI\_NOT\_READY and the return TargetHwAddress should be “0:0:0:0:0:0”.  7. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |

### Configure()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.1.3.1 | 0xcdbd6b40, 0x3b1f, 0x4cd5, 0x8b, 0xd9, 0x33, 0x99, 0x63, 0x8e, 0x80, 0x35 | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_INVALID\_PARAMETER with invalid SwAddressLength. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with a SwAddressLength value of 0.  The return status must be  EFI\_INVALID\_PARAMETER.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.2 | 0x072fb583, 0x5885, 0x4b2e, 0x99, 0x72, 0xe7, 0x2c, 0x5b, 0xd3, 0x34, 0xd5 | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_INVALID\_PARAMETER with invalid StationAddress. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with a StationAddress value of NULL.  The return status must be  EFI\_INVALID\_PARAMETER.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.3 | 0x3a8fde87, 0x1d5d, 0x462e, 0x8e, 0x3c, 0x01, 0xec, 0x3b, 0x9f, 0xf7, 0x5b | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_ACCESS\_DENIEDwhen the StationAddress is different from the one that has already registered. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Configure() with different StationAddress with the one that has already registerd.  The return status must be  EFI\_ACCESS\_DENIED.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.4 | 0x2747e156, 0xee8d, 0x4533, 0xb4, 0x63, 0xa8, 0xb0, 0x5f, 0xe0, 0x6b, 0xc1 | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_ACCESS\_DENIED when the *SwAddressLength* is different from the one that has already registered. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Configure() with a different *SwAddressLength* from the one that has already registered.  The return status must be  EFI\_ACCESS\_DENIED.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.5 | 0x790466e9, 0x0f6e, 0x4a3d, 0xa7, 0xdb, 0x5c, 0xb5, 0x6b, 0x59, 0x01, 0xef | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_ACCESS\_DENIED when the *SwAddressLength* is different from the one that has already registered. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Configure() with different *SwAddressLength* from the one that has already registerd.  The return status must be  EFI\_ACCESS\_DENIED.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.6 | 0xab90d4d0, 0xa0ac, 0x44c3, 0xb7, 0x03, 0x12, 0xdd, 0x10, 0x37, 0x74, 0x1d | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_ACCESS\_DENIED when the *SwAddressType* is different from the one that has already registered. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Configure() with different *SwAddressType* from the one that has already registered.  The return status must be  EFI\_ACCESS\_DENIED.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.7 | 0xf41970a5, 0x733f, 0x47d4, 0x8f, 0x52, 0xd5, 0x5c, 0x86, 0xd7, 0x96, 0x9f | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_ACCESS\_DENIED when the *SwAddressType* is different from the one that has already registered. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Configure() with different *SwAddressType* from the one that has already registerd.  The return status must be  EFI\_ACCESS\_DENIED.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.8 | 0x8b9bcd53, 0x9a83, 0x45c0, 0x9b, 0x5f, 0xf2, 0x99, 0x2c, 0x78, 0xf8, 0x1b | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_SUCCESS with valid parameters. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress “172.16.210.161”.  4. The OS side should capture the request packet, and send back the ARP reply packet filled with source IP”172.16.210.161” and source MAC “0:2:3:4:5:6”.  5. The return status must be EFI\_NOT\_READY and the return TargetHwAddress should be“0:2:3:4:5:6”.  6. The OS side sends a request packet to resolve IP “172.16.210.102” with the source IP”172.16.210.161” and source MAC”0:2:3:4:5:7”.  7. Then OS should capture the ARP reply packet sent from the EUT side.  8. If having captured, call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress “172.16.210.161”.  9. The return TargetHwAddress must be “0:2:3:4:5:7”, and  The return status must be  EFI\_SUCCESS.  10. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.9 | 0xeee99be3, 0xa701, 0x4612, 0x98, 0x1a, 0xad, 0x8c, 0x06, 0x4a, 0xd7, 0xa5 | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_SUCCESS with valid parameters. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Configure() with a *ConfigData* value of NULL to reset the ARP driver instance.  4. Call EFI\_ARP\_PROTOCOL.Request(), the return status should be EFI\_NOT\_STARTED.  5. Call EFI\_ARP\_PROTOCOL.Configure() again with valid parameters.  6. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress “172.16.210.161”.  7. The OS side should capture the request packet, and send back the ARP reply packet filled with source IP”172.16.210.161” and source MAC “0:2:3:4:5:6”.  The return status must be EFI\_SUCCESS.  8. Verify if the return TargetHwAddress is “0:2:3:4:5:6”.  9 Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.10 | 0x4423e5b6, 0x6f3c, 0x41c3, 0x8c, 0x50, 0xea, 0x71, 0xd8, 0x52, 0x3b, 0x74 | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_SUCCESS with parameter timeout set. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters with timeout set to be 50.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress “172.16.210.161”.  4. The OS side should capture the request packet, and send back the ARP reply packet filled with source IP”172.16.210.161” and source MAC “0:2:3:4:5:6”.  The return status must be EFI\_SUCCESS.  5. Verify if the return TargetHwAddress is “0:2:3:4:5:6”.  6. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress “172.16.210.161” again.  7. The return status should be EFI\_SUCCESS and the TargetHwAddress be “0:2:3:4:5:6”.  8. Stall 30 seconds, call EFI\_ARP\_PROTOCOL.Request() like the step 6 again.  9. The return status should be EFI\_SUCCESS and the TargetHwAddress be “0:2:3:4:5:6”.  10. Stall 20 seconds, call EFI\_ARP\_PROTOCOL.Request() like the step 6 again.  11. This time the return status should be **EFI\_NOT\_READY** and the TargetHwAddress be “0:0:0:0:0:0”.  12. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.11 | 0x79f9aacd, 0xfb79, 0x4746, 0x8f, 0x5c, 0x38, 0x4b, 0xf9, 0x2e, 0x0a, 0x53 | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_SUCCESS and packet count is correct when *ConfigData.RetryCount* is 5. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with  The return status must be EFI\_SUCCESS.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress “172.16.210.161”.  4. The OS side should capture the request packet for 5 times.  The return status should be **EFI\_NOT\_READY** and the TargetHwAddress should be “0:0:0:0:0:0”.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.12 | 0x970634b0, 0x57a5, 0x40c5, 0x92, 0x01, 0xcb, 0xb2, 0x00, 0x8c, 0xbb, 0x43 | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_SUCCESS with valid parameters. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with EntryTimeOut, RetryCount, RetryTimeOut value of 0.  3. Call EFI\_ARP\_PROTOCOL.Configure() with a *EntryTimeOut* value of 5000000, a *RetryCount* value of 30, and a *RetryTimeOut* value of 5000000.  4. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress “172.16.210.161”.  4. The OS side should capture the request packet.  5. If having captured, the OS side sends an ARP reply back with source IP “172.16.210.161”, source MAC “0:2:3:4:5:6”.  The return status must be EFI\_SUCCESS.  In addition, the TargetHwAddress should be “0:2:3:4:5:6”.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.13 | 0xc6c2e0c3, 0x9715, 0x48a8, 0x86, 0xba, 0x36, 0xbd, 0xac, 0x70, 0x71, 0x6d | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_SUCCESS when SwAddressLength is 1. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with a SwAddressLength value of 1.  The return status must be EFI\_SUCCESS.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress “171.16.210.161”.  4. The OS side should capture the request packet.  5. If having captured, the OS side sends an ARP reply back with source IP “171”, source MAC “0:2:3:4:5:6”, Target IP “172”.  The return status should be **EFI\_NOT\_READY** and the TargetHwAddress should be “0:2:3:4:5:6”.  6. The OS sends an ARP request to the broadcast address with source IP “171” and source MAC” 0:2:3:4:5:6” to resolve Target IP “172”.  7. The OS should capture the packet.  8. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.3.14 | 0xf4972462, 0x1dc5, 0x484f, 0xb6, 0x55, 0x8b, 0x2e, 0x89, 0xec, 0x2c, 0x46 | EFI\_ARP\_PROTOCOL.Configure() - returns EFI\_SUCCESS when SwAddressLength is 16. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with a SwAddressLength value of 16.  The return status must be EFI\_SUCCESS.  3. Call EFI\_ARP\_PROTOCOL.Request() with valid TargetSwAddress “171.16.210.161”.  4. The OS side should capture the request packet.  5. If having captured, the OS side sends an ARP reply back filled with source IP “172.16.210.161.0.0.0.0.0.0.0.0.0.0.0.0”, source MAC “0:2:3:4:5:6”, Target IP “172.16.210.102.0.0.0.0.0.0.0.0.0.0.0.0”.  The return status should be “EFI\_NOT\_READY” and the TargetHwAddress “0:2:3:4:5:6”.  6. The OS sends an ARP request to the broadcast address with source IP “172.16.210.161.0.0.0.0.0.0.0.0.0.0.0.0” and source MAC” 0:2:3:4:5:6” to resolve Target IP “172.16.210.102.0.0.0.0.0.0.0.0.0.0.0.0”.  7. The OS should capture the packet.  8. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |

### Delete()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.1.4.1 | 0x1ba44874, 0x8e16, 0x422e, 0x97, 0x73, 0x43, 0x6f, 0x06, 0x2f, 0x6f, 0x01 | EFI\_ARP\_PROTOCOL.Delete() - returns EFI\_NOT\_FOUND when the specified deletion key of MacAddress is not found. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Delete() to delete the added entry with key specified as IpAddress which is the same as the TargetSwAddress of added entry.  5. Call EFI\_ARP\_PROTOCOL.Delete() again with key specified as MacAddress which is the same with the TargetHwAddress of added entry.  The return status must be  EFI\_NOT\_FOUND.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.4.2 | 0xab90c68f, 0xa0af, 0x4188, 0x9a, 0x74, 0x66, 0x5a, 0x9c, 0x8a, 0x4b, 0x92 | EFI\_ARP\_PROTOCOL.Delete() - returns EFI\_NOT\_FOUND when the specified deletion key of IpAddress was not found. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Delete() to delete the added entry with key specified as MacAddress which is the same with the TargetHwAddress of the added entry.  5. Call EFI\_ARP\_PROTOCOL.Delete() again with key specified as IpAddress which is the same as the TargetSwAddress of the added entry.  The return status must be  EFI\_NOT\_FOUND.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.4.3 | 0xe03b088c, 0x8cf0, 0x4db9, 0xa0, 0xc1, 0x77, 0xa9, 0xf4, 0x1a, 0xce, 0x0a | EFI\_ARP\_PROTOCOL.Delete() - returns EFI\_NOT\_STARTED when ARP driver instance has not been configured. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Delete() with key specified as IpAddress.  The return status must be  EFI\_NOT\_STARTED.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.4.4 | 0x4b8b9c7f, 0x96fc, 0x41fb, 0xbc, 0x58, 0x32, 0x1d, 0x13, 0x75, 0xed, 0x7b | EFI\_ARP\_PROTOCOL.Delete() - returns EFI\_NOT\_STARTED when ARP driver instance has not been configured. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Delete() with key specified as MacAddress.  The return status must be  EFI\_NOT\_STARTED.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.4.5 | 0x494278d5, 0x4ff5, 0x4ac5, 0x9e, 0xd7, 0xfa, 0x53, 0xa1, 0x7e, 0x03, 0xed | EFI\_ARP\_PROTOCOL.Delete() - returns EFI\_SUCCESS when deleting the normal entry. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Delete() to delete the added entry with key specified as IpAddress which is the same as the TargetSwAddress of the added entry.  The return status must be  EFI\_SUCCESS.  5. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress which is the same as the TargetSwAddress of the added entry.  The return status must be **EFI\_NOT\_READY**.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.4.6 | 0xd2477a4f, 0xef0d, 0x46a2, 0x9a, 0x86, 0x32, 0x82, 0x3f, 0x2c, 0x4b, 0xa3 | EFI\_ARP\_PROTOCOL.Delete() - returns EFI\_SUCCESS when deleting the normal entry. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Delete() to delete the added entry with key specified as MacAddress which is the same as the TargetHwAddress of the added entry.  The return status must be  EFI\_SUCCESS.  5. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress which is the same as the TargetSwAddress of the added entry.  The return status must be **EFI\_NOT\_READY**.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.4.7 | 0x1e618ee9, 0x40b9, 0x4f79, 0xb9, 0x26, 0xee, 0x2b, 0xa3, 0x73, 0x51, 0x4c | EFI\_ARP\_PROTOCOL.Delete() - returns EFI\_SUCCESS when deleting all entries. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Delete()with AddressBuffer set to NULL and *BySwAddress* set to **TRUE** to delete all entries.  The return status must be  EFI\_SUCCESS.  5. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress which is the same as the TargetSwAddress of the added entry.  The return status must be **EFI\_NOT\_READY**.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.4.8 | 0x34a1c3fa, 0xf335, 0x471d, 0x83, 0x03, 0xef, 0x50, 0x98, 0xa3, 0x05, 0x30 | EFI\_ARP\_PROTOCOL.Delete() - returns EFI\_SUCCESS when deleting all entries. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Delete()with AddressBuffer set to NULL and *BySwAddress* set to **FALSE** to delete all entries.  The return status must be  EFI\_SUCCESS.  5. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress which is the same as the TargetSwAddress of the added entry.  The return status must be **EFI\_NOT\_READY**.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |

### Find()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.1.5.1 | 0x16bcb5a1, 0xf2c1, 0x419a, 0x8a, 0xf1, 0xea, 0x4b, 0xd9, 0x89, 0x5f, 0xda | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_INVALID\_PARAMETER when both EntryLength and EntryCount are NULL and Refresh is FALSE. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Find() with both EntryLength and EntryCount are NULL, BySwAddress is TRUE,and Refresh is FALSE.  The return status must be  EFI\_INVALID\_PARAMETER.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.2 | 0x210ce61b, 0xa76d, 0x4c56, 0xbe, 0x24, 0xe7, 0xb8, 0x11, 0x50, 0xd7, 0x10 | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_INVALID\_PARAMETER when both EntryLength and EntryCount are NULL and Refresh is FALSE. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Find() with both the EntryLength and EntryCount value of NULL and a BySwAddress value of FALSE while Refresh is FALSE.  The return status must be  EFI\_INVALID\_PARAMETER.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.3 | 0xf6244c19, 0x6e26, 0x4b9e, 0x84, 0xd3, 0x43, 0x65, 0xb7, 0x6c, 0x17, 0x39 | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_INVALID\_PARAMETER when both EntryLength and EntryCount are NULL and Entries are not NULL while Refresh is TRUE. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Find() with both EntryLength and EntryCount are NULL and Entries are not NULL while Refresh is TRUE.  The return status must be  EFI\_INVALID\_PARAMETER.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.4 | 0x5508b3bb, 0x7062, 0x46e7, 0xa4, 0x31, 0xf2, 0xed, 0x67, 0x0b, 0xee, 0x61 | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_NOT\_FOUND when no matching entries were found. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Find() with the specified IpAddress.  The return status must be  EFI\_NOT\_FOUND.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.5 | 0x9d95d0d7, 0x8e23, 0x4db4, 0xb1, 0xb6, 0x76, 0xc2, 0xee, 0xdc, 0x0f, 0x4f | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_NOT\_FOUND when no matching entries were found. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Find() with the specified MacAddress.  The return status must be  EFI\_NOT\_FOUND.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.6 | 0x056e9bc8, 0xb221, 0x4063, 0xa2, 0x59, 0x19, 0xe0, 0x08, 0xff, 0x86, 0xda | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_NOT\_FOUND when no matching entries were found. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Find() with the specified IpAddress and a *Refresh* value of **TRUE**.  The return status must be  EFI\_NOT\_FOUND.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.7 | 0xc8b3f76f, 0x5ec3, 0x40f6, 0x98, 0x72, 0x31, 0xea, 0x23, 0x6f, 0xc8, 0x08 | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_NOT\_FOUND when no matching entries were found. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Find() with the specified MacAddress and a *Refresh* value of **TRUE**.  The return status must be  EFI\_NOT\_FOUND.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.8 | 0xe0814da9, 0x47fb, 0x443d, 0x84, 0xce, 0xaf, 0x65, 0x01, 0x33, 0x3f, 0x69 | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_NOT\_FOUND when no matching entries were found. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Find() with AddressBuffer set to NULL and *BySwAddress* set to **FALSE** while *Refresh* is **TRUE** so as to *refresh* all the entries.  The return status must be  EFI\_NOT\_FOUND.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.9 | 0xdb367aca, 0xbc94, 0x4c36, 0x92, 0xbd, 0x3b, 0xba, 0x16, 0x9e, 0xc0, 0x6e | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_NOT\_FOUND when no matching entries were found. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Find() with AddressBuffer set to NULL and *BySwAddress* set to **TRUE** while *Refresh* is **TRUE** so as to *refresh* all the entries.  The return status must be  EFI\_NOT\_FOUND.  4. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.10 | 0x883abd28, 0xd498, 0x4868, 0xb1, 0xa7, 0xe3, 0x22, 0xd1, 0x22, 0x6a, 0x12 | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_NOT\_STARTED when the ARP driver instance has not been configured. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Find() with specified key of IpAddress when *Refresh* is **FALSE**.  The return status must be  EFI\_NOT\_STARTED.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.11 | 0x9301dc5d, 0xc1f2, 0x4858, 0x93, 0xcf, 0xda, 0x77, 0x96, 0xa6, 0x2a, 0x8f | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_NOT\_STARTED when the ARP driver instance has not been configured. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddressand *Refresh* is **TRUE**.  The return status must be  EFI\_NOT\_STARTED.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.12 | 0x6350837b, 0x0e0e, 0x4241, 0xbd, 0x10, 0x87, 0x77, 0xb3, 0x35, 0xa7, 0xd3 | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_SUCCESS when finding the entry. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Add() to add another normal entry with the same TargetHwAddress as the one used in the first call to EFI\_ARP\_PROTOCOL.Add(), while the TargetSwAddress is different.  5. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress that is the same as the TargetHwAddress in the EFI\_ARP\_PROTOCOL.Add() call.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2. |
| 5.24.1.5.13 | 0x81716a64, 0x63db, 0x4625, 0xad, 0x87, 0xf1, 0x23, 0x46, 0x94, 0x9f, 0xa9 | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_SUCCESS when finding the entry. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Add() to add another normal entry with the same TargetHwAddress as the first EFI\_ARP\_PROTOCOL.Add(), while the TargetSwAddress is different.  5. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress that is the same as the TargetHwAddress in the EFI\_ARP\_PROTOCOL.Add() call.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2.  6. Call EFI\_ARP\_PROTOCOL.Delete() to delete the entry added in the second time.  7. Call EFI\_ARP\_PROTOCOL.Find() with specified key of IpAddress that is the same as the TargetSwAddress in the first EFI\_ARP\_PROTOCOL.Add() call.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x1.  8. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.14 | 0x34fd32ad, 0x8e3e, 0x4f49, 0xa0, 0xd7, 0xcc, 0xca, 0xac, 0xa3, 0xce, 0x1f | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_SUCCESS when finding the entry. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Add() to add another normal entry with the same TargetHwAddress as the one used in the first call to EFI\_ARP\_PROTOCOL.Add(), while the TargetSwAddress is different.  5. Call EFI\_ARP\_PROTOCOL.Find() with AddressBuffer set to NULL to find all the entries.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2. |
| 5.24.1.5.15 | 0x3b98d05b, 0x0cd1, 0x41a3, 0xa4, 0x8b, 0x2c, 0xe3, 0x37, 0x6e, 0x0f, 0x09 | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_SUCCESS when finding the entry. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Add() to add another normal entry with the same TargetHwAddress as the first EFI\_ARP\_PROTOCOL.Add(), while the TargetSwAddress is different.  5. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress that is the same as the TargetHwAddress in the EFI\_ARP\_PROTOCOL.Add() call.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2.  6. Call EFI\_ARP\_PROTOCOL.Delete() to delete the entry added in the second time.  7. Call EFI\_ARP\_PROTOCOL.Find() with AddressBuffer set to NULL to find all the entries.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x1.  8. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.16 | 0x0c8090e4, 0xa0c5, 0x427f, 0xa2, 0xf9, 0x34, 0xd8, 0x10, 0x91, 0x11, 0x2f | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_SUCCESS when finding the entry with refreshing. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry with timeout set to 50s.  4. Call EFI\_ARP\_PROTOCOL.Add() to add another normal entry with the same TargetHwAddress as the one used in the first all to EFI\_ARP\_PROTOCOL.Add(),while the TargetSwAddress is different. In addition, timeout is set to 50s.  5. Stall 20 s.  6. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress the same as TargetHwAddress in the EFI\_ARP\_PROTOCOL.Add( ) call when **refresh** is **TRUE**.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2.  7. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.17 | 0x89474dd0, 0x461b, 0x49c3, 0xa8, 0x5e, 0xaa, 0x16, 0x74, 0xad, 0x6f, 0x9d | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_SUCCESS when finding the entry without refreshing. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry with timeout set to 50s.  4. Call EFI\_ARP\_PROTOCOL.Add() to add another normal entry with the same TargetHwAddress as the one used in the first all to EFI\_ARP\_PROTOCOL.Add(),while the TargetSwAddress is different. In addition, timeout is set to 50s.  5. Stall 20 s.  6. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress the same as TargetHwAddress in the **EFI\_ARP\_PROTOCOL.Add(**) call when *refresh* is **TRUE**.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2.  7. Stall 35 s.  8. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress the same as TargetHwAddress in the **EFI\_ARP\_PROTOCOL.Add()** call and a *refresh* value of **FALSE**.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x16 and the return *EntryCount* should be 0x2.  9. Stall 20 s.  10. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress the same as TargetHwAddress in the EFI\_ARP\_PROTOCOL.Add( ) call with a *refresh* value of **FALSE**.  The return status must be **EFI\_NOT\_FOUND**.  11. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.18 | 0x97fbb88f, 0x0566, 0x4b4b, 0x93, 0xfe, 0x5e, 0xc9, 0xad, 0x60, 0x8d, 0x7e | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_SUCCESS when finding the entry with a SwAddressLength value of 16. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with a SwAddressLength value of 16.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Add() to add another normal entry with the same TargetHwAddress as the one used in the first call to EFI\_ARP\_PROTOCOL.Add(), while TargetSwAddress is different.  5. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress the same as TargetHwAddress in the **EFI\_ARP\_PROTOCOL.Add()** call.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x22 and the return *EntryCount* should be 0x2.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.19 | 0xcbd6f47d, 0x2edc, 0x4235, 0x91, 0x50, 0x1f, 0xba, 0xe9, 0x07, 0xac, 0x26 | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_SUCCESS when finding the entry with a SwAddressLength value of 16. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with a SwAddressLength value of 16.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Add() to add another normal entry with the same TargetHwAddress as the one used in the first call to **EFI\_ARP\_PROTOCOL.Add()**, while TargetSwAddress is different.  5. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress the same as TargetHwAddress in the **EFI\_ARP\_PROTOCOL.Add()** call.  The return status must be EFI\_SUCCESS.  In addition, the return **EntryLength** should be 0x22 and the return **EntryCount** should be 0x2.  6. Call EFI\_ARP\_PROTOCOL.Delete() to delete the entry added in the second time.  7. Call EFI\_ARP\_PROTOCOL.Find() with specified key of IpAddress the same as TargetSwAddress in the first **EFI\_ARP\_PROTOCOL.Add()** call.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x22 and the return *EntryCount* should be 0x1.  8. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.20 | 0x630e139e, 0x287a, 0x456c, 0xa5, 0xf7, 0x58, 0x35, 0xaf, 0x42, 0xf7, 0x7d | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_SUCCESS when finding the entry with a SwAddressLength value of 1. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with a SwAddressLength value of 1.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Add() to add another normal entry with the same TargetHwAddress as the first **EFI\_ARP\_PROTOCOL.Add()**, while the TargetSwAddress is different.  5. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress the same as the TargetHwAddress in the EFI\_ARP\_PROTOCOL.Add() call.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x13 and the return *EntryCount* should be 0x2.  8. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.5.21 | 0xf7c0f95a, 0xfaa2, 0x4577, 0x8c, 0x66, 0xb4, 0x76, 0x82, 0x00, 0x85, 0x5d | EFI\_ARP\_PROTOCOL.Find() - returns EFI\_SUCCESS when finding the entry with a SwAddressLength value of 1. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with a SwAddressLength value of 1.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Add() to add another normal entry with the same TargetHwAddress as the first **EFI\_ARP\_PROTOCOL.Add()**, while the TargetSwAddress is different.  5. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress the same as the TargetHwAddress in the **EFI\_ARP\_PROTOCOL.Add()** call.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x13 and the return *EntryCount* should be 0x2.  6. Call EFI\_ARP\_PROTOCOL.Delete() to delete the entry added in the second time.  7. Call EFI\_ARP\_PROTOCOL.Find() with specified key of IpAddress the same as the TargetSwAddress in the first **EFI\_ARP\_PROTOCOL.Add()** call.  The return status must be EFI\_SUCCESS.  In addition, the return *EntryLength* should be 0x13 and the return *EntryCount* should be 0x1.  8. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |

### Flush()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.1.6.1 | 0x057bd5b9, 0xc869, 0x4446, 0xa9, 0xd1, 0x79, 0x07, 0xdc, 0xf8, 0x74, 0xf0 | EFI\_ARP\_PROTOCOL.Flush() - returns EFI\_NOT\_FOUND when flushing the entry again after the first flush. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal permanent entry 1.  4. Call EFI\_ARP\_PROTOCOL.Add() to add a normal dynamic entry 2.  5. Call EFI\_ARP\_PROTOCOL.Add() to add a normal permanent entry 3.  6. Call EFI\_ARP\_PROTOCOL.Add() to add a normal dynamic entry 4.  7. Call EFI\_ARP\_PROTOCOL.Flush() to remove all dynamic cache entries.  8. Call EFI\_ARP\_PROTOCOL.Flush() again.  The return status must be  EFI\_NOT\_FOUND.  9. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.6.2 | 0xe34bd9b5, 0x94b2, 0x422a, 0xb8, 0xd1, 0x6c, 0x18, 0x07, 0x6c, 0xef, 0xbb | EFI\_ARP\_PROTOCOL.Flush() - returns EFI\_NOT\_STARTED when the arp driver instance has not been configured. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Flush().  The return status must be  EFI\_NOT\_STARTED.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.6.3 | 0xf2cc7ff1, 0x9049, 0x4daa, 0xa3, 0x4d, 0xca, 0x55, 0xf5, 0xe9, 0x67, 0x55 | EFI\_ARP\_PROTOCOL.Flush() - returns EFI\_SUCCESS when flushing the entry. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.e  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal permanent entry 1.  4. Call EFI\_ARP\_PROTOCOL.Add() to add a normal dynamic entry 2 – timeout is 50s.  5. Call EFI\_ARP\_PROTOCOL.Add() to add a normal permanent entry 3.  6. Call EFI\_ARP\_PROTOCOL.Add() to add a normal dynamic entry 4 – timeout is 50s.  7. Call EFI\_ARP\_PROTOCOL.Flush() to remove all dynamic cache entries.  The return status must be  EFI\_SUCCESS.  8. Call EFI\_ARP\_PROTOCOL.Find() with specified key of IpAddress the same as the TargetSwAddress in the first **EFI\_ARP\_PROTOCOL.Add()** call.  The return status must be EFI\_SUCCESS.  9. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress the same as the TargetHwAddress in the first **EFI\_ARP\_PROTOCOL.Add()** call.  The return status must be EFI\_SUCCESS.  10. Call EFI\_ARP\_PROTOCOL.Find() with specified key of IpAddress the same as the TargetSwAddress in the second **EFI\_ARP\_PROTOCOL.Add()** call.  The return status must be **EFI\_NOT\_FOUND**.  11. Call EFI\_ARP\_PROTOCOL.Find() with specified key of MacAddress as same as the TargetHwAddress in the second **EFI\_ARP\_PROTOCOL.Add()** call.  The return status must be **EFI\_NOT\_FOUND**.  14. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()to destroy the newly created Arp child handle and clean up the environment. |

### Request()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.1.7.1 | 0x464366ea, 0xf5a5, 0x47a0, 0x8b, 0x3b, 0x67, 0x09, 0x89, 0xcf, 0x43, 0xd2 | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_INVALID\_PARAMETER when TargetHwAddress is NULL. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Request() with a TargetHwAddress value of NULL ,and both the ResolvedEvent and TargetSwAdddress value other than NULL.  The return status must be  EFI\_INVALID\_PARAMETER.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.7.2 | 0xb4df082c, 0xb895, 0x4ec8, 0xac, 0xc7, 0x26, 0x58, 0x87, 0xc7, 0xe3, 0xbb | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_INVALID\_PARAMETER when TargetHwAddress is NULL. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Request() with a TargetHwAddress value of NULL ,a ResolvedEvent value of NULL, and a TargetSwAdddress value other than NULL.  The return status must be  EFI\_INVALID\_PARAMETER.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.7.3 | 0x58d0454a, 0xeed1, 0x4ccd, 0xa3, 0xd0, 0x10, 0xa5, 0xa8, 0x71, 0x46, 0x38 | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_INVALID\_PARAMETER when TargetHwAddress TargetHwAddress is NULL. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Request() with a TargetHwAddress value of NULL, and both the ResolvedEvent and *TargetSwAdddress* value of NULL.  The return status must be  EFI\_INVALID\_PARAMETER.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.7.4 | 0xe726cb6e, 0x3ee3, 0x474e, 0x9c, 0x1c, 0xa7, 0xc7, 0xa6, 0x93, 0x85, 0x1d | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_ACCESS\_DENIED when the requested Address is present in the deny address list. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry.  4. Call EFI\_ARP\_PROTOCOL.Add() to add a deny entry whose TargetSwAddress is the same as the one used in the first Add() call to overwrite the entry first added.  5. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress the same as the one used in the call to EFI\_ARP\_PROTOCOL.Add().  The return status must be  EFI\_ACCESS\_DENIED.  In addition, the return TargetHwAddress should be 0:0:0:0:0:0.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.7.5 | 0xd774703f, 0x7ed8, 0x48da, 0x9f, 0x86, 0x5e, 0xf8, 0x19, 0x47, 0xb6, 0x47 | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_NOT\_STARTED when the ARP driver instance has not been configured. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Request() when both TargetSwAddress and ResolvedEvent are not NULL.  The return status must be  EFI\_NOT\_STARTED.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.7.6 | 0x122d41e6, 0x252a, 0x4afb, 0xa2, 0x47, 0x03, 0x56, 0xd5, 0x3c, 0x4a, 0x64 | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_NOT\_STARTED when the ARP driver instance has not been configured. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Request() when TargetSwAddress is not NULL and ResolvedEvent is NULL.  The return status must be  EFI\_NOT\_STARTED.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.7.7 | 0xca3946d0, 0x64ff, 0x4139, 0x97, 0x66, 0x82, 0x91, 0xcb, 0xc1, 0x12, 0x09 | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_NOT\_STARTED – when the ARP driver instance has not been configured. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Request() when both TargetSwAddress and ResolvedEvent are NULL.  The return status must be  EFI\_NOT\_STARTED.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.7.8 | 0xf4b08f82, 0xdafd, 0x4618, 0x94, 0xed, 0x15, 0xf8, 0x54, 0xce, 0xe3, 0x9f | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_NOT\_READY – when the request has been started and is not finished. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress “172.16.210.161”.  4. The OS side should capture the request packet and send back the reply packet with SourceIp “172.16.210.161”, SourceMac “0:2:3:4:5:6”.  The return status must be  EFI\_NOT\_READY.  In addition, the return TargetHwAddress should be “0:2:3:4:5:6”.  10. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.7.9 | 0x3d6668d9, 0x631c, 0x4cee, 0xae, 0xc9, 0xc1, 0x0f, 0x3f, 0xe6, 0xee, 0x27 | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_NOT\_READY – when the request has been started and is not finished. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress “172.16.210.161”.  4. The OS side should capture the request packet and send back the reply packet with SourceIp “172.16.210.161”, SourceMac “0:2:3:4:5:6”.  The return status must be  EFI\_NOT\_READY.  In addition, the return TargetHwAddress should be “0:2:3:4:5:6”.  5. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry with TargetSwAddress “172.16.210.161” and TargetHwAddress “0:2:3:4:5:6”.  6. Call EFI\_ARP\_PROTOCOL.Add() to add a deny entry with the same TargetHwAddress as the one used in the first Add( ) to overwrite the entry first added.  7. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress “172.16.210.161”.  8. The OS side should capture the request packet and send back the first reply packet with SourceIp “172.16.210.161”, SourceMac “0:2:3:4:5:6”.  9. Then OS sends back the second reply packet with SourceIp “172.16.210.161”, SourceMac “0:2:3:4:5:7”.  The return status must be  EFI\_NOT\_READY.  In addition, the return TargetHwAddress should be “0:2:3:4:5:7”.  10. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.7.10 | 0xe37f681b, 0xab41, 0x4370, 0xab, 0x02, 0xf6, 0xd5, 0xfb, 0x0a, 0xf2, 0xb7 | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_SUCCESS when the data was copied from the ARP cache into the TargetHwAddress buffer. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress “172.16.210.161”.  4. The OS side should capture the request packet and validate whether the packet is rightly sent from the EUT side.  5. The OS sends back the reply packet with SourceIp “172.16.210.161”, SourceMac “0:2:3:4:5:6”.  The return status must be  EFI\_SUCCESS.  In addition, the return TargetHwAddress “0:2:3:4:5:6”.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment |
| 5.24.1.7.11 | 0x93e9a6d8, 0xb732, 0x40d7, 0x8d, 0x1e, 0xe5, 0xdb, 0xa6, 0xf6, 0x02, 0x1e | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_SUCCESS when the data was copied from the ARP cache into the TargetHwAddress buffer. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress “172.16.210.161”.  4. The OS side should capture the request packet and validate whether the packet is rightly sent from the EUT side.  5. The OS sends back the reply packet with SourceIp “172.16.210.161”, SourceMac “0:2:3:4:5:6”.  The return status must be  EFI\_SUCCESS.  In addition, the return TargetHwAddress “0:2:3:4:5:6”.  6. Call EFI\_ARP\_PROTOCOL.Request() with broadcast destination address to resolve TargetSwAddress “172.16.210.161”.  The return status must be  EFI\_SUCCESS.  In addition, the return TargetHwAddress should be “0:2:3:4:5:6”.  7. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment |
| 5.24.1.7.12 | 0xa227797d, 0x00b5, 0x4ff0, 0xb4, 0x62, 0x46, 0x87, 0xa1, 0x31, 0xa0, 0x1c | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_SUCCESS when the data was copied from the ARP cache into the TargetHwAddress buffer. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry with TargetSwAddress “172.16.210.161” and TargeHwAddress “0:2:3:4:5:6”  4. Call EFI\_ARP\_PROTOCOL.Request() with TargetSwAddress “172.16.210.161”.  The return status must be  EFI\_SUCCESS.  In addition, the return TargetHwAddress should be “0:2:3:4:5:6”.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment |
| 5.24.1.7.13 | 0xd958bbd5, 0x3429, 0x4b94, 0x9f, 0xe5, 0x8e, 0xe1, 0xf4, 0x8b, 0xfd, 0xd2 | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_SUCCESS when requesting the entry whose TargetSwAddress is a multicast IP address. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry with TargetSwAddress “172.16.210.161” and TargeHwAddress “0:2:3:4:5:6”  4. Call EFI\_ARP\_PROTOCOL.Request() to resolve multicast IP address “224.0.1.2”.  The return status must be  EFI\_SUCCESS.  In addition, the return TargetHwAddress should be “1:0:5e:0:1:2”.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.7.14 | 0x46146a28, 0x7af5, 0x43c5, 0xb7, 0xd1, 0x6f, 0xfb, 0xd6, 0xa4, 0x89, 0x97 | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_SUCCESS when requesting the entry whose TargetSwAddress is a multicast IP address. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry with TargetSwAddress “172.16.210.161” and TargetHwAddress “0:2:3:4:5:6”  4. Call EFI\_ARP\_PROTOCOL.Request() to resolve multicast IP address “238.255.255.255”.  The return status must be  EFI\_SUCCESS.  In addition, the return TargetHwAddress is “1:0:5e:7f: ff: ff”.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment |
| 5.24.1.7.15 | 0x50ecb99e, 0xfdab, 0x441c, 0x85, 0x08, 0x92, 0x5f, 0x1b, 0xdf, 0x42, 0x4b | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_SUCCESS when requesting the entry whose TargetSwAddress is NULL. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry with TargetSwAddress “172.16.210.161” and TargeHwAddress “0:2:3:4:5:6”  4. Call EFI\_ARP\_PROTOCOL.Request() when TargetSwAddress is NULL.  The return status must be  EFI\_SUCCESS.  In addition, the return TargetHwAddress should be “ff: ff: ff: ff: ff: ff”. (Network interface hardware broadcast address).  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment |
| 5.24.1.7.16 | 0x50d9cb20, 0x1177, 0x4b13, 0xbc, 0x41, 0xf0, 0xf3, 0x2a, 0x3d, 0xf9, 0x02 | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_SUCCESS when requesting the entry whose TargetSwAddress is “255.255.255.255”. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Add() to add a normal entry with TargetSwAddress “172.16.210.161” and TargeHwAddress “0:2:3:4:5:6”  4. Call EFI\_ARP\_PROTOCOL.Request() when TargetSwAddress is “255.255.255.255”.  The return status must be  EFI\_SUCCESS.  In addition, the return TargetHwAddress should be “ff: ff: ff: ff: ff: ff”. (Network interface hardware broadcast address).  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment |
| 5.24.1.7.17 | 0xf7140dcf, 0x0d15, 0x438a, 0xa3, 0x4d, 0x47, 0x23, 0x97, 0x6f, 0x0b, 0xc8 | EFI\_ARP\_PROTOCOL.Request() - returns EFI\_SUCCESS when calling Request () twice with the same TargetSwAddress. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Call EFI\_ARP\_PROTOCOL.Request() when TargetSwAddress is “172.16.210.161”.  The return status should be **EFI\_NOT\_READY**.  4. Call EFI\_ARP\_PROTOCOL.Request() again when TargetSwAddress is “172.16.210.161”.  The return status should be **EFI\_NOT\_READY**.  5. The OS side should capture the request packet and send back the reply packet with SouceIP “172.16.210.161” and SourceMac ”0:2:3:4:5:6”.  The return EventContext should be 2.  The return status must be  EFI\_SUCCESS.  In addition, the return TargetHwAddress should be “0:2:3:4:5:6”.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment |

### CreateChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.1.8.1 | 0xd01e591b, 0x6b83, 0x417c, 0xbf, 0xe0, 0x1d, 0xb3, 0x78, 0xea, 0x2c, 0x78 | EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - returns EFI\_INVALID\_PARAMETER with NULL child handle. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() with the parameter a ChildHandle value of NULL.  The return status must be EFI\_INVALID\_PARAMETER. |
| 5.24.1.8.2 | 0x51d66e16, 0x39f6, 0x4fff, 0x8a, 0x99, 0xf2, 0x95, 0x01, 0xe3, 0x4b, 0xe8 | EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - returns EFI\_INVALID\_PARAMETER with invalid child handle. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() with a invalid ChildHandle. The return status must be EFI\_INVALID\_PARAMETER. |
| 5.24.1.8.3 | 0x460a6262, 0xaa4d, 0x4e25, 0x92, 0x6b, 0x55, 0x1e, 0xf0, 0xb5, 0x6d, 0x37 | EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() – invokes CreateChild() to create different childs. | Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild()to create childs three times and then destroy them. |

### DestroyChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.1.9.1 | 0xfaabc3ef, 0xc56f, 0x44d1, 0xbe, 0xb6, 0x53, 0x5b, 0x26, 0x4d, 0xba, 0x63 | EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - returns EFI\_UNSUPPORTED with invalid child handle. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle  The return status must be EFI\_SUCCESS.  3. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() again with value of Handle set to be 8 and clean up the environment.  The return status must be EFI\_UNSUPPORTED. |
| 5.24.1.9.2 | 0x7b8de1fe, 0x93e1, 0x48a4, 0xa0, 0x5e, 0x38, 0xad, 0x8f, 0x26, 0xf0, 0x83 | EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - returns EFI\_INVALID\_PARAMETER with NULL child. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the NULL child. |
| 5.24.1.9.3 | 0xf651081a, 0xb71f, 0x4617, 0x99, 0x7a, 0xd1, 0x87, 0x7a, 0x07, 0x03, 0x28 | EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - returns EFI\_INVALID\_PARAMETER and inexistent child. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the inexistent child. |
| 5.24.1.9.4 | 0x5772a154, 0xb8f5, 0x4fec, 0xaa, 0x80, 0xae, 0xb9, 0x0c, 0x4c, 0xd2, 0x5d | EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() – invokes DestroyChild() to destroy different childs | Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL. DestroyChild ()to destroy the newly three created Arp childs. |

### RFC Related

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.1.10.1 | 0x0f6557a8, 0xf383, 0x436e, 0x96, 0x2b, 0x88, 0x2a, 0x28, 0x3c, 0x4c, 0x64 | EFI\_ARP\_PROTOCOL.Rfc – Send an ARP request and check the ARP reply . | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Send ARP request to the broadcast address with sender ip “172.16.210.161” and sender Mac”0:2:3:4:5:7” to resolve the Target ip”172.16.210.102”.  4. Then the OS side should capture the reply packet.  5. If having captured, dump the reply packet and validate whether the sender Mac is the MacAddress of TargetMachine.  6. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |
| 5.24.1.10.2 | 0x842c7377, 0x04b6, 0x459f, 0x92, 0x56, 0x39, 0xbf, 0x2e, 0x2f, 0xc5, 0x93 | EFI\_ARP\_PROTOCOL.Rfc – without reply when sending an ARP request with opcode invalid. | 1. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Arp child handle.  2. Call EFI\_ARP\_PROTOCOL.Configure() with valid parameters.  3. Send ARP request to the broadcast address with sender ip “172.16.210.161” and sender Mac”0:2:3:4:5:7” to resolve the Target ip”172.16.210.102” – the opcode set to 255.  4. Then the OS side shouldn’t capture the reply packet.  5. Call EFI\_ARP\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() to destroy the newly created Arp child handle and clean up the environment. |

### 

## EFI\_DHCP4\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_DHCP4\_PROTOCOL Section.

### GetModeData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.2.1.1 | 0x52159e94, 0x4a67, 0x44f6, 0x9b, 0x0b, 0x83, 0x21, 0x93, 0x41, 0xe1, 0xf3 | EFI\_DHCP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get all mode data when the Dhcp4 child has not been configured. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL**.**GetModeData() to get all mode data when the Dhcp4 child has not been configured. The ModeData.State should be Dhcp4Stopped.The return status should be EFI\_SUCCESS.  3. Call EFI\_DHCP4\_PROTOCOL**.Stop()** to verify the Dhcp4 child in the Dhcp4Stopped state.  4. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.1.2 | 0x969e5dac, 0x2097, 0x4a3f, 0xaa, 0x15, 0xb0, 0x6d, 0xff, 0x26, 0x48, 0xec | EFI\_DHCP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get DHCP4 mode data during the configuration process. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0".  3. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData() to get Dhcp4 mode data when the Dhcp4 child has been configured. The ModeData.State should be Dhcp4Init.The return status should be EFI\_SUCCESS.  4. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  5. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data during the configuration process. The ModeData.State should be Dhcp4Selecting.The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.1.3 | 0xca520116, 0x5097, 0x4cda, 0x80, 0x79, 0x4a, 0x9b, 0x8f, 0xdd, 0x88, 0x38 | EFI\_DHCP4\_PROTOCOL.GetModeData() - invokes GetModeData() to get DHCP4 mode data during the configuration process. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.1.24".  3. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData() to get Dhcp4 mode data when the Dhcp4 child has been configured. The ModeData.State should be Dhcp4InitReboot.The return status should be EFI\_SUCCESS.  4. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  5. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data during the configuration process. The ModeData.State should be Dhcp4Rebooting.The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |

### Configure()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.2.2.1 | 0xbd919c90, 0x708b, 0x4502, 0xad, 0xd7, 0xd5, 0x85, 0x30, 0x4b, 0x84, 0x0e | EFI\_DHCP4\_PROTOCOL.Configure() - invokes Configure()when this driver instance was not in the Dhcp4Stopped, Dhcp4Init, Dhcp4InitReboot, or Dhcp4Bound state. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a *CompletionEvent* value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data during the configuration process. The ModeData.State should be Dhcp4Selecting.The return status should be EFI\_SUCCESS.  5. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure the child during the configuration process. The return status should be **EFI\_ACCESS\_DENIED**.  6. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.2.2 | 0x57b62321, 0x14a8, 0x4412, 0xb4, 0x20, 0xad, 0x49, 0x5d, 0x6a, 0xab, 0xbb | EFI\_DHCP4\_PROTOCOL.Configure() - invokes Configure() when Another instance is already in a valid configured state. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child1.  2. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child2.  3. Call EFI\_DHCP4\_PROTOCOL.Configure() to configure child1.  4. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child2. The return status should be EFI\_ACCESS\_DENIED.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.2.3 | 0x5101b2b6, 0x8021, 0x4a04, 0x90, 0x83, 0xf6, 0x6b, 0x9f, 0x4d, 0x10, 0x1f | EFI\_DHCP4\_PROTOCOL.Configure() - invokes Configure() with invalid parameters, among which DiscoverTryCount is positive and DiscoverTimeout is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure() to configure the new instance with a *DiscoverTryCount* value of positive and a *DiscoverTimeout* value of NULL. The return status should be **EFI\_INVALID\_PATAMETER**.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.2.4 | 0x50f034a4, 0x2aa4, 0x4d1a, 0x8a, 0x8c, 0x9d, 0x7c, 0x06, 0x48, 0xc9, 0x35 | EFI\_DHCP4\_PROTOCOL.Configure() - invokes Configure() with invalid parameters, among which RequestTryCount is positive and RequestTimeout is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure() to configure the new instance with a *RequestTryCount* value of positive and a *RequestTimeout* value of NULL. The return status should be **EFI\_INVALID\_PATAMETER**.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.2.5 | 0xc199419b, 0x62b1, 0x4cda, 0xb4, 0x38, 0x9d, 0xcd, 0xed, 0x4d, 0x83, 0x6d | EFI\_DHCP4\_PROTOCOL.Configure() - invokes Configure() with invalid parameters, among which OptionCount is positive and OptionList is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure() to configure the new instance with a *OptionCount* value of positive and a *OptionList* value of NULL. The return status should be **EFI\_INVALID\_PATAMETER**.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.2.6 | 0xada01077, 0x4869, 0x4c21, 0x8f, 0x6d, 0x6e, 0x65, 0x93, 0x41, 0xbc, 0xa6 | EFI\_DHCP4\_PROTOCOL.Configure() - invokes Configure() with invalid parameters, except that ClientAddress is an invalid unicast address. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure() to configure the new instance with a *ClientAddress* value of an invalid unicast address. The return status should be **EFI\_INVALID\_PATAMETER**.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.2.7 | 0xde6079f0, 0x4aa4, 0x4665, 0x80, 0x8b, 0xa0, 0x22, 0x3c, 0x8b, 0xf6, 0x40 | EFI\_DHCP4\_PROTOCOL.Configure() - invokes Configure() to Validate the configuration data effect before and after calling Dhcp.start() to start the Configuration. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure() to configure child with setting *ClientAddress* "0.0.0.0".  3. Call EFI\_DHCP4\_PROTOCOL.GetModeData()to check the configuration data effect.  4. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a *CompletionEvent* value of NULL.  5. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init.  6. Call Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting *ClientAddress* "192.168.2.3".  7. Call EFI\_DHCP4\_PROTOCOL.GetModeData()to check the configuration data effect.  8. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a *CompletionEvent* value of NULL.  9. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init.  10. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.2.8 | 0x73401b2e, 0x30aa, 0x422d, 0xa3, 0xca, 0x9f, 0x36, 0x78, 0x1c, 0xfa, 0x94 | EFI\_DHCP4\_PROTOCOL.Configure() - invokes Configure() to Validate the configuration data effect before and after calling Dhcp->start to start the Configuration, Call Dhcp.stop() before calling Dhcp.start() again. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0".  3. Call EFI\_DHCP4\_PROTOCOL.GetModeData()to check the configuration data effect.  4. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a *CompletionEvent* value of NULL.  5. Call EFI\_DHCP4\_PROTOCOL.Stop()to stop the configuration.  6. Call Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting **ClientAddress** "192.168.2.3".  7. Call EFI\_DHCP4\_PROTOCOL.GetModeData()to check the configuration data effect.  8. Call EFI\_DHCP4\_PROTOCOL.Start() again to start the configuration process with a CompletionEvent value of NULL.  9. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init.  10. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.2.9 | 0x1a27208e, 0x08a8, 0x42a6, 0xb9, 0x3f, 0x8b, 0x95, 0x94, 0x24, 0x46, 0xb7 | EFI\_DHCP4\_PROTOCOL.Configure() - invokes Configure() with the following condition:  if one instance wants to make it possible for another instance to configure successfully, it must call EFI\_DHCP4\_PROTOCOL.Configure() with DhcpCfgData set to NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child1.  2. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child2.  3. Call EFI\_DHCP4\_PROTOCOL.Configure() to configure child1.  4. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child2. The return status should be EFI\_ACCESS\_DENIED.  5. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child1 with setting ConfigData to NULL. The return status should be EFI\_SUCCESS.  6. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child2. The return status should be EFI\_SUCCESS.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |

### Start()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.2.3.1 | 0xbac2be63, 0xd705, 0x4667, 0x9d, 0x1b, 0x04, 0xe0, 0x5e, 0xeb, 0xcb, 0x3a | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** when the driver instance is in the Dhcp4Stopped state. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value other than NULL. The return status should be EFI\_NOT\_STARTED.  3. Call EFI\_DHCP4\_PROTOCOL.Stop()to stop the configuration process.  4. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.2 | 0xc67ae0d7, 0x3401, 0x4daf, 0xa6, 0x4c, 0xb9, 0xa6, 0x0e, 0xea, 0x17, 0x71 | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** with no response during the specified timeout value. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0".  2. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value other than NULL. The return status should be EFI\_TIMEOUT.  3. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init.  4. Call EFI\_DHCP4\_PROTOCOL.Stop()to stop the configuration process.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.3 | 0xd7cd1980, 0x7509, 0x4612, 0x80, 0xc0, 0x5c, 0x21, 0x5b, 0x9e, 0x8e, 0x10 | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** while the user aborts the DHCP process. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0" and Dhcp4Callback=1(Callbackfunctionlist[1]=Aborted)  2. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value other than NULL. The return status should be EFI\_ABORTED.  3. Call EFI\_DHCP4\_PROTOCOL.Stop()to stop the configuration process.  4. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.4 | 0x580e7e81, 0x506d, 0x4339, 0xb7, 0xc2, 0x9f, 0x05, 0x53, 0x8f, 0xf5, 0xde | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** to start configuration process while another instance has already started the DHCP process. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child1.  2. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child2.  3. Call EFI\_DHCP4\_PROTOCOL.Configure() to configure child1.  4. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process of child1 with a CompletionEvent value other than NULL. The return status should be EFI\_SUCCESS.  5. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process of child2 with a CompletionEvent value of NULL. The return status should be EFI\_ALREADY\_STARTED.  6. Call EFI\_DHCP4\_PROTOCOL.Stop()to stop the configuration process of child1.  7. Call EFI\_DHCP4\_PROTOCOL.Stop()to stop the configuration process of child2.  8. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.5 | 0x8bd59e83, 0x3f3a, 0x4649, 0xb8, 0x61, 0x36, 0x56, 0x23, 0x5c, 0x8f, 0x7d | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4Init State and Asynchronous Mode. (Calling functions in sequence A). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of not NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.6 | 0xaca2403d, 0x458b, 0x4c8e, 0x8f, 0x77, 0x1f, 0x87, 0x85, 0x31, 0x08, 0xed | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4Init State and Asynchronous Mode. (Calling functions in sequence B). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value other than NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.7 | 0x7344b984, 0x306d, 0x467b, 0xa4, 0x3d, 0x36, 0x77, 0xf8, 0xc9, 0x79, 0x78 | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4Init State and Asynchronous Mode. (Calling functions in sequence C). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value other than NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPNAK packet. The ModeData.State should be Dhcp4Init.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.8 | 0xf9a23299, 0xeb65, 0x472b, 0xbe, 0x96, 0xe5, 0xea, 0x77, 0x2e, 0x03, 0xc0 | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4InitReboot State and Asynchronous Mode. (Calling functions in sequence A). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting *ClientAddress* "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a *CompletionEvent* value other than NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.9 | 0x723e3088, 0x5f48, 0x4b09, 0x9b, 0x17, 0x80, 0x45, 0x86, 0xf9, 0x9a, 0xad | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4InitReboot State and Asynchronous Mode. (Calling functions in sequence B). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value other than NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver havng stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.10 | 0xa8fcde55, 0x522b, 0x49ea, 0xbc, 0xe8, 0x6b, 0xea, 0x80, 0x57, 0x91, 0x21 | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4InitReboot State and Asynchronous Mode. (Calling functions in sequence C). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a *CompletionEvent* value other than NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPNAK packet. The ModeData.State should be Dhcp4Init.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.11 | 0x941de4e1, 0xc289, 0x417b, 0x87, 0xeb, 0xef, 0x3e, 0x1e, 0xc0, 0x12, 0x3d | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4Init State and synchronous Mode. (Calling functions in sequence A). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.12 | 0xff3f4b6d, 0x2b40, 0x49b5, 0xb9, 0xe0, 0x7e, 0x11, 0x8a, 0x73, 0x70, 0x0a | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4Init State and synchronous Mode. (Calling functions in sequence B). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.13 | 0x35972f03, 0x90dc, 0x41ae, 0x8e, 0x1e, 0x27, 0x72, 0x47, 0x3b, 0x06, 0xb6 | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4Init State and synchronous Mode. (Calling functions in sequence C). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPNAK packet. The ModeData.State should be Dhcp4Init.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.14 | 0x90924db4, 0x1237, 0x4d59, 0x88, 0xf3, 0x11, 0x8b, 0xed, 0x01, 0x80, 0xae | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4InitReboot State and synchronous Mode. (Calling functions in sequence A). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a *CompletionEvent* value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.15 | 0x434f1845, 0xd940, 0x4129, 0xaa, 0xeb, 0x7a, 0x1b, 0xe7, 0xe1, 0x39, 0x48 | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4InitReboot State and synchronous Mode. (Calling functions in sequence B). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.3.16 | 0x340ff4c6, 0x7412, 0x44d4, 0x8f, 0x33, 0xeb, 0xc2, 0x6f, 0x22, 0x1d, 0x0c | EFI\_DHCP4\_PROTOCOL.Start() - invokes **Start()** in Dhcp4InitReboot State and synchronous Mode. (Calling functions in sequence C). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPNAK packet. The ModeData.State should be Dhcp4Init.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |

### RenewRebind()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.2.4.1 | 0x15bdc212, 0xbad5, 0x4213, 0xb2, 0x38, 0x50, 0xac, 0x76, 0x18, 0xdc, 0x90 | EFI\_DHCP4\_PROTOCOL.RenewRebind() - invokes **RenewRebind()** when the driver instance is in the Dhcp4Stopped state. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.RenewRebind() with a RebindRequest value of TRUE. The return status should be EFI\_NOT\_STARTED.  3. Call EFI\_DHCP4\_PROTOCOL.RenewRebind()with a RebindRequest value of FALSE. The return status should be EFI\_NOT\_STARTED.  4. Call EFI\_DHCP4\_PROTOCOL.Stop()to stop the configuration process.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.4.2 | 0x2949dc87, 0xdbcd, 0x4d64, 0x8f, 0x10, 0x68, 0x2f, 0xa2, 0x27, 0xe0, 0x88 | EFI\_DHCP4\_PROTOCOL.RenewRebind() - invokes **RenewRebind()** while getting no response during the specified time. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call EFI\_DHCP4\_PROTOCOL.RenewRebind()with a RebindRequest value of FALSEand aCompletionEventvalue of NULL. The return status should be EFI\_TIMEOUT.  6. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.4.3 | 0xd7f4cb11, 0xc3dc, 0x421f, 0x98, 0x80, 0x5c, 0x2a, 0x2d, 0x73, 0x06, 0x02 | EFI\_DHCP4\_PROTOCOL.RenewRebind() - invokes **RenewRebind()** when the driver instance is not in the Dhcp4Bound state. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value other than NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having sent the DHCPREQUEST packet. The ModeData.State should be Dhcp4Rebooting.  5. Call EFI\_DHCP4\_PROTOCOL.RenewRebind()with a RebindRequest value of **TRUE** and aCompletionEventvalue of NULL. The return status should be EFI\_ACCESS\_DENIED.  6. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.4.4 | 0x38bb70ba, 0xb05c, 0x4431, 0xb4, 0xf9, 0x8f, 0x4e, 0x9b, 0x10, 0xc7, 0x54 | EFI\_DHCP4\_PROTOCOL.RenewRebind() - invokes **RenewRebind()** with the driver instance extending lease time in Asynchronous Mode using unicast.  (Calling functions in sequence A). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call EFI\_DHCP4\_PROTOCOL.RenewRebind()with a RebindRequest value of FALSEand a*CompletionEvent*value of not NULL. The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.4.5 | 0x432ccefe, 0x8586, 0x4358, 0xb7, 0xee, 0xf1, 0x36, 0xe3, 0x8a, 0xd8, 0x30 | EFI\_DHCP4\_PROTOCOL.RenewRebind() - invokes **RenewRebind()** with the driver instance extending lease time in Asynchronous Mode using unicast.  (Calling functions in sequence B). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call EFI\_DHCP4\_PROTOCOL.RenewRebind()with a RebindRequest value of FALSEand aCompletionEventvalue of not NULL. The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Bound.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.4.6 | 0xc0b17d39, 0x32bb, 0x41f8, 0xbd, 0x44, 0x6b, 0xb8, 0x53, 0x0f, 0xa4, 0xaf | EFI\_DHCP4\_PROTOCOL.RenewRebind() - invokes **RenewRebind()** with the driver instance extending lease time in Asynchronous Mode using broadcast.  (Calling functions in sequence A). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call EFI\_DHCP4\_PROTOCOL.RenewRebind()with a RebindRequest value of TRUEand aCompletionEventvalue of not NULL. The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.4.7 | 0x819f530e, 0x0d51, 0x43ce, 0x83, 0x73, 0x0b, 0x27, 0xc6, 0x36, 0x3b, 0x63 | EFI\_DHCP4\_PROTOCOL.RenewRebind() - invokes **RenewRebind()** with the driver instance extending lease time in Asynchronous Mode using broadcast.  Sequence B. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call EFI\_DHCP4\_PROTOCOL.RenewRebind()with a RebindRequest value of TRUEand aCompletionEventvalue of not NULL. The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Bound.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.4.8 | 0x982b5d48, 0x2d87, 0x40ea, 0xbe, 0x60, 0x44, 0x60, 0x49, 0xfe, 0x08, 0x98 | EFI\_DHCP4\_PROTOCOL.RenewRebind() - invokes **RenewRebind()** with the driver instance extending lease time in synchronous Mode using unicast.  Sequence A. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call EFI\_DHCP4\_PROTOCOL.RenewRebind()with a RebindRequest value of FALSEand aCompletionEventvalue of NULL. The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.4.9 | 0x4cc9abee, 0xd9e8, 0x444b, 0xb8, 0x34, 0x3e, 0xd4, 0x57, 0x96, 0x25, 0xc9 | EFI\_DHCP4\_PROTOCOL.RenewRebind() - invokes **RenewRebind()** with the driver instance extending lease time in synchronous Mode using unicast.  Sequence B. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a *CompletionEvent* value of not NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call EFI\_DHCP4\_PROTOCOL.RenewRebind() with a RebindRequest value of TRUEand aCompletionEventvalue of NULL. The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Bound.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.4.10 | 0x061ca38f, 0x5092, 0x483b, 0xa4, 0xd2, 0xf3, 0x1f, 0x53, 0x3f, 0xe7, 0xac | EFI\_DHCP4\_PROTOCOL.RenewRebind() - invokes **RenewRebind()** with the driver instance extending lease time in synchronous Mode using broadcast.  Sequence A. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call EFI\_DHCP4\_PROTOCOL.RenewRebind()with a RebindRequest value of FALSEand aCompletionEventvalue of NULL. The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.4.11 | 0xf9fa2078, 0x6283, 0x4510, 0xad, 0x21, 0xba, 0xe1, 0x15, 0x21, 0x56, 0xf9 | EFI\_DHCP4\_PROTOCOL.RenewRebind() - invokes **RenewRebind()** with the driver instance extending lease time in synchronous Mode using broadcast.  Sequence B. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a *CompletionEvent* value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call EFI\_DHCP4\_PROTOCOL.RenewRebind() with a RebindRequest value of TRUEand aCompletionEventvalue of not NULL. The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Bound.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |

### Release()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.2.5.1 | 0xa80fa204, 0x87dd, 0x4e92, 0x8a, 0x5d, 0xee, 0x55, 0x6c, 0x83, 0xac, 0x7c | EFI\_DHCP4\_PROTOCOL.Release() - invokes **Release()** with the driver in the configuration process, but not in the Dhcp4Bound or Dhcp4InitReboot state. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of not NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.Release()** after Stop the REQUEST packet from the driver. The return status should be **EFI\_ACCESS\_DENIED**.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.5.2 | 0x70f9485c, 0x4fef, 0x4bf3, 0xac, 0xd5, 0x2e, 0xe0, 0xba, 0x30, 0x3d, 0xd9 | EFI\_DHCP4\_PROTOCOL.Release() - invokes **Release()** with the driver in the Dhcp4Stopped state, but not in the Dhcp4Bound or Dhcp4InitReboot state. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Stop()** to verify the driver in the Dhcp4Stopped state.  3. Call **EFI\_DHCP4\_PROTOCOL.Release()** when the driver is in the Dhcp4Stopped state. The return status should be **EFI\_ACCESS\_DENIED**.  4. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.5.3 | 0x67c1be03, 0xf9c4, 0x4419, 0x88, 0xf0, 0xb9, 0xfc, 0x6c, 0x1a, 0xd2, 0x67 | EFI\_DHCP4\_PROTOCOL.Release() - invokes **Release()** when the driver is in the DhcpBound State**.** | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "0.0.0.0".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a CompletionEvent value of NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after the driver having stopped the DHCPACK packet. The ModeData.State should be Dhcp4Bound.  5. Call **EFI\_DHCP4\_PROTOCOL.Release()** and capture ARPREQUEST packet from the driver, send ARPREPLY packet to the driver, then capture DHCPRELEASE packet from the driver. The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.5.4 | 0x555d101b, 0xf86a, 0x4e6f, 0x95, 0x70, 0x1c, 0xfa, 0xe7, 0xd2, 0xd6, 0x8a | EFI\_DHCP4\_PROTOCOL.Release() - invokes **Release()** when the driver is in the DhcpInitReboot State**.** | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data. The ModeData.State should be DhcpInitReboot.  5. Call **EFI\_DHCP4\_PROTOCOL.Release()** and capture ARPREQUEST packet from the driver, send ARPREPLY packet to the driver, then capture DHCPRELEASE packet from the driver. The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after time out. The ModeData.State should be Dhcp4Init.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |

### 

### Stop()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.2.6.1 | 0xda8661a5, 0x82d4, 0x4b1b, 0xa2, 0x68, 0xf3, 0x4f, 0xe5, 0xab, 0x03, 0x57 | EFI\_DHCP4\_PROTOCOL.Stop() - invokes **Stop()** when the driver is in the DhcpInitReboot State**.** | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting *ClientAddress* "192.168.2.4".  3. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after receiving REQUEST packet from the driver. The ModeData.State should be DhcpInitReboot.  4. Call **EFI\_DHCP4\_PROTOCOL.Stop()** to stop the configuration process. The return status should be EFI\_SUCCESS.  5. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.6.2 | 0x0f6193fc, 0x21f7, 0x4831, 0xbf, 0x53, 0x39, 0x28, 0xc0, 0x49, 0x6b, 0x48 | EFI\_DHCP4\_PROTOCOL.Stop() - invokes **Stop()** when the driver is in the configuration process. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Configure()to configure child with setting ClientAddress "192.168.2.4".  3. Call EFI\_DHCP4\_PROTOCOL.Start() to start the configuration process with a *CompletionEvent* value other than NULL.  4. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data after receiving REQUEST packet from the driver. The ModeData.State should be Dhcp4Rebooting.  5. Call **EFI\_DHCP4\_PROTOCOL.Stop()** to stop the configuration process. The return status should be EFI\_SUCCESS.  6. Call **EFI\_DHCP4\_PROTOCOL.**GetModeData()to get Dhcp4 mode data. The ModeData.State should be Dhcp4Stopped.  7. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |

### Build()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.2.7.1 | 0xc2aa2960, 0xdd52, 0x4e56, 0x87, 0x7e, 0x8c, 0x44, 0x6a, 0x5e, 0xea, 0x31 | EFI\_DHCP4\_PROTOCOL.Build() - invokes Build()when the parameter SeedPacket is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Build() with a SeedPacket value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.7.2 | 0xf19cc8c3, 0x9a84, 0x4d62, 0x94, 0xae, 0xc3, 0x4b, 0x06, 0x3a, 0xea, 0x91 | EFI\_DHCP4\_PROTOCOL.Build() - invokes **Build()** when the parameter *SeedPacket* is not a well-formed DHCP packet (Magic Number Error). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Build() with a SeedPacket.EFI\_DHCP4\_PROTOCOL.Magik value of error magic cookie. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.7.3 | 0xc650067b, 0x4ab0, 0x4170, 0x9b, 0x4b, 0x4f, 0x7a, 0xeb, 0x77, 0xc0, 0x5e | EFI\_DHCP4\_PROTOCOL.Build() - invokes **Build()** when the parameter AppendCountis not 0 and AppendList is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Build() with a *AppendCount*value other than NULL and *AppendList* value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.7.4 | 0x1debfafe, 0xdfbe, 0x4ff5, 0x8a, 0xcd, 0x8f, 0xe1, 0x11, 0x82, 0x30, 0xe0 | EFI\_DHCP4\_PROTOCOL.Build() - invokes Build()when the parameter DeleteCount is not 0 and DeleteList is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Build() with a *DeleteCount* value of NULL and a *DeleteList* value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.7.5 | 0xd0beca24, 0xa8f3, 0x4753, 0x8c, 0xdb, 0x96, 0xe6, 0x00, 0x92, 0x78, 0x47 | EFI\_DHCP4\_PROTOCOL.Build() - invokes Build()when the parameter NewPacket is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Build() with a *NewPacket* value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.7.6 | 0x7d05c782, 0xccf3, 0x42d0, 0x9a, 0x6e, 0x0d, 0x6b, 0x5c, 0x8d, 0x9c, 0x20 | EFI\_DHCP4\_PROTOCOL.Build() - invokes Build()when the parameter both DeleteCount and OptionCount are 0 and NewPacket is not NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Build() with both the *DeleteCount* and *OptionCount* value of 0 and a *NewPacket* value other than NULL. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.7.7 | 0xf52d8032, 0xd5c6, 0x48e1, 0x86, 0xb0, 0xac, 0x47, 0xae, 0x82, 0x93, 0xed | EFI\_DHCP4\_PROTOCOL.Build() - invokes **Build()** when the parameter *AppendCount*and *AppendList* are not NULL, and build a new packet with DHCP options appended. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Build() with both the *AppendCount*and *AppendList* value other than NULL. The return status should be EFI\_SUCCESS.  3. Call **EFI\_DHCP4\_PROTOCOL.Parse()** to parse the packet returned by the parameter *NewPacket* of EFI\_DHCP4\_PROTOCOL.Build(). The *NewPacket* should include the DHCP options matching the parameter *AppendList* of EFI\_DHCP4\_PROTOCOL.Build().  4. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.7.8 | 0x78dae7e2, 0x579a, 0x47a1, 0xb2, 0x45, 0x8c, 0xad, 0x39, 0xc8, 0x07, 0x27 | EFI\_DHCP4\_PROTOCOL.Build() - invokes **Build()** to delete defined options. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Build() with both the *DeleteCount* and *DeleteList* value other than NULL. The return status should be EFI\_SUCCESS.  3. Call **EFI\_DHCP4\_PROTOCOL.Parse()** to parse the packet returned by the parameter *NewPacket* of EFI\_DHCP4\_PROTOCOL.Build(). The *NewPacket* should not include the DHCP options matching the parameter *DeleteList* of EFI\_DHCP4\_PROTOCOL.Build().  4. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.7.9 | 0xfc1f9cb7, 0xed3d, 0x4e6d, 0x93, 0x2a, 0x63, 0xb5, 0xcf, 0xb4, 0xb3, 0x37 | EFI\_DHCP4\_PROTOCOL.Build() - invokes **Build()** to delete an undefined option. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Build() with both the *DeleteCount* and *DeleteList* value other than NULL, and *DeleteList* include an undefined option. The return status should be EFI\_SUCCESS.  3. Call **EFI\_DHCP4\_PROTOCOL.Parse()** to parse the packet returned by the parameter *NewPacket* of EFI\_DHCP4\_PROTOCOL.Build(). The *NewPacket* should not include the DHCP options matching the parameter *DeleteList* of EFI\_DHCP4\_PROTOCOL.Build().  4. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |

### Transmit**Receive()**

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.2.8.1 | 0x6d1bb6a7, 0x5d67, 0x4982, 0x96, 0x35, 0x54, 0xeb, 0x4b, 0x0c, 0xfa, 0xd5 | EFI\_DHCP4\_PROTOCOL.TransmitReceive() - invokes TransmitReceive() when the parameter RemoteAddress is 0. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.TransmitReceive() with a RemoteAddress value of 0. The return status should be EFI\_UNSUPPORTED.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.8.2 | 0xd2bec02f, 0x8304, 0x4713, 0x8a, 0x95, 0x4b, 0xd3, 0x4c, 0x69, 0x89, 0xc0 | EFI\_DHCP4\_PROTOCOL.TransmitReceive() - invokes TransmitReceive() when the parameter *Packet* is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.TransmitReceive() with a Packet value of NULL. The return status should be EFI\_UNSUPPORTED.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.8.3 | 0x9dfd549b, 0x59eb, 0x4f5d, 0x99, 0x5f, 0xb8, 0x2d, 0xdd, 0x18, 0x02, 0xba | EFI\_DHCP4\_PROTOCOL.TransmitReceive() - invokes **Transmit**Receive() when the parameter *Packet* is not a well-formed DHCP packet(Magic Number error). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Transmit**Receive()with a *Packet* value of not a well-formed DHCP packet(Magic Number error). The return status should be **EFI\_UNSUPPORTE**.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.8.4 | 0xce99ae23, 0x910a, 0x4818, 0xa0, 0x89, 0xf3, 0xf4, 0x5b, 0xc5, 0xeb, 0xa8 | EFI\_DHCP4\_PROTOCOL.TransmitReceive() - invokes **Transmit**Receive() when the transaction ID in *Packet* is in used by another DHCP process. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Transmit**Receive()when the transaction ID in *Packet* is in use by another DHCP process. The return status should be **EFI\_UNSUPPORTED**.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.8.5 | 0xbe6683bd, 0x807a, 0x4fb0, 0xbc, 0x7b, 0xf7, 0x51, 0x07, 0x0e, 0x0e, 0x66 | EFI\_DHCP4\_PROTOCOL.TransmitReceive() - invokes **Transmit**Receive() when the previous call to this function has not finished yet. Try to call this function after collection process completed. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Transmit**Receive()with the previous call to this function not finished yet. The return status should be **EFI\_UNSUPPORTE**.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |

### Parse()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.2.9.1 | 0x7cca1a2c, 0x4136, 0x4ff0, 0xbc, 0x22, 0xca, 0x80, 0x56, 0x8d, 0xfd, 0xbf | EFI\_DHCP4\_PROTOCOL.Parse() - invokes Parse() when the parameter Packet is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Parse()with a Packet value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.9.2 | 0x225ddf1b, 0x9fb9, 0x4a9b, 0xb3, 0xb6, 0xca, 0x25, 0xeb, 0x31, 0x0d, 0xbb | EFI\_DHCP4\_PROTOCOL.Parse() - invokes Parse() when the parameter OptionCount is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Parse()with a OptionCount value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.9.3 | 0xea1a95dd, 0xdb6c, 0x4200, 0xb7, 0xc7, 0x19, 0xb0, 0xa3, 0x81, 0x06, 0x5d | EFI\_DHCP4\_PROTOCOL.Parse() - invokes **Parse()** when the Packet is not a well-formed DHCP packet (Magic Number error). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Parse()** with a *Packet* value other than a well-formed DHCP packet (Magic Number error). The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.9.4 | 0x91e4d243, 0x4ed6, 0x451a, 0xb0, 0x9c, 0x0a, 0x35, 0x6a, 0x06, 0x1d, 0xda | EFI\_DHCP4\_PROTOCOL.Parse() - invokes **Parse()** when the Packet is not well-formed DHCP packet (No End option). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Parse()** with a Packet value other than a well-formed DHCP packet (No End option). The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.9.5 | 0xd836cddd, 0x6bb4, 0x455e, 0x9e, 0xc4, 0x49, 0x9f, 0xc3, 0x27, 0xdd, 0x21 | EFI\_DHCP4\_PROTOCOL.Parse() - invokes **Parse()** when the Packet is not a well-formed DHCP packet (Length < Header Size). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Parse()** with a Packet value other than a well-formed DHCP packet (Length < Header Size). The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.9.6 | 0xed5c8f2b, 0x0043, 0x4f43, 0xae, 0x83, 0xa6, 0xbf, 0xab, 0x5b, 0xa2, 0xba | EFI\_DHCP4\_PROTOCOL.Parse() - invokes **Parse()** when the Packet is not a well-formed DHCP packet (Size < Length). | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Parse()** with a Packet value other than a well-formed DHCP packet (Size < Length). The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.9.7 | 0x4bd82a66, 0xcede, 0x4132, 0xa8, 0xca, 0xd9, 0x95, 0xe8, 0xe7, 0x9a, 0xb2 | EFI\_DHCP4\_PROTOCOL.Parse() - invokes **Parse()** when the parameter OptionCount is smaller than the number of options that were found in the Packet. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Parse()** with the parameter **OptionCount** smaller than the number of options that were found in the Packet. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.9.8 | 0xa73ac67a, 0xe5c9, 0x41e7, 0xb6, 0xc0, 0x80, 0xa2, 0x6f, 0x27, 0x7e, 0xc0 | EFI\_DHCP4\_PROTOCOL.Parse() - invokes Parse() when the parameter PacketOptionList is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call EFI\_DHCP4\_PROTOCOL.Parse()with a PacketOptionList value of NULL. The return status should be EFI\_INVALID\_PARAMETER.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.9.9 | 0xc84a412c, 0x702a, 0x40e1, 0xa3, 0x9c, 0x55, 0xa8, 0x8c, 0xbe, 0x60, 0x5a | EFI\_DHCP4\_PROTOCOL.Parse() - invokes **Parse()** when options exist in packet. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Parse()** to check the *PacketOptionList* when options exist in packet. The return status should be EFI\_SUCCESS.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.9.10 | 0x2ba25811, 0x4069, 0x45da, 0xb3, 0x9e, 0xfa, 0x05, 0x14, 0x42, 0x4a, 0x4c | EFI\_DHCP4\_PROTOCOL.Parse() - invokes **Parse()** when no options exist in packet | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Parse()** to check the *PacketOptionList* when no options exist in packet. The return status should be EFI\_SUCCESS.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |
| 5.24.2.9.11 | 0x6ce744e5, 0x9e5a, 0x4fb5, 0xa5, 0xf2, 0x3b, 0xe8, 0xf5, 0xb5, 0xad, 0x42 | EFI\_DHCP4\_PROTOCOL.Parse() - invokes **Parse()** with Pad Option included in packet | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_PROTOCOL.Parse()** to check the *PacketOptionList* with Pad Option included in packet. The return status should be EFI\_SUCCESS.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child and clean up the environment. |

### CreateChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.2.10.1 | 0x4b66733f, 0xd324, 0x4af9, 0x9d, 0x92, 0x91, 0x4f, 0x5f, 0x77, 0x2e, 0xf0 | EFI\_DHCP4\_PROTOCOL.CreateChild() - invokes CreateChild() when Child Handle is NULL. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child with NULL Handle Pointer. The return status should be **EFI\_INVALID\_PATAMETER**. |
| 5.24.2.10.2 | 0x1e0f5047, 0x1be9, 0x4db0, 0xa5, 0x71, 0xfc, 0x82, 0xbc, 0x2d, 0x0a, 0x06 | EFI\_DHCP4\_PROTOCOL.CreateChild() - to test the function of CreateChild(). | Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create childs three times and then destroy them. |

### DestroyChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.2.11.1 | 0x1f92470a, 0x7aec, 0x4fb4, 0xa4, 0x0d, 0x5f, 0x0c, 0xd2, 0x40, 0x1f, 0x08 | EFI\_DHCP4\_PROTOCOL.DestroyChild() – invokes DestroyChild() when Call this function twice. | 1. Call EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.CreateChild() to create a new Dhcp4 child.  2. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child. The return status should be EFI\_SUCCESS.  3. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the created Dhcp4 child again. The return status should be **EFI\_UNSUPPORTED**. |
| 5.24.2.11.2 | 0x06b43e55, 0xd8af, 0x494f, 0x8b, 0x93, 0x78, 0xf8, 0xd0, 0x7a, 0xa4, 0xc8 | EFI\_DHCP4\_PROTOCOL.DestroyChild() – invokes DestroyChild() when Child Handle is NULL. | 1. Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to Destroy a Dhcp4 child with NULL Handle Pointer. The return status should be **EFI\_INVALID\_PATAMETER**. |
| 5.24.2.11.3 | 0xc44a4b68, 0x1f16, 0x4098, 0xb2, 0x6d, 0x2c, 0x43, 0xcb, 0x27, 0x4d, 0xae | EFI\_DHCP4\_PROTOCOL.DestroyChild() – to test the function of DestroyChild(). | Call **EFI\_DHCP4\_SERVICE\_BINDING\_PROTOCOL.DestroyChild()** to destroy the newly three created Dhcp4 childs. |

## EFI\_DHCP6\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_DHCP6\_PROTOCOL Section.

### CreateChild()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.3.1.1 | 0xbd25610a, 0xa4b3, 0x412a, 0xbf, 0x03, 0xb0, 0xf7, 0xce, 0x80, 0x98, 0xbf | EFI\_DHCP6\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - CreateChild() returns EFI\_INVALID\_PARAMETER with a NULL ChildHandle. | Call CreateChild()with a NULL ChildHandle, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.1.2 | 0xcbf5cb1d, 0xd74d, 0x45bc, 0x94, 0xd2, 0x72, 0xda, 0x7f, 0xf7, 0xbe, 0xda | EFI\_DHCP6\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - CreateChild() returns EFI\_SUCCESS with the 1st validChildHandle. | 5.24.3.1.2 to 5.24.3.1.5 belong to one case.  1. Call CreateChild()with the 1st valid ChildHandle, the return status should be EFI\_SUCCESS. |
| 5.24.3.1.3 | 0xb9cfe63d, 0x2cc2, 0x4940, 0xb3, 0x01, 0x39, 0x22, 0xf3, 0xff, 0xdd, 0x35 | EFI\_DHCP6\_SERVICE\_BINDING\_PROTOCOL.CreateChild() - CreateChild() returns EFI\_SUCCESS with the 2nd validChildHandle. | 2. Call CreateChild()with the 2nd valid ChildHandle, the return status should be EFI\_SUCCESS. |
| 5.24.3.1.4 | 0x2336ebe8, 0x4934, 0x4a6c, 0xae, 0x72, 0x06, 0x73, 0xb6, 0x7a, 0xa0, 0xa6 | EFI\_DHCP6\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - DestroyChild() returns EFI\_SUCCESS with the 2nd validChildHandle. | 3. Call DestroyChild()with the 2nd valid ChildHandle, the return status should be EFI\_SUCCESS. |
| 5.24.3.1.5 | 0x0fe6555e, 0x3487, 0x4989, 0x89, 0x96, 0x18, 0xa7, 0x2a, 0x71, 0x52, 0xd5 | EFI\_DHCP6\_SERVICE\_BINDING\_PROTOCOL.DestroyChild() - DestroyChild() returns EFI\_SUCCESS with the 1st validChildHandle. | 4. Call DestroyChild()with the 1st valid ChildHandle, the return status should be EFI\_SUCCESS. |

### DestroyChild ()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.3.2.1 | 0x6e2206aa, 0xbee7, 0x4f16, 0xa7, 0xaa, 0x71, 0x54, 0xa2, 0xe9, 0x63, 0x65 | EFI\_DHCP6\_SERVICE\_BINDING\_PROTOCOL. DestroyChild() - DestroyChild() returns EFI\_INVALID\_PARAMETER with a NULL ChildHandle*.* | Call DestroyChild() with a NULL ChildHandle, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.2.2 | 0x061893a7, 0x48de, 0x431a, 0xad, 0x5b, 0x56, 0x29, 0xb6, 0x9c, 0xe6, 0xce | EFI\_DHCP6\_SERVICE\_BINDING\_PROTOCOL. DestroyChild() - DestroyChild() returns EFI\_UNSUPPORTED with a ChildHandlewhich has been destroyed. | Call DestroyChild() with a ChildHandlewhich has been destroyed, the return status should be EFI\_UNSUPPORTED. |

### GetModeData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.3.3.1 | 0x3678146a, 0x0596, 0x4661, 0x8e, 0x53, 0xf6, 0x61, 0xa6, 0xec, 0xe2, 0xf3 | EFI\_DHCP6 PROTOCOL.GetModeData() - GetModeData()returns EFI\_ACCESS\_DENIED with an instance which has not been configured. | Call GetModeData()with an instance which has not been configured, The return status should be EFI\_ACCESS\_DENIED. |
| 5.24.3.3.2 | 0xf58195a9, 0x1924, 0x4490, 0x95, 0x4b, 0x17, 0x75, 0xfc, 0x1c, 0xbf, 0xb0 | EFI\_ DHCP6 PROTOCOL.GetModeData() - GetModeData()returns EFI\_INVALID\_PARAMETER with **NULL** **Dhcp6ConfigData** and **Dhcp6ModeData** | Call GetModeData()with **NULL** **Dhcp6ConfigData** and **Dhcp6ModeData,** The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.3.3 | 0x99d01c9a, 0x2bd6, 0x442f, 0x8f, 0xe5, 0xda, 0x8a, 0xa6, 0x88, 0x27, 0x29 | Dhcp6CfgData.CallbackContext should be 5. | 5.24.3.3.3 to 5.24.3.1.13 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Create an event for the Dhcp6CfgData  3. Call Configure() to initialize the DHCP6 instance.  4. Call Start() to start S.A.R.R process.  5. The Dhcp6CfgData.CallbackContext should be 5. The reason is Callback() is called by SendSolict/RcvdAdvertise/SelectAdvertise/SendRequest/RcvdReply. Callback() add Dhcp6CfgData.CallbackContext with 1 each time. |
| 5.24.3.3.4 | 0x46993cb1, 0xfb2c, 0x44b3, 0xad, 0xe1, 0x7e, 0xa1, 0xe8, 0x43, 0xbd, 0x2e | Dhcp6CfgData.IaInfoEvent should be signaled. | 6. When Start() return, the Dhcp6CfgData.IaInfoEvent should be signaled. |
| 5.24.3.3.5 | 0x6a6bd40b, 0xb963, 0x4313, 0x8b, 0x4f, 0x45, 0x0e, 0x11, 0x4b, 0x6e, 0xeb | EFI\_ DHCP6 PROTOCOL.GetModeData() - GetModeData()returns EFI\_SUCCESS with **Dhcp6ConfigData** and **Dhcp6ModeData** | 7. Call GetModeData()with **Dhcp6ConfigData** and **Dhcp6ModeData,** The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.3.6 | 0x24694dfa, 0x5cc6, 0x4358, 0x9a, 0x14, 0x5c, 0xf2, 0x5e, 0x3a, 0x1a, 0xa4 | Dhcp6ModeData.Ia.State should be Dhcp6Bound | 8. Dhcp6ModeData.Ia.State should be Dhcp6Bound |
| 5.24.3.3.7 | 0x6a19ff82, 0x9ea9, 0x44c1, 0xb8, 0x71, 0x69, 0x05, 0xfe, 0x18, 0x58, 0xbc | **Dhcp6ConfigData.OptionCount** should be same with configured the value. | 9. **Dhcp6ConfigData.OptionCount** should be same with the configured value. |
| 5.24.3.3.8 | 0x9fa4ae6e, 0x82b3, 0x4ed7, 0xb9, 0xfc, 0x69, 0x0f, 0x0a, 0x98, 0xe5, 0xde | **Dhcp6ConfigData.OptionList** should be same with configured the value. | 10. **Dhcp6ConfigData.OptionList** should be same with the configured value. |
| 5.24.3.3.9 | 0xa803b115, 0x47b7, 0x496f, 0x95, 0xdb, 0x38, 0xf2, 0x3e, 0x27, 0x3c, 0x20 | **Dhcp6ConfigData.IaDescriptor** should be same with configured the value. | 11. **Dhcp6ConfigData.IaDescriptor** should be same with the configured value. |
| 5.24.3.3.10 | 0x2e4a61f7, 0x3a07, 0x4dd9, 0x8b, 0xf6, 0xc3, 0xef, 0xbb, 0x35, 0xb7, 0x90 | **Dhcp6ConfigData.** **IaInfoEvent** should be same with configured the value. | 12. **Dhcp6ConfigData.** **IaInfoEvent** should be same with the configured value. |
| 5.24.3.3.11 | 0x32797b99, 0x3b8b, 0x4456, 0x9d, 0xca, 0x3f, 0x76, 0xc6, 0x3f, 0x1c, 0xbf | **Dhcp6ConfigData.** **ReconfigureAccept** should be same with configured the value. | 13. **Dhcp6ConfigData.** **ReconfigureAccept** should be same with the configured value. |
| 5.24.3.3.12 | 0xb2f4a83b, 0xe44d, 0x4770, 0x81, 0xef, 0xef, 0x06, 0x29, 0xbd, 0x7f, 0xd7 | **Dhcp6ConfigData.** **RapidCommit** should be same with configured the value. | 14. **Dhcp6ConfigData.** **RapidCommit** should be same with the configured value. |
| 5.24.3.3.13 | 0x45ea153f, 0x2d5f, 0x40b4, 0xbd, 0x34, 0x04, 0x52, 0x27, 0xd9, 0xb5, 0xc3 | **Dhcp6ConfigData.SolicitRetransmission** should be same with configured the value. | 15. **Dhcp6ConfigData.SolicitRetransmission** should be same with the configured value. |

### Configure()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.3.4.1 | 0x8aa05b75, 0x4bdf, 0x45e6, 0x81, 0x74, 0x21, 0x85, 0x55, 0x88, 0x19, 0x74 | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER with **Dhcp6ConfigData.OptionCount** > 0 and **Dhcp6ConfigData. OptionList** is **NULL** | Call Configure() with **Dhcp6ConfigData.OptionCount** > 0 and **Dhcp6ConfigData. OptionList** is **NULL**, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.4.2 | 0xee84c2d5, 0xda69, 0x45ca, 0x9b, 0x62, 0x6c, 0x5f, 0x9a, 0xd9, 0x0d, 0xe2 | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER with **OptionList** containing ClientId option. | Call Configure() with **OptionList** containing ClientId option, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.4.3 | 0xd6cda19e, 0xcec6, 0x458a, 0xb9, 0xc7, 0x9d, 0x5e, 0xc8, 0x3d, 0xdd, 0x3f | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER with **OptionList** containing ReconfigAccept option. | Call Configure() with **OptionList** containing ReconfigAccept option, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.4.4 | 0x8a694b28, 0x7d56, 0x4171, 0xa9, 0x91, 0x07, 0x89, 0x56, 0x08, 0xf3, 0xb2 | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER with **OptionList** containing RapidCommit option. | Call Configure() with **OptionList** containing RapidCommit option, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.4.5 | 0x671c33eb, 0x66ab, 0x46db, 0xac, 0x12, 0xb6, 0x41, 0xca, 0xf3, 0xc2, 0xad | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER with **OptionList** containing IA for Non-temporary Addresses Option. | Call Configure() with **OptionList** containing IA for Non-temporary Addresses Option, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.4.6 | 0x438764a3, 0x3419, 0x48c1, 0xbc, 0xb6, 0xa7, 0x82, 0x21, 0xaf, 0x4d, 0xb7 | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER with **OptionList** containing IA for temporary Addresses Option. | Call Configure() with **OptionList** containing IA for temporary Addresses Option, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.4.7 | 0x4ae68d37, 0x1f81, 0x41a9, 0xbf, 0x5a, 0xf7, 0x5f, 0xd6, 0xcf, 0x04, 0x11 | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER with an invalid **IaDescriptor.Type** (neither **EFI\_DHCP6\_IA\_TYPE\_NA** nor **EFI\_DHCP6\_IA\_TYPE\_TA**). | Call Configure() with an invalid **IaDescriptor.Type** (neither **EFI\_DHCP6\_IA\_TYPE\_NA** nor **EFI\_DHCP6\_IA\_TYPE\_TA**), The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.4.8 | 0xffb74292, 0x6403, 0x4e09, 0xb3, 0x83, 0xe9, 0xa8, 0x14, 0x98, 0x54, 0xfa | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER with an **IaDescriptor** is not unique. | Call Configure() with an **IaDescriptor** is not unique, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.4.9 | 0x286b8508, 0x13bc, 0x44cc, 0xaa, 0x6a, 0xc2, 0xd9, 0xac, 0xc7, 0xeb, 0x49 | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER with both IaInfoEvent and SolicitRetransmission NULL. | Call Configure() with both IaInfoEvent and SolicitRetransmission NULL, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.4.10 | 0xc74fd682, 0x5e75, 0x455d, 0xbf, 0xc2, 0x28, 0xe0, 0xf3, 0x54, 0x34, 0xfa | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER with a non NULL SolicitRetransmission while Mrc and Mrd are zero. | Call Configure()with a non NULL Dhcp6ConfigData while Mrc and Mrd are zero, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.4.11 | 0x49935e3b, 0xe516, 0x423f, 0xa9, 0xb1, 0x99, 0x97, 0xea, 0xd4, 0x1c, 0x96 | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_ACCESS\_DENIED with a non NULL Dhcp6ConfigData while the instance has already been configured. | Call Configure()with a non NULL Dhcp6ConfigData while the instance has already been configured, The return status should be EFI\_ACCESS\_DENIED. |
| 5.24.3.4.12 | 0x59090898, 0x378c, 0x4555, 0xa6, 0xab, 0x14, 0x10, 0x96, 0xdc, 0x4f, 0xde | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_SUCCESS with a valid Dhcp6ConfigData | 5.24.3.4.12 to 5.24.3.4.15 belong to one case.  1. Call Configure()with a valid Dhcp6ConfigData, The return status should be EFI\_SUCCESS. |
| 5.24.3.4.13 | 0x568406ba, 0xa297, 0x4917, 0x8e, 0x7f, 0x77, 0xbb, 0x73, 0x6b, 0x53, 0xae | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_SUCCESS with a NULL Dhcp6ConfigData | 2. Call Configure()with a NULL Dhcp6ConfigData, The return status should be EFI\_SUCCESS. |
| 5.24.3.4.14 | 0x670d8a4d, 0x57e4, 0x424a, 0xbb, 0x72, 0x02, 0xb6, 0x72, 0xb0, 0x2d, 0x78 | Dhcp6ModeData.ClientId should not be 0. | 3. Call GetModeData()to get GetModeData.  4. Dhcp6ModeData.ClientId should not be 0. |
| 5.24.3.4.15 | 0x93080b8e, 0x5908, 0x4c54, 0x8d, 0xa7, 0xf6, 0x73, 0x2c, 0x66, 0x68, 0x92 | Dhcp6ModeData.Ia should be 0. | 5. Dhcp6ModeData.Ia should be 0. |

### Start()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.3.5.1 | 0x2153bcbb, 0xd5d3, 0x487e, 0x80, 0x98, 0xea, 0x02, 0x22, 0x79, 0x60, 0x11 | EFI\_ DHCP6 PROTOCOL.Start() - Start()returns EFI\_ACCESS\_DENIED with the non configured instance. | Call Start() with the non configured instance, The return status should be EFI\_ACCESS\_DENIED. |
| 5.24.3.5.2 | 0x5b1e8f26, 0x72e7, 0x429a, 0xbc, 0xbd, 0xff, 0xd0, 0x27, 0x91, 0x8a, 0x35 | EFI\_ DHCP6 PROTOCOL.Start() - Start()returns EFI\_ALREADY\_STARTED with the configured instance which has been started. | Call Start() with the configured instance which has been started, The return status should be EFI\_ALREADY\_STARTED. |
| 5.24.3.5.3 | 0xc5eca119, 0x7635, 0x4c13, 0x98, 0x5d, 0xde, 0xed, 0xf6, 0x94, 0x83, 0x37 | EFI\_ DHCP6 PROTOCOL.Start() - Start()returns EFI\_NO\_RESPONSE while DHCPv6 S.A.R.R process failed because of no response. | Call Start() while DHCPv6 S.A.R.R process failed because of no response, The return status should be EFI\_NO\_RESPONSE. |
| 5.24.3.5.4 | 0x23731450, 0xf84f, 0x43cc, 0xa6, 0x2a, 0x87, 0x6c, 0x10, 0xb7, 0xb2, 0x08 | EFI\_ DHCP6 PROTOCOL.Start() - Start()returns EFI\_ABORTED when the user returns error status from callback function. | Call Configure()when the user returns error status from callback function, The return status should be EFI\_ABORTED. |
| 5.24.3.5.5 | 0xd5a092e9, 0xed43, 0x4e5e, 0x8d, 0x9f, 0xc9, 0xc4, 0x92, 0x65, 0x27, 0xce | EFI\_ DHCP6 PROTOCOL.Start() - Start() returns EFI\_SUCCESS when the S.A.R.R process successfully. | 5.24.3.5.5 to 5.24.3.5.7 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Create an event for the Dhcp6CfgData  3. Call Configure() to initialize the DHCP6 instance.  4. Call Start() to start S.A.R.R process.  5. Get the return status of Start(), it should be EFI\_SUCCESS |
| 5.24.3.5.6 | 0xbb8655d9, 0x8d41, 0x452a, 0x92, 0x6e, 0xc8, 0xe7, 0x92, 0xf8, 0xc4, 0xcc | GetModeData.Ia.State should be Dhcp6Bound. | 6. Call GetModeData() to get the GetModeData  7. GetModeData.Ia.State should be Dhcp6Bound |
| 5.24.3.5.7 | 0xb7d13d3b, 0x6492, 0x4955, 0x9d, 0x51, 0xe0, 0xba, 0x96, 0x69, 0xfd, 0x43 | Dhcp6ConfigData.IaInfoEvent should be signaled. | 8. Dhcp6ConfigData.IaInfoEvent should be signaled |
| 5.24.3.5.8 | 0x6e3cc768, 0x1a9c, 0x466f, 0xa6, 0x0f, 0xac, 0xd4, 0x58, 0x76, 0xdb, 0x7f | EFI\_ DHCP6 PROTOCOL.Start() - Start() returns EFI\_SUCCESS when the S.A.R.R process successfully. | 5.24.3.5.8 to 5.24.3.5.9 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Call Configure() to initialize the DHCP6 instance.  3. Call Start() to start S.A.R.R process.  4. Get the return status of Start(), it should be EFI\_SUCCESS |
| 5.24.3.5.9 | 0xf68a6461, 0x26cf, 0x4f37, 0xa5, 0xd2, 0x65, 0xb2, 0x65, 0xd1, 0x1a, 0x84 | EFI\_ DHCP6 PROTOCOL.Configure() - Configure()returns EFI\_INVALID\_PARAMETER with both IaInfoEvent and SolicitRetransmission NULL. | 5. Call GetModeData() to get the GetModeData  6. GetModeData.Ia.State should be Dhcp6Bound |

### InfoRequest()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.3.6.1 | 0x5bf750bc, 0x349f, 0x4aa2, 0xa5, 0x9f, 0xfd, 0x09, 0xba, 0xf0, 0xcf, 0xc1 | EFI\_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest()returns EFI\_INVALID\_PARAMETER with NULL OptionRequest. | Call InfoRequest() with NULL OptionRequest, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.6.2 | 0x3e90fc45, 0x7a27, 0x4c9b, 0x88, 0x8b, 0xfc, 0xa8, 0x56, 0x9f, 0x80, 0xef | EFI\_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest()returns EFI\_INVALID\_PARAMETER with non zero OptionCount and an NULL OptionList. | Call InfoRequest() with non zero OptionCount and an NULL OptionList, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.6.3 | 0xa85f59d4, 0x3a09, 0x4a74, 0xa8, 0xd6, 0x71, 0xee, 0x08, 0x20, 0x2f, 0x7e | EFI\_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest()returns EFI\_INVALID\_PARAMETER when OptionList contains client identity option. | Call InfoRequest() when OptionList contains client identity option, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.6.4 | 0x8647418d, 0xb3f9, 0x4bf5, 0xb5, 0x24, 0xf4, 0xc1, 0x7d, 0x36, 0x00, 0x20 | EFI\_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest()returns EFI\_INVALID\_PARAMETER with an NULL Retransmission. | Call InfoRequest() with an NULL Retransmission, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.6.5 | 0xf18e8693, 0xd00f, 0x497f, 0x86, 0xfe, 0xf9, 0x3a, 0x2f, 0x50, 0x38, 0x04 | EFI\_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest()returns EFI\_INVALID\_PARAMETER when both Retransmission.Mrd and Retransmission.Mrt are zero. | Call InfoRequest() when both Retransmission.Mrd and Retransmission.Mrt are zero, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.6.6 | 0x1669a032, 0x433a, 0x4dbc, 0x8c, 0x00, 0x81, 0xc4, 0xb6, 0x59, 0x78, 0x1f | EFI\_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest()returns EFI\_INVALID\_PARAMETER when ReplyCallback is NULL. | Call InfoRequest() when ReplyCallback is NULL, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.6.7 | 0xaa884b5b, 0xb369, 0x46cc, 0x85, 0xa9, 0xfe, 0xb0, 0x33, 0xd1, 0xaa, 0x48 | EFI\_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest()returns EFI\_NO\_RESPONSE when Dhcp6 server doesn’t response. | Call InfoRequest() when Dhcp6 server doesn’t response, The return status should be EFI\_NO\_RESPONSE. |
| 5.24.3.6.8 | 0x3ade8458, 0xd07a, 0x4f45, 0xbc, 0xc3, 0x49, 0x68, 0x20, 0xe9, 0x85, 0x0b | EFI\_ DHCP6 PROTOCOL.InfoRequest() - InfoRequest()returns EFI\_ABORTED when the user returns error status from ReplyCallback function. | Call InfoRequest()when the user returns error status from ReplyCallback function, The return status should be EFI\_ABORTED. |
| 5.24.3.6.9 | 0xc7cb2c53, 0xd008, 0x40b5, 0xb0, 0x53, 0xb2, 0x68, 0x08, 0xb8, 0x81, 0x3a | InfoRequestPacket should be received. | 5.24.3.6.9 to 5.24.3.6.12 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Create a timeout event.  3. Call InfoRequest() to obtain configuration information without ant IA address.  4. InfoRequestPacket should be received. |
| 5.24.3.6.10 | 0x730310e5, 0x5df3, 0x41f9, 0xbf, 0x4a, 0x75, 0x1b, 0x01, 0xf9, 0x59, 0xef | The return status of InfoRequest() should be EFI\_SUCCESS | 5. Send the the Reply packet for the InfoRequest message.  6. The return status of InfoRequest() should be EFI\_SUCCESS |
| 5.24.3.6.11 | 0x1cb6efc5, 0x1d58, 0x4c8e, 0xb5, 0x7d, 0x83, 0x7d, 0xd2, 0x8c, 0xb0, 0xd3 | The CallbackContext should be updated with ReplyCallback() | 7. The CallbackContext should be updated with ReplyCallback() |
| 5.24.3.6.12 | 0x5738bba8, 0xf1ad, 0x4889, 0x87, 0xed, 0x29, 0x21, 0x59, 0x17, 0x61, 0x48 | The Timeout event should not be signaled. | 8. The Timeout event should not be signaled. |
| 5.24.3.6.13 | 0xa0995b80, 0x76ad, 0x4d99, 0xa5, 0xd3, 0x0d, 0x55, 0x1d, 0xb0, 0x94, 0x75 | InfoRequestPacket should be received. | 5.24.3.6.13 to 5.24.3.6.15 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Call InfoRequest() to obtain configuration information without ant IA address.  3. InfoRequestPacket should be received. |
| 5.24.3.6.14 | 0x46a40db0, 0x5b97, 0x4272, 0x98, 0x98, 0x9c, 0xbb, 0xe7, 0xa2, 0x22, 0x5f | The return status of InfoRequest() should be EFI\_SUCCESS | 4. Send the the Reply packet for the InfoRequest message.  5. The return status of InfoRequest() should be EFI\_SUCCESS |
| 5.24.3.6.15 | 0x4b1612fa, 0x7561, 0x4b55, 0xb9, 0xa2, 0x76, 0x40, 0x02, 0xc6, 0x95, 0xe1 | The CallbackContext should be updated with ReplyCallback() | 6. The CallbackContext should be updated with ReplyCallback() |

### RenewRebind()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.3.7.1 | 0x613614f9, 0x2c96, 0x45ee, 0xad, 0xb8, 0xf0, 0x88, 0x72, 0xfd, 0x86, 0xf9 | EFI\_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()returns EFI\_ACCESS\_DENIED when the instance has not been configured. | Call RenewRebind() when the instance has not been configured, The return status should be EFI\_ACCESS\_DENIED. |
| 5.24.3.7.2 | 0x28ce0a5d, 0x6f3d, 0x47ad, 0xb1, 0x95, 0xc2, 0x5f, 0xce, 0xd8, 0x98, 0xb5 | EFI\_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()returns EFI\_ACCESS\_DENIED when the instance is not in Dhcp6Bound state. | Call RenewRebind() when the instance is not in Dhcp6Bound state, The return status should be EFI\_ACCESS\_DENIED. |
| 5.24.3.7.3 | 0x5c85dc0c, 0x634a, 0x4db3, 0x95, 0x81, 0x72, 0x0d, 0x1b, 0xda, 0x6c, 0x84 | EFI\_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()returns EFI\_ALREADY\_STARTED with RebindRequest TRUE when the instance in Dhcp6Rebinding state. | Call RenewRebind() with RebindRequest TRUE when the instance in Dhcp6Rebinding state, The return status should be EFI\_ALREADY\_STARTED. |
| 5.24.3.7.4 | 0x94bc77a0, 0xb016, 0x4d71, 0x8f, 0x5b, 0xd0, 0x49, 0x1a, 0x2c, 0x4f, 0x0c | EFI\_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()returns EFI\_ALREADY\_STARTED with RebindRequest FALSE when the instance in Dhcp6Rebinding state. | Call RenewRebind() with RebindRequest FALSE when the instance in Dhcp6Rebinding state, The return status should be EFI\_ALREADY\_STARTED. |
| 5.24.3.7.5 | 0xcc0b1c38, 0x2b99, 0x4ef4, 0xb9, 0x35, 0x63, 0x2e, 0x12, 0x46, 0x4f, 0xf7 | EFI\_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()returns EFI\_ABORTED when the user returns error status from callback function. | Call RenewRebind() when the user returns error status from callback function, The return status should be EFI\_ABORTED. |
| 5.24.3.7.6 | 0x2957725b, 0x7693, 0x40ac, 0xae, 0x81, 0x59, 0x54, 0x88, 0x25, 0xf7, 0x48 | EFI\_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()returns EFI\_SUCCESS when the exchange process is executed successfully. | 5.24.3.7.6 to 5.24.3.7.8 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Call Configure() to initialize the DHCP6 instance.  3. Call Start() to start S.A.R.R process.  4. Get the return status of Start(), it should be EFI\_SUCCESS  5. Call GetModeData() to get the GetModeData  6. GetModeData.Ia.State should be Dhcp6Bound  7. Call RenewRebind() and execute exchange process, including RENEW-REPLY, the return status should be EFI\_SUCCESS |
| 5.24.3.7.7 | 0xf495e992, 0xe807, 0x4a38, 0xbf, 0x42, 0x57, 0x1d, 0xd1, 0xfe, 0x8f, 0xc7 | CallbackContext should updated. | 8. CallbackContext should be updated. |
| 5.24.3.7.8 | 0x23d22d31, 0x1852, 0x4527, 0x80, 0x73, 0xcf, 0x8a, 0x51, 0x16, 0xff, 0x92 | The state is still Dhcp6Bound. | 9. The state is still Dhcp6Bound |
| 5.24.3.7.9 | 0x6ae394d7, 0xa5dc, 0x4147, 0x93, 0x5e, 0xf5, 0x07, 0xb2, 0xb8, 0xea, 0x35 | EFI\_ DHCP6 PROTOCOL.RenewRebind() - RenewRebind()returns EFI\_SUCCESS when the exchange process is executed successfully. | 5.24.3.7.9 to 5.24.3.7.10 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Call Configure() to initialize the DHCP6 instance.  3. Call Start() to start S.A.R.R process.  4. Get the return status of Start(), it should be EFI\_SUCCESS  5. Call GetModeData() to get the GetModeData  6. GetModeData.Ia.State should be Dhcp6Bound  7. Call RenewRebind() and execute exchange process, including RENEW-REBIND-REPLY, the return status should be EFI\_SUCCESS |
| 5.24.3.7.10 | 0x9f653dd2, 0x3edd, 0x47d6, 0xa6, 0x2e, 0x6c, 0x79, 0x99, 0x3d, 0xd9, 0x58 | The state is still Dhcp6Bound. | 8. The state is still Dhcp6Bound. |

### Decline()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.3.8.1 | 0x30c90eee, 0x69f1, 0x4a41, 0x88, 0x4d, 0x27, 0x6e, 0x9f, 0x6c, 0x0e, 0x33 | EFI\_ DHCP6 PROTOCOL.Decline() - Decline()returns EFI\_ACCESS\_DENIED when the instance has not been configured. | Call Decline() when the instance has not been configured, The return status should be EFI\_ACCESS\_DENIED. |
| 5.24.3.8.2 | 0x2f3cd8a1, 0x8987, 0x434d, 0xa1, 0xbb, 0xfc, 0xb6, 0x83, 0x04, 0xf6, 0x0d | EFI\_ DHCP6 PROTOCOL.Decline() - Decline()returns EFI\_ACCESS\_DENIED when the instance is not in Dhcp6Bound state. | Call Decline() when the instance is not in Dhcp6Bound state, The return status should be EFI\_ACCESS\_DENIED. |
| 5.24.3.8.3 | 0x6224a781, 0xfa3a, 0x4190, 0xa4, 0xfa, 0x5b, 0xec, 0x33, 0xbf, 0x3f, 0xfc | EFI\_ DHCP6 PROTOCOL.Decline() - Decline()returns EFI\_INVALID\_PARAMETER when the AddressCount is zero. | Call Decline() when the AddressCount is zero, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.8.4 | 0x1c8166c0, 0xbc5e, 0x4d1f, 0xa3, 0x8b, 0x65, 0x34, 0x7e, 0x76, 0x10, 0x69 | EFI\_ DHCP6 PROTOCOL.Decline() - Decline()returns EFI\_INVALID\_PARAMETER when the Addresses is NULL. | Call Decline() when the Addresses is NULL, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.8.5 | 0xc14f0d80, 0xe7e5, 0x4742, 0x9c, 0xc5, 0x27, 0xd0, 0x37, 0x79, 0x1b, 0x0d | EFI\_ DHCP6 PROTOCOL.Decline() - Decline()returns EFI\_NOT\_FOUND when any specified address in Addresses is not correlated with the configured IA. | Call Decline() when any specified address in Addresses is not correlated with the configured IA, The return status should be EFI\_NOT\_FOUND. |
| 5.24.3.8.6 | 0x44b4fcda, 0xf970, 0x4f3e, 0x88, 0xbb, 0x52, 0xf2, 0x52, 0xe9, 0x81, 0xdf | EFI\_ DHCP6 PROTOCOL.Decline() - Decline()returns EFI\_ABORTED when the user returns error status from callback function. | Call Decline() when the user returns error status from callback function, The return status should be EFI\_ABORTED. |
| 5.24.3.8.7 | 0x86606604, 0x5e2b, 0x4268, 0x91, 0xcd, 0x99, 0xc6, 0xb5, 0x7a, 0x42, 0xd8 | EFI\_ DHCP6 PROTOCOL.Decline() - Decline()returns EFI\_SUCCESS with execute exchange process, including DECLINE- REPLY. | 5.24.3.8.7 to 5.24.3.8.8 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Call Configure() to initialize the DHCP6 instance.  3. Call Start() to start S.A.R.R process.  4. Get the return status of Start(), it should be EFI\_SUCCESS  5. The CallbackContext is updated  Call GetModeData() to get the GetModeData  6. GetModeData.Ia.State should be Dhcp6Bound  7. Call Decline() and execute exchange process, including DECLINE- REPLY, the return status should be EFI\_SUCCESS |
| 5.24.3.8.8 | 0x1119b246, 0x8627, 0x45a1, 0x87, 0x89, 0x5b, 0xba, 0x7b, 0x4c, 0x0b, 0x48 | The state is still Dhcp6Bound. | 8. The state is still Dhcp6Bound |
| 5.24.3.8.9 | 0x554529cc, 0x30e2, 0x4269, 0x88, 0xb7, 0x72, 0x8e, 0x31, 0x1d, 0xbd, 0x1b | EFI\_ DHCP6 PROTOCOL.Decline() - Decline()returns EFI\_SUCCESS to decline all IP6 addresses of the configured IA and execute exchange process, including DECLINE- REPLY. | 5.24.3.8.9 to 5.24.3.8.10 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Call Configure() to initialize the DHCP6 instance.  3. Call Start() to start S.A.R.R process.  4. Get the return status of Start(), it should be EFI\_SUCCESS  5. The CallbackContext is updated  Call GetModeData() to get the GetModeData  6. GetModeData.Ia.State should be Dhcp6Bound  7. Call Decline() to decline all IP6 addresses of the configured IA and execute exchange process, including DECLINE- REPLY, the return status should be EFI\_SUCCESS |
| 5.24.3.8.10 | 0xf7449f19, 0x53e0, 0x4130, 0xba, 0x62, 0xea, 0x2b, 0x1f, 0x74, 0x8c, 0xa0 | The state is still Dhcp6Init. | 8. The state is still Dhcp6Init. |
| 5.24.3.8.11 | 0xcdbd802e, 0x7647, 0x41bc, 0x9b, 0xe6, 0xe4, 0x11, 0x9f, 0x6c, 0x79, 0x2d | EFI\_ DHCP6 PROTOCOL.Decline() - Decline()returns EFI\_SUCCESS to decline all IP6 addresses of the configured IA and execute exchange process, including DECLINE- REPLY. | 5.24.3.8.11 to 5.24.3.8.13 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Create IaInfoEvent  3. Call Configure() to initialize the DHCP6 instance.  4. Call Start() to start S.A.R.R process.  5. Get the return status of Start(), it should be EFI\_SUCCESS  6. The CallbackContext is updated  Call GetModeData() to get the GetModeData  7. GetModeData.Ia.State should be Dhcp6Bound  8. Call Decline() to decline all IP6 addresses of the configured IA and execute exchange process, including DECLINE- REPLY, the return status should be EFI\_SUCCESS |
| 5.24.3.8.12 | 0xfce31eb4, 0xeb16, 0x4b22, 0xb3, 0x55, 0xa8, 0xb0, 0x82, 0x0f, 0x0d, 0x3d | After the Decline exchange process returns,the IaInfoEvent will be signaled. | 9. After the Decline exchange process returns,the IaInfoEvent will be signaled. |
| 5.24.3.8.13 | 0x313da4fc, 0xf2ce, 0x4ecc, 0xa9, 0x97, 0x03, 0xea, 0x77, 0xfb, 0xdb, 0x59 | The state is still Dhcp6Init. | 10. The state is still Dhcp6Init. |
| 5.24.3.8.14 | 0x60c90ab2, 0x4372, 0x4b75, 0x84, 0x56, 0xe6, 0xe1, 0xfa, 0x34, 0x71, 0xad | EFI\_ DHCP6 PROTOCOL.Decline() - Decline()returns EFI\_NO\_RESPONSE to decline all IP6 addresses of the configured IA without the response from server. | 5.24.3.8.14 to 5.24.3.8.15 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Call Configure() to initialize the DHCP6 instance.  3. Call Start() to start S.A.R.R process.  4. Get the return status of Start(), it should be EFI\_SUCCESS  5. The CallbackContext is updated  Call GetModeData() to get the GetModeData  6. GetModeData.Ia.State should be Dhcp6Bound  7. Call Decline() to decline all IP6 addresses of the configured IA without the response from server, the return status should be EFI\_NO\_RESPONSE |
| 5.24.3.8.15 | 0x6af27ff2, 0xecb2, 0x4e96, 0xaf, 0xf7, 0xa7, 0x6b, 0x18, 0xe6, 0x38, 0xfa | The state is still Dhcp6Init. | 8. The state is still Dhcp6Init. |

### Release()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.3.9.1 | 0xfd3f1c62, 0x37d9, 0x4f34, 0x85, 0xe5, 0x93, 0x85, 0x28, 0x2f, 0xd3, 0xc4 | EFI\_ DHCP6 PROTOCOL.Release() - Release()returns EFI\_ACCESS\_DENIED when the instance has not been configured. | Call Release() when the instance has not been configured, The return status should be EFI\_ACCESS\_DENIED. |
| 5.24.3.9.2 | 0x38bc0e62, 0x4d8f, 0x4706, 0xb1, 0x39, 0xe0, 0xa7, 0x1c, 0xbd, 0x6d, 0x56 | EFI\_ DHCP6 PROTOCOL.Release() - Release()returns EFI\_ACCESS\_DENIED when the instance is not in Dhcp6Bound state. | Call Release() when the instance is not in Dhcp6Bound state, The return status should be EFI\_ACCESS\_DENIED. |
| 5.24.3.9.3 | 0x8e214193, 0x3dfb, 0x48e3, 0xb6, 0xe3, 0xdb, 0x4b, 0xde, 0xa4, 0xbc, 0xef | EFI\_ DHCP6 PROTOCOL.Release() - Release()returns EFI\_INVALID\_PARAMETER when the AddressCount is not zero and Addresses is NULL. | Call Release() when the AddressCount is not zero and Addresses is NULL, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.9.4 | 0x4b411cb3, 0x2427, 0x4315, 0xa3, 0x74, 0xa9, 0xdd, 0x29, 0xf7, 0x9a, 0xed | EFI\_ DHCP6 PROTOCOL.Release() - Release()returns EFI\_NOT\_FOUND when any specified address in Addresses is not correlated with the configured IA. | Call Release() when any specified address in Addresses is not correlated with the configured IA, The return status should be EFI\_NOT\_FOUND. |
| 5.24.3.9.5 | 0xa4b55b0e, 0x1037, 0x4717, 0x83, 0x53, 0x29, 0x24, 0xd3, 0x18, 0x23, 0x5d | EFI\_ DHCP6 PROTOCOL.Release() - Release()returns EFI\_ABORTED when the user returns error status from callback function. | Call Release() when the user returns error status from callback function, The return status should be EFI\_ABORTED. |
| 5.24.3.9.6 | 0x1459bb4e, 0xa926, 0x42cc, 0x99, 0x7d, 0xf8, 0x87, 0xf7, 0xd0, 0xbb, 0x71 | EFI\_ DHCP6 PROTOCOL.Release() - Release()returns EFI\_SUCCESS to release one of the IPv6 address that has already been assigned to the configured IA. | 5.24.3.9.6 to 5.24.3.9.7 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Call Configure() to initialize the DHCP6 instance.  3. Call Start() to start S.A.R.R process.  4. Get the return status of Start(), it should be EFI\_SUCCESS  5. The CallbackContext is updated  Call GetModeData() to get the GetModeData  6. GetModeData.Ia.State should be Dhcp6Bound  7. Call Release() to release one of the IPv6 address that has already been assigned to the configured IA, the return status should be EFI\_SUCCESS |
| 5.24.3.9.7 | 0x7251daef, 0x57ae, 0x4fc6, 0x81, 0xf4, 0x10, 0xe2, 0x34, 0xa5, 0x87, 0xa4 | The state is still Dhcp6Bound. | 8. The state is still Dhcp6Bound |
| 5.24.3.9.8 | 0x692e0cfb, 0x587d, 0x4906, 0x91, 0xa1, 0xcb, 0x20, 0x3b, 0x1e, 0xba, 0x2d | EFI\_ DHCP6 PROTOCOL.Decline() - Decline()returns EFI\_SUCCESS to release all IP6 addresses of the configured IA and execute exchange process, including RELEASE- REPLY. | 5.24.3.9.8 to 5.24.3.9.9 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Call Configure() to initialize the DHCP6 instance.  3. Call Start() to start S.A.R.R process.  4. Get the return status of Start(), it should be EFI\_SUCCESS  5. The CallbackContext is updated  Call GetModeData() to get the GetModeData  6. GetModeData.Ia.State should be Dhcp6Bound  7. Call Release() to release all IP6 addresses of the configured IA and execute exchange process, including RELEASE- REPLY, the return status should be EFI\_SUCCESS |
| 5.24.3.9.9 | 0x309de757, 0x2ab4, 0x4d5b, 0xb3, 0x7c, 0xb7, 0xdc, 0x46, 0x40, 0x4d, 0x1c | The state is still Dhcp6Init. | 8. The state is still Dhcp6Init. |
| 5.24.3.9.10 | 0x7b131129, 0x2fdb, 0x4a67, 0x8f, 0xaa, 0xe9, 0x0c, 0x1d, 0x08, 0xab, 0x94 | EFI\_ DHCP6 PROTOCOL.Release() - Release()returns EFI\_SUCCESS to release all IP6 addresses of the configured IA and execute exchange process, including RELEASE- REPLY. | 5.24.3.9.10 to 5.24.3.9.12 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Create IaInfoEvent  3. Call Configure() to initialize the DHCP6 instance.  4. Call Start() to start S.A.R.R process.  5. Get the return status of Start(), it should be EFI\_SUCCESS  6. The CallbackContext is updated  Call GetModeData() to get the GetModeData  7. GetModeData.Ia.State should be Dhcp6Bound  8. Call Release() to release all IP6 addresses of the configured IA and execute exchange process, including RELEASE- REPLY, the return status should be EFI\_SUCCESS |
| 5.24.3.9.11 | 0x47d072fd, 0x5782, 0x413b, 0xb4, 0x62, 0xb3, 0x18, 0x58, 0x04, 0xad, 0x4e | After the Release exchange process returns,the IaInfoEvent will be signaled. | 9. After the Release exchange process returns, the IaInfoEvent will be signaled. |
| 5.24.3.9.12 | 0x22dc90e4, 0xd93c, 0x465d, 0x90, 0x27, 0x35, 0xe9, 0xab, 0x3f, 0x3a, 0x3a | The state is still Dhcp6Init. | 10. The state is still Dhcp6Init. |
| 5.24.3.9.13 | 0x52b03918, 0x1e8c, 0x4620, 0xa1, 0x44, 0x02, 0x09, 0xae, 0xf3, 0xc7, 0x9d | EFI\_ DHCP6 PROTOCOL.Release() - Release()returns EFI\_NO\_RESPONSE to release all IP6 addresses of the configured IA without the response from server. | 5.24.3.9.14 to 5.24.3.9.15 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Call Configure() to initialize the DHCP6 instance.  3. Call Start() to start S.A.R.R process.  4. Get the return status of Start(), it should be EFI\_SUCCESS  5. The CallbackContext is updated  Call GetModeData() to get the GetModeData  6. GetModeData.Ia.State should be Dhcp6Bound  7. Call Release() to release all IP6 addresses of the configured IA without the response from server, the return status should be EFI\_NO\_RESPONSE |
| 5.24.3.9.14 | 0xc65a96c1, 0x448c, 0x4d75, 0x81, 0x90, 0x19, 0x13, 0x76, 0x1e, 0x79, 0x3d | The state is still Dhcp6Init. | 8. The state is still Dhcp6Init. |

### Stop()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.3.10.1 | 0x592d9e8d, 0x82cd, 0x44d8, 0xbf, 0x26, 0x0b, 0x40, 0x81, 0x25, 0x65, 0x17 | EFI\_ DHCP6 PROTOCOL.Stop() - Stop()returns EFI\_SUCCESS when the instance has not been configured. | Call Stop() when the instance has not been configured, The return status should be EFI\_SUCCESS. |
| 5.24.3.10.2 | 0x69ac94c1, 0xb57f, 0x4251, 0xb9, 0x56, 0x20, 0xaa, 0x9f, 0x30, 0x0d, 0xc1 | EFI\_ DHCP6 PROTOCOL.Stop() - Stop()returns EFI\_SUCCESS when the instance has been configured. | Call Stop() when the instance has been configured, The return status should be EFI\_SUCCESS. |
| 5.24.3.10.3 | 0x51255767, 0x7218, 0x400d, 0xa2, 0xd7, 0x3f, 0x3e, 0x50, 0x8c, 0x90, 0x64 | EFI\_ DHCP6 PROTOCOL.Release() - Release()returns EFI\_INVALID\_PARAMETER when the AddressCount is not zero and Addresses is NULL. | 5.24.3.9.3 to 5.24.3.9.5 belong to one case.  1. Call CreateChild() to create an DHCP6 instance.  2. Create IaInfoEvent  3. Call Configure() to initialize the DHCP6 instance.  4. Call Start() to start S.A.R.R process.  5. Get the return status of Start(), it should be EFI\_SUCCESS  6. The CallbackContext is updated  Call GetModeData() to get the GetModeData  7. GetModeData.Ia.State should be Dhcp6Bound  8. IaInfoEvent should be signaled.  9. Call Stop() to stop all IP6 addresses of the configured IA and execute exchange process, including RELEASE- REPLY, the return status should be EFI\_SUCCESS |
| 5.24.3.10.4 | 0xd00b1578, 0x5f23, 0x4ab7, 0x99, 0x40, 0x98, 0x51, 0x8a, 0x30, 0x8c, 0x08 | IaInfoEvent should be signaled. | IaInfoEvent should be signaled. |
| 5.24.3.10.5 | 0xcfa8dc36, 0xc246, 0x45d7, 0x94, 0xf1, 0xc9, 0x18, 0x54, 0xd6, 0x38, 0xad | The state of IA should be Dhcp6Init. | The state of IA should be Dhcp6Init |

### Parse()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.24.3.11.1 | 0x15a7d1de, 0x4bf6, 0x4507, 0xa3, 0xe2, 0xa1, 0xa4, 0x2e, 0xdd, 0x43, 0x23 | EFI\_ DHCP6 PROTOCOL.Parse() - Parse()returns EFI\_INVALID\_PARAMETER when the Packet is NULL. | Call Parse() when the Packet is NULL, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.11.2 | 0x28a7d965, 0x82bf, 0x49c6, 0xb1, 0xd8, 0x56, 0x08, 0x37, 0x0b, 0xdd, 0x62 | EFI\_ DHCP6 PROTOCOL.Parse() - Parse()returns EFI\_INVALID\_PARAMETER when the Packet is not well-formed(length is too small). | Call Parse() when the Packet is not well-formed(length is too small), The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.11.3 | 0x2228cc36, 0xa56b, 0x4aa8, 0xa2, 0x15, 0x06, 0x01, 0xce, 0xfe, 0x00, 0x94 | EFI\_ DHCP6 PROTOCOL.Parse() - Parse()returns EFI\_INVALID\_PARAMETER when the OptionCount is not zero and PacketOptionList is NULL. | Call Parse() when the OptionCount is not zero and PacketOptionList is NULL, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.11.4 | 0x444b0ef0, 0x0297, 0x4805, 0x8b, 0x2a, 0xc4, 0xa2, 0xf8, 0x82, 0xac, 0x2c | EFI\_ DHCP6 PROTOCOL.Parse() - Parse()returns EFI\_INVALID\_PARAMETER when the OptionCount is NULL. | Call Parse() when the OptionCount is NULL, The return status should be EFI\_INVALID\_PARAMETER. |
| 5.24.3.11.5 | 0x49182e78, 0x34dc, 0x4450, 0xb6, 0x2c, 0xfe, 0x28, 0x33, 0x51, 0xc1, 0x96 | EFI\_ DHCP6 PROTOCOL.Parse() - Parse()returns EFI\_BUFFER\_TOO\_SMALL when the OptionCount is NULL. | Call Parse() when OptionCount is smaller than the number of option that were found in the Packet, The return status should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.24.3.11.6 | 0x43dcf866, 0x9f05, 0x47d5, 0x92, 0xa1, 0x1e, 0x6f, 0x26, 0xf4, 0x1f, 0x61 | OptionCount should be update to the right number of option that is found in the packet. | OptionCount should be update to the right number of option that is found in the packet. |
| 5.24.3.11.7 | 0xacfb1bb7, 0x7b28, 0x4c35, 0xbd, 0x9f, 0x7e, 0x89, 0xa1, 0x9e, 0x54, 0xe2 | EFI\_ DHCP6 PROTOCOL.Parse() - Parse()returns EFI\_SUCCESS with the valid parameters. | Call Parse() with the valid parameters, The return status should be EFI\_SUCCESS. |
| 5.24.3.11.8 | 0xbb477381, 0x7731, 0x4259, 0x87, 0x01, 0xca, 0x1f, 0x71, 0xd6, 0xf9, 0x7e | The OpCode should be retrieved correctly. | The OpCode should be retrieved correctly. |