# Protocols Bootable Image Support Test

## EFI\_LOAD\_FILE\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_LOAD\_FILE\_PROTOCOL Section.

No automatic test is designed to test this protocol.

## EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL Section.

### OpenVolume()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.2.1.1 | 0xe1bbbe46, 0x1fe6, 0x4f0b, 0x8d, 0x2e, 0x1b, 0x94, 0x5c, 0x16, 0xf4, 0x87 | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – OpenVolume() returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call OpenVolume() to open root directory at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.2.1.2 | 0xdf0cc997, 0x16b5, 0x4f26, 0x9f, 0x95, 0xb5, 0x53, 0x5c, 0x73, 0xe6, 0x86 | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – OpenVolume() returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call OpenVolume() to open root directory at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.2.1.3 | 0xe4d6498c, 0xc4d5, 0x4dd2, 0x93, 0x88, 0x3c, 0x7b, 0xd2, 0x94, 0x9b, 0x4c | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – GetInfo() to retrieve EFI\_FILE\_INFO on root directory at TPL\_APPLICATION returns EFI\_SUCCESS. | 1. Call OpenVolume() to open root directory at TPL\_APPLICATION.  2. Call GetInfo() to retrieve EFI\_FILE\_INFO on root directory. The return code should be EFI\_SUCCESS. |
| 5.7.2.1.4 | 0xeca437ce, 0xcca2, 0x4f7d, 0xb0, 0x55, 0x42, 0x99, 0x78, 0x46, 0xe5, 0x5c | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – GetInfo() to retrieve EFI\_FILE\_INFO on root directory at TPL\_CALLBACK returns EFI\_SUCCESS. | 1. Call OpenVolume() to open root directory at TPL\_CALLBACK.  2. Call GetInfo() to retrieve EFI\_FILE\_INFO on root directory. The return code should be EFI\_SUCCESS. |
| 5.7.2.1.5 | 0xacf4bb1e, 0x292b, 0x46a5, 0x9d, 0x98, 0xac, 0xa1, 0x04, 0x02, 0x43, 0x1c | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume - EFI\_FILE\_INFO.Attribute & EFI\_FILE\_DIRECTORY != 0 at TPL\_APPLICATION. | 1. Call OpenVolume() to open root directory at TPL\_APPLICATION.  2. Call GetInfo() to retrieve EFI\_FILE\_INFO on root directory.  3. EFI\_FILE\_INFO.Attribute & EFI\_FILE\_DIRECTORY != 0. |
| 5.7.2.1.6 | 0x7639775e, 0xb879, 0x4c64, 0x87, 0xde, 0x96, 0x6b, 0xb7, 0x76, 0xb8, 0x6b | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume - EFI\_FILE\_INFO.Attribute & EFI\_FILE\_DIRECTORY != 0 at TPL\_CALLBACK. | 1. Call OpenVolume() to open root directory at TPL\_CALLBACK.  2. Call GetInfo() to retrieve EFI\_FILE\_INFO on root directory. The return code should be EFI\_SUCCESS.  3. EFI\_FILE\_INFO.Attribute & EFI\_FILE\_DIRECTORY != 0. |
| 5.7.2.1.7 | 0x21746222, 0x29c8, 0x4b78, 0x87, 0x3e, 0x35, 0x4e, 0x58, 0x26, 0x79, 0xde | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO on root directory at TPL\_APPLICATION returns EFI\_SUCCESS. | 1. Call OpenVolume() to open root directory at TPL\_APPLICATION.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO on root directory. The return code should be EFI\_SUCCESS. |
| 5.7.2.1.8 | 0x454082d8, 0x05b5, 0x48df, 0xb0, 0x91, 0x99, 0xb7, 0xdd, 0x87, 0x05, 0x10 | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO on root directory at TPL\_CALLBACK returns EFI\_SUCCESS. | 1. Call OpenVolume() to open root directory at TPL\_CALLBACK.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO on root directory. The return code should be EFI\_SUCCESS. |
| 5.7.2.1.9 | 0x31b71760, 0xbe9c, 0x47aa, 0x8c, 0x49, 0x4c, 0xcf, 0x33, 0x44, 0x57, 0x9f | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL on root directory at TPL\_APPLICATION returns EFI\_SUCCESS. | 1. Call OpenVolume() to open root directory at TPL\_APPLICATION.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL on root directory. The return code should be EFI\_SUCCESS. |
| 5.7.2.1.10 | 0x7853b6a4, 0x66ba, 0x4d50, 0xa9, 0x06, 0xd7, 0x9a, 0x12, 0xa9, 0x21, 0x8e | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL on root directory at TPL\_CALLBACK returns EFI\_SUCCESS. | 1. Call OpenVolume() to open root directory at TPL\_CALLBACK.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL on root directory. The return code should be EFI\_SUCCESS. |
| 5.7.2.1.11 | 0x943883d4, 0xb2c6, 0x4041, 0x98, 0xab, 0x34, 0x2f, 0x9c, 0x24, 0x8c, 0x0c | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – Volume labels gotten from EFI\_FILE\_SYSTEM\_INFO and EFI\_FILE\_SYSTEM\_VOLUME\_LABEL should be the same at TPL\_APPLICATION. | 1. Call OpenVolume() to open root directory at TPL\_APPLICATION.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO on root directory.  3. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL on root directory.  4. Volume labels gotten from EFI\_FILE\_SYSTEM\_INFO and EFI\_FILE\_SYSTEM\_VOLUME\_LABEL should be the same. |
| 5.7.2.1.12 | 0x6fdeb4e4, 0xe12d, 0x4c6b, 0x8e, 0x8c, 0xcd, 0x83, 0x34, 0x0b, 0x1f, 0xe6 | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – Volume labels gotten from EFI\_FILE\_SYSTEM\_INFO and EFI\_FILE\_SYSTEM\_VOLUME\_LABEL should be the same at TPL\_CALLBACK. | 1. Call OpenVolume() to open root directory at TPL\_CALLBACK.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO on root directory.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL on root directory.  4. Volume labels gotten from EFI\_FILE\_SYSTEM\_INFO and EFI\_FILE\_SYSTEM\_VOLUME\_LABEL should be the same |
| 5.7.2.1.13 | 0x2b9fe6a3, 0xd6b0, 0x4ab9, 0x9e, 0x92, 0xbe, 0x93, 0xba, 0x4f, 0xcd, 0xe1 | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – Delete() root directory returns EFI\_WARN\_DELETE\_FAILURE at TPL\_APPLICATION. | 1. Call OpenVolume() to open root directory at TPL\_APPLICATION.  2. Call Delete() to delete root directory. The return code should be EFI\_WARN\_DELETE\_FAILURE. |
| 5.7.2.1.14 | 0xf958f344, 0xa399, 0x437e, 0xa8, 0x85, 0x29, 0xab, 0x58, 0xe6, 0x88, 0x91 | EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL.OpenVolume – Delete() root directory returns EFI\_WARN\_DELETE\_FAILURE at TPL\_CALLBACK. | 1. Call OpenVolume() to open root directory at TPL\_CALLBACK.  2. Call Delete() to delete root directory. The return code should be EFI\_WARN\_DELETE\_FAILURE. |

## EFI\_FILE\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_FILE\_ PROTOCOL Section.

### Open()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.1.1 | 0x9c974f8c, 0x9e6a, 0x4188, 0x81, 0xc5, 0x7f, 0x1a, 0x12, 0x33, 0x60, 0x94 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under root directory with pure filename returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create file under root directory with pure filename at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.2 | 0x02e9e015, 0x3ed6, 0x4c43, 0x91, 0xec, 0x6e, 0x70, 0x05, 0xe1, 0xfd, 0xc0 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under root directory with pure filename returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create file under root directory with pure filename at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.3 | 0x76e95e01, 0xf92b, 0x4068, 0xab, 0x80, 0x06, 0x25, 0x30, 0x8b, 0x8a, 0x06 | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under root directory returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create directory under root directory at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.4 | 0xa5073db1, 0x277c, 0x4714, 0xb2, 0x67, 0x24, 0xd2, 0x2f, 0xbc, 0x4b, 0x96 | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under root directory returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create directory under root directory at TPL\_CALLBACK. The return code should be EFI\_SUCCESS |
| 5.7.3.1.5 | 0xecc31b62, 0x9297, 0x454d, 0xbd, 0x50, 0x9c, 0x63, 0xd4, 0x8a, 0x01, 0xe9 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under root directory with filename containing sub directory name returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create file under root directory with filename containing sub directory name at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.6 | 0x99a1fb48, 0xe279, 0x4b2f, 0x9c, 0x74, 0x42, 0xa1, 0x35, 0x7b, 0x83, 0x93 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under root directory with filename containing sub directory name returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create file under root directory with filename containing sub directory name at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.7 | 0xc2a6e394, 0x4e56, 0x41a0, 0x84, 0xce, 0xf2, 0xd3, 0x30, 0x74, 0xda, 0xa3 | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under root directory returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create directory under root directory at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.8 | 0x6e444a6d, 0x6eb0, 0x42cc, 0x9b, 0xcb, 0x26, 0x79, 0x29, 0xc1, 0x69, 0x25 | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under root directory returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create directory under root directory at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.9 | 0x355911f3, 0x0f0e, 0x4deb, 0x9e, 0x8b, 0x70, 0xa2, 0x04, 0x6b, 0x77, 0x38 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under sub directory with pure filename returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create file under sub directory with pure filename at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.10 | 0xc2c3a263, 0x7b56, 0x4845, 0x8f, 0x50, 0x8c, 0xf4, 0x61, 0xdb, 0x7f, 0x53 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under sub directory with pure filename returns EFI\_SUCCESS at TPL\_CALLBACK | 1. Call Open() to create file under sub directory with pure filename at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.11 | 0xaffa623a, 0x30f8, 0x44e3, 0xad, 0x85, 0x36, 0xa3, 0xa3, 0xdb, 0xfd, 0x9f | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under root directory returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create directory under root directory at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.12 | 0xed784eaf, 0x75db, 0x4bde, 0x8d, 0x5e, 0xeb, 0x5d, 0x22, 0x9a, 0x59, 0x39 | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under root directory returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create directory under root directory at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.13 | 0xf9517e49, 0x4aea, 0x4b7b, 0xa3, 0x92, 0xd9, 0x37, 0x55, 0x3e, 0x3a, 0x6c | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under sub directory returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create directory under sub directory at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.14 | 0x9294bf3e, 0x589f, 0x498b, 0x97, 0xca, 0xf3, 0xb4, 0xda, 0x1a, 0xb8, 0x4c | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under sub directory returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create directory under sub directory at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.15 | 0xe01a3218, 0x4f72, 0x4c8f, 0x9d, 0x13, 0x41, 0xbf, 0x02, 0xc4, 0x39, 0xb3 | EFI\_FILE\_PROTOCOL.Open - Open() to create file with sub directory handle and filename containing sub directory name returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create directory(dir1) under root directory at TPL\_APPLICATION.  2. Call Open() to create directory(dir2) under sub directory(dir1) .  3. Call Open() to create file with sub directory handle(dir1) and filename containing sub directory name(dir2\filename). The return code should be EFI\_SUCCESS. |
| 5.7.3.1.16 | 0x6b32a44f, 0x5670, 0x4ce6, 0xbb, 0xb5, 0x36, 0xd8, 0x29, 0x0e, 0x2c, 0x50 | EFI\_FILE\_PROTOCOL.Open - Open() to create file with sub directory handle and filename containing sub directory name returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create directory(dir1) under root directory at TPL\_CALLBACK.  2. Call Open() to create directory(dir2) under sub directory(dir1) .  3. Call Open() to create file with sub directory handle(dir1) and filename containing sub directory name(dir2\filename). The return code should be EFI\_SUCCESS. |
| 5.7.3.1.17 | 0x36c16a36, 0x0891, 0x4108, 0x84, 0x86, 0x9f, 0x24, 0x3d, 0xde, 0x25, 0xe2 | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under root directory returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create directory under root directory at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.18 | 0xf9f0d04b, 0x1409, 0x4157, 0x9b, 0x51,0x80, 0xf0, 0xb5, 0xea, 0xa4, 0x92 | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under root directory returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create directory under root directory at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.19 | 0xda80d9df, 0xa96b, 0x44d1, 0xac, 0xba, 0x0b, 0x92, 0x1d, 0xe1, 0xf2, 0x72 | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under sub directory returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create directory under sub directory at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.20 | 0xdf2be803, 0x6ae8, 0x477f, 0x99, 0xf3, 0xd4, 0x90, 0x80, 0x90, 0x15, 0x2c | EFI\_FILE\_PROTOCOL.Open - Open() to create directory under sub directory returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create directory under sub directory at TPL\_CALLBACK. The return code should be EFI\_SUCCESS |
| 5.7.3.1.21 | 0xc48ebac5, 0xc94a, 0x434d, 0x8a, 0x35, 0xb6, 0x40, 0x61, 0x98, 0xf9, 0xa3 | EFI\_FILE\_PROTOCOL.Open - Open() to create file with root handle and filename containing absolute file path returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create directory(dir1) under root directory at TPL\_APPLICATION.  2. Call Open() to create directory(dir2) under sub directory(dir1) .  3. Call Open() to create file with root handle and filename containing absolute file path (dir1\dir2\filename). The return code should be EFI\_SUCCESS. |
| 5.7.3.1.22 | 0x9ae5e6ce, 0x1e6e, 0x42b7, 0x93, 0x6b, 0x66, 0xd6, 0x94, 0x2f, 0x9b, 0x98 | EFI\_FILE\_PROTOCOL.Open - Open() to create file with root handle and filename containing absolute file path returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create directory(dir1) under root directory at TPL\_CALLBACK.  2. Call Open() to create directory(dir2) under sub directory(dir1) .  3. Call Open() to create file with root handle and filename containing absolute file path (dir1\dir2\filename). The return code should be EFI\_SUCCESS. |
| 5.7.3.1.23 | 0x533f1869, 0xebc8, 0x444c, 0x8c, 0xf6, 0x44, 0x09, 0x80, 0x31, 0xce, 0xa4 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under root directory with pure filename returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create file under root directory with pure filename at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.24 | 0xe2ba78af, 0xa282, 0x45f4, 0xaa, 0x92, 0xe6, 0xa2, 0xc7, 0xad, 0x8c, 0x70 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under root directory with pure filename returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create file under root directory with pure filename at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.25 | 0x9eb5fd5d, 0x3d06, 0x4e49, 0x98, 0xb4, 0x41, 0x77, 0x61, 0xca, 0x3a, 0x75 | EFI\_FILE\_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL\_APPLICATION. | 1. Call Open() to create an existing file under root directory with pure filename at TPL\_APPLICATION. The existing file should be opened and the return code should be EFI\_SUCCESS. |
| 5.7.3.1.26 | 0x6ed38ac8, 0x0f4a, 0x4294, 0x9d, 0x9c, 0xb6, 0x6f, 0xa6, 0x00, 0xcf, 0xd1 | EFI\_FILE\_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL\_CALLBACK. | 1. Call Open() to create an existing file under root directory with pure filename at TPL\_CALLBACK. The existing file should be opened and the return code should be EFI\_SUCCESS. |
| 5.7.3.1.27 | 0x1a6ec46e, 0x5a2b, 0x43e0, 0x98, 0x59, 0x36, 0x6a, 0xbb, 0xdf, 0xf4, 0x86 | EFI\_FILE\_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI\_SUCCESS except read-only mode at TPL\_APPLICATION. | 1. Call Open() to create a file.  2. Call Write() and SetInfo() to the new file. The return code should be EFI\_SUCCESS and the file size should be equal to the set value. |
| 5.7.3.1.28 | 0x5e81beb3, 0x3cee, 0x4724, 0xa9, 0xa4, 0x6d, 0x64, 0xd4, 0x80, 0x87, 0x5d | EFI\_FILE\_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI\_SUCCESS except read-only mode at TPL\_CALLBACK | 1. Call Open() to create a file.  2. Call Write() and SetInfo() to the new file. The return code should be EFI\_SUCCESS and the file size should be equal to the set value. |
| 5.7.3.1.29 | 0x249b05c8, 0x931f, 0x4d21, 0xb3, 0x09, 0xbc, 0x0f, 0x32, 0x8f, 0x17, 0xc6 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under root directory with filename containing sub directory name returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create file under root directory with filename containing sub directory name at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.30 | 0x2c6a8296, 0x3fd8, 0x4e72, 0x80, 0x84, 0x8f, 0x3e, 0x61, 0x59, 0x10, 0xe2 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under root directory with filename containing sub directory name returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create file under root directory with filename containing sub directory name at TPL\_CALLBACK. The return code should be EFI\_SUCCESS |
| 5.7.3.1.31 | 0xf2ec0ec0, 0xc79b, 0x4035, 0xbf, 0x70, 0x3a, 0x0f, 0x02, 0xf7, 0xe9, 0x4f | EFI\_FILE\_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL\_APPLICATION. | 1. Call Open() to create an existing file under root directory with pure filename at TPL\_APPLICATION. The existing file should be opened and the return code should be EFI\_SUCCESS. |
| 5.7.3.1.32 | 0x18b07457, 0x2108, 0x4c00, 0xb4, 0xb6, 0x88, 0xcb, 0xb1, 0x0e, 0x79, 0xfe | EFI\_FILE\_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL\_CALLBACK | 1. Call Open() to create an existing file under root directory with pure filename at TPL\_CALLBACK. The existing file should be opened and the return code should be EFI\_SUCCESS. |
| 5.7.3.1.33 | 0x4ee79e47, 0x1530, 0x42ee, 0xbb, 0x70, 0x1a, 0xf3, 0xad, 0x6a, 0xef, 0xda | EFI\_FILE\_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI\_SUCCESS except read-only mode at TPL\_APPLICATION. | 1. Call Open() to create a file.  2. Call Write() and SetInfo() to the new file. The return code should be EFI\_SUCCESS and the file size should be equal to the set value. |
| 5.7.3.1.34 | 0x0e5ad766, 0x5368, 0x4465, 0xa5, 0x15, 0x1d, 0x5e, 0x65, 0xc9, 0x28, 0x9b | EFI\_FILE\_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI\_SUCCESS except read-only mode at TPL\_CALLBACK. | 1. Call Open() to create a file.  2. Call Write() and SetInfo() to the new file. The return code should be EFI\_SUCCESS and the file size should be equal to the set value. |
| 5.7.3.1.35 | 0x5f2d183d, 0x748e, 0x4b7b, 0x82, 0xd8, 0xa9, 0xf3, 0x27, 0xa1, 0x96, 0xe3 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under directory handle with pure filename returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create file under directory handle with pure filename at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.36 | 0x953e3193, 0x444c, 0x47d0, 0x8f, 0x79, 0xb8, 0xa3, 0x02, 0xd2, 0x31, 0x85 | EFI\_FILE\_PROTOCOL.Open - Open() to create file under directory handle with pure filename returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create file under directory handle with pure filename at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.37 | 0xbf004c75, 0x42b1, 0x4038, 0xac, 0x38, 0x1c, 0xfb, 0x6d, 0x24, 0xa9, 0xfa | EFI\_FILE\_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL\_APPLICATION. | 1. Call Open() to create an existing file under directory handle with pure filename at TPL\_APPLICATION. The existing file should be opened and the return code should be EFI\_SUCCESS. |
| 5.7.3.1.38 | 0xe5bae1ec, 0x1ce7, 0x4fde, 0xad, 0x6d, 0xfd, 0x0c, 0x85, 0xe8, 0x51, 0x44 | EFI\_FILE\_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL\_CALLBACK. | 1. Call Open() to create an existing file under directory handle with pure filename at TPL\_CALLBACK. The existing file should be opened and the return code should be EFI\_SUCCESS. |
| 5.7.3.1.39 | 0x16fb933e, 0x1f91, 0x46e6, 0x9e, 0xff, 0xfc, 0x4f, 0x6c, 0x91, 0xde, 0xc4 | EFI\_FILE\_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI\_SUCCESS except read-only mode at TPL\_APPLICATION. | 1. Call Open() to create a file.  2. Call Write() and SetInfo() to the new file. The return code should be EFI\_SUCCESS and the file size should be equal to the set value. |
| 5.7.3.1.40 | 0x1225566e, 0xb893, 0x4059, 0xa2, 0x55, 0xfc, 0x4c, 0x0c, 0xb5, 0x72, 0x9c | EFI\_FILE\_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI\_SUCCESS except read-only mode at TPL\_CALLBACK. | 1. Call Open() to create a file.  2. Call Write() and SetInfo() to the new file. The return code should be EFI\_SUCCESS and the file size should be equal to the set value. |
| 5.7.3.1.41 | 0x62066bfd, 0x6a13, 0x43db, 0x99, 0x85, 0x3f, 0xeb, 0x92, 0xd3, 0x00, 0x7d | EFI\_FILE\_PROTOCOL.Open - Open() to create file with directory handle and filename containing sub directory name returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create file with directory handle and filename containing sub directory name at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.42 | 0x2c61c286, 0xd23a, 0x414e, 0x9a, 0x1a, 0x5a, 0xe0, 0xf6, 0xb5, 0x40, 0x33 | EFI\_FILE\_PROTOCOL.Open - Open() to create file with directory handle and filename containing sub directory name returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create file with directory handle and filename containing sub directory name at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.43 | 0x0f0c6895, 0x5d36, 0x4ec9, 0xa4, 0xd8, 0x10, 0x32, 0x94, 0xf2, 0x8a, 0x91 | EFI\_FILE\_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL\_APPLICATION | 1. Call Open() to create an existing file with directory handle and filename containing sub directory name at TPL\_APPLICATION. The existing file should be opened and the return code should be EFI\_SUCCESS. |
| 5.7.3.1.44 | 0x961d3514, 0x0fa1, 0x46d0, 0x83, 0x63, 0x0e, 0xa9, 0x96, 0x45, 0xe4, 0xec | EFI\_FILE\_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL\_CALLBACK. | 1. Call Open() to create an existing file with directory handle and filename containing sub directory name at TPL\_CALLBACK. The existing file should be opened and the return code should be EFI\_SUCCESS. |
| 5.7.3.1.45 | 0x84c25d8c, 0xc15c, 0x4b18, 0x8c, 0xf8, 0x75, 0x82, 0x73, 0xb1, 0x3f, 0x3e | EFI\_FILE\_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI\_SUCCESS except read-only mode at TPL\_APPLICATION. | 1. Call Open() to create a file.  2. Call Write() and SetInfo() to the new file. The return code should be EFI\_SUCCESS and the file size should be equal to the set value. |
| 5.7.3.1.46 | 0x59f0b532, 0x9f66, 0x44f7, 0xa4, 0x7b, 0x0e, 0xb3, 0x65, 0xe8, 0x47, 0x06 | EFI\_FILE\_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI\_SUCCESS except read-only mode at TPL\_CALLBACK. | 1. Call Open() to create a file.  2. Call Write() and SetInfo() to the new file. The return code should be EFI\_SUCCESS and the file size should be equal to the set value. |
| 5.7.3.1.47 | 0xcf2f3c12, 0x0608, 0x4661, 0xa3, 0x86, 0xf7, 0xc3, 0x13, 0x93, 0xef, 0x7a | EFI\_FILE\_PROTOCOL.Open - Open() to create file with directory handle and filename containing absolute directory name returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create file with directory handle and filename containing absolute directory name at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.48 | 0x4b3cdcfd, 0xf479, 0x43c2, 0xbb, 0x48, 0xd6, 0xa4, 0x06, 0x06, 0xc5, 0xc2 | EFI\_FILE\_PROTOCOL.Open - Open() to create file with directory handle and filename containing absolute directory name returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create file with directory handle and filename containing absolute directory name at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.1.49 | 0x7c4c7717, 0x89cf, 0x46b0, 0x84, 0x89, 0xd6, 0x18, 0x54, 0xa3, 0xc3, 0x2b | EFI\_FILE\_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL\_APPLICATION. | 1. Call Open() to create an existing file with directory handle and filename containing absolute directory name at TPL\_APPLICATION. The existing file should be opened and the return code should be EFI\_SUCCESS. |
| 5.7.3.1.50 | 0xc4849d07, 0x41e8, 0x4636, 0xa8, 0x3e, 0xb1, 0x7c, 0xc0, 0xe9, 0x4f, 0x26 | EFI\_FILE\_PROTOCOL.Open - Open() to create an existing file opens the existing file at TPL\_CALLBACK | 1. Call Open() to create an existing file with directory handle and filename containing absolute directory name at TPL\_CALLBACK. The existing file should be opened and the return code should be EFI\_SUCCESS. |
| 5.7.3.1.51 | 0xddd23c97, 0xecc8, 0x434d, 0xb0, 0x69, 0x22, 0xcb, 0x26, 0xb5, 0x88, 0xfe | EFI\_FILE\_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI\_SUCCESS except read-only mode at TPL\_APPLICATION | 1. Call Open() to create a file.  2. Call Write() and SetInfo() to the new file. The return code should be EFI\_SUCCESS and the file size should be equal to the set value. |
| 5.7.3.1.52 | 0x0e86769e, 0xe067, 0x4593, 0x82, 0x4c, 0xc9, 0x85, 0x97, 0x51, 0xac, 0x61 | EFI\_FILE\_PROTOCOL.Open – Write() and SetInfo() to existing file returns EFI\_SUCCESS except read-only mode at TPL\_CALLBACK | 1. Call Open() to create a file.  2. Call Write() and SetInfo() to the new file. The return code should be EFI\_SUCCESS and the file size should be equal to the set value. |
| 5.7.3.1.53 | 0x77240620, 0xcee3, 0x481d, 0xa6, 0xb4, 0x8d, 0x68, 0x50, 0x83, 0x91, 0xd1 | EFI\_FILE\_PROTOCOL.Open – Open() with non‑existent file name returns EFI\_NOT\_FOUND. | 1. Call Open() to open a non‑existent file. The return code should be EFI\_NOT\_FOUND. |
| 5.7.3.1.54 | 0xa08a7f58, 0xbedb, 0x467b, 0x99, 0x43, 0xe7, 0xe9, 0x6e, 0x62, 0x0d, 0x11 | EFI\_FILE\_PROTOCOL.Open – Open() with non‑existent file path returns EFI\_NOT\_FOUND. | 1. Call Open() to create a file handle with non‑existent file path. The return code should be EFI\_NOT\_FOUND. |
| 5.7.3.1.55 | 0xe2310546, 0xf1ac, 0x47ce, 0xa5, 0x65, 0x88, 0xd0, 0x03, 0x2a, 0x50, 0xa7 | EFI\_FILE\_PROTOCOL.Open – Open() with invalid open-mode returns EFI\_INVALID\_PARAMETER. | 1. Call Open() to open file handle with invalid open-mode. The return code should be EFI\_INVALID\_PARAMETER. |

### Close()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.2.1 | 0x8f787cb1, 0xe4d7, 0x4d58, 0x97, 0x5b, 0xb6, 0xf1, 0x42, 0xc8, 0xcb, 0xc8 | EFI\_FILE\_PROTOCOL.Close – Close() file handle or directory handle returns EFI\_SUCCESS at TPL\_APPLICATION | 1. Call Open() to create file or directory handles at TPL\_APPLICATION.  2. Call Close() to close file handle and directory handles. The return code should be EFI\_SUCCESS. |
| 5.7.3.2.2 | 0x301114f7, 0x1f9d, 0x4dcb, 0xb2, 0xc7, 0x24, 0x17, 0x24, 0x66, 0xc6, 0xd9 | EFI\_FILE\_PROTOCOL.Close – Close() file handle or directory handle returns EFI\_SUCCESS at TPL\_CALLBACK | 1. Call Open() to create file or directory handles at TPL\_CALLBACK.  2. Call Close() to close file handle and directory handles. The return code should be EFI\_SUCCESS. |
| 5.7.3.2.3 | 0x134343f0, 0xee4d, 0x4c3d, 0xa5, 0x5d, 0xa2, 0x3c, 0x48, 0x75, 0x51, 0x0c | EFI\_FILE\_PROTOCOL.Close – Closing a directory does not affect access to files under that directory if the files were opened before the directory was closed at TPL\_APPLICATION. | 1. Call Open() to create directory and file handles under the directory at TPL\_APPLICATION.  2. Call Close() to close the directory handle.  3. Read/Write/GetInfo/SetInfo/GetPosition/SetPosition to the file handles under the closed directory should be a success. |
| 5.7.3.2.4 | 0x46f37004, 0x407a, 0x481f, 0x9a, 0xe6, 0x9f, 0x74, 0x40, 0x93, 0xd7, 0xe8 | EFI\_FILE\_PROTOCOL.Close – Closing a directory does not affect access to files under that directory if the files were opened before the directory was closed at TPL\_CALLBACK. | 1. Call Open() to create directory and file handles under the directory at TPL\_CALLBACK.  2. Call Close() to close the directory handle.  3. Read/Write/GetInfo/SetInfo/GetPosition/SetPosition to the file handles under the closed directory should be a success. |
| 5.7.3.2.5 | 0xc5da488d, 0x0bbb, 0x49f2, 0xb5, 0xc5, 0xb0, 0x3a, 0xbb, 0x40, 0xe0, 0x42 | EFI\_FILE\_PROTOCOL.Close – Re-Open closed file or directory handle returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create file or directory handles at TPL\_APPLICATION.  2. Call Close() to close file handle and directory handles.  3. Call Open() to re‑open the closed handles. The return code should be EFI\_SUCCESS. |
| 5.7.3.2.6 | 0xb9478756, 0x46c4, 0x4eaa, 0xa0, 0x35, 0x8b, 0xd2, 0x28, 0xbb, 0xd7, 0x9c | EFI\_FILE\_PROTOCOL.Close – Re-Open closed file or directory handle returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create file or directory handles at TPL\_CALLBACK.  2. Call Close() to close file handle and directory handles.  3. Call Open() to re‑open the closed handles. The return code should be EFI\_SUCCESS. |

### Delete()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.3.1 | 0xaf9e9d9c, 0x1814, 0x4623, 0x87, 0xac, 0xe5, 0xa3, 0xff, 0x79, 0xfa, 0xf2 | EFI\_FILE\_PROTOCOL.Delete – Delete() file handle returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Delete() to delete file handles at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.3.2 | 0x7db63d3b, 0x7819, 0x4f45, 0xa1, 0xfd, 0x75, 0xeb, 0x18, 0xcc, 0xfc, 0x33 | EFI\_FILE\_PROTOCOL.Delete – Delete() file handle returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Delete() to delete file handles at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.3.3 | 0xb250cf45, 0x9dd8, 0x41f7, 0x8e, 0x27, 0x96, 0x5e, 0x89, 0xc2, 0xd6, 0x32 | EFI\_FILE\_PROTOCOL.Delete – Re‑open deleted file handle returns EFI\_NOT\_FOUND at TPL\_APPLICATION. | 1. Call Delete() to delete file handles at TPL\_APPLICATION.  2. Call Open() to re‑open the deleted file handle. The return code should be EFI\_NOT\_FOUND. |
| 5.7.3.3.4 | 0xf4dc2e77, 0xd9c7, 0x40d0, 0x83, 0xbd, 0x8f, 0x1e, 0xc6, 0x64, 0x86, 0x69 | EFI\_FILE\_PROTOCOL.Delete – Re‑open deleted file handle returns EFI\_NOT\_FOUND at TPL\_CALLBACK. | 1. Call Delete() to delete file handles at TPL\_CALLBACK.  2. Call Open() to re‑open the deleted file handle. The return code should be EFI\_NOT\_FOUND. |
| 5.7.3.3.5 | 0xb656663f, 0x5c23, 0x4e47, 0xa1, 0x77, 0xc3, 0x34, 0x14, 0x0c, 0x11, 0x07 | EFI\_FILE\_PROTOCOL.Delete – Delete() directory handle returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Delete() to delete directory handles at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.3.6 | 0x0f51d637, 0xa67a, 0x4c97, 0x81, 0xcf, 0xbb, 0x8c, 0x4d, 0xf2, 0xdb, 0xdf | EFI\_FILE\_PROTOCOL.Delete – Delete() directory handle returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Delete() to delete directory handles at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.3.7 | 0xe0678dae, 0x5aa6, 0x426c, 0xa4, 0xcb, 0x58, 0xa2, 0x7a, 0x9a, 0xa7, 0xb2 | EFI\_FILE\_PROTOCOL.Delete – Re‑open deleted directory handle returns EFI\_NOT\_FOUND at TPL\_APPLICATION. | 1. Call Delete() to delete directory handles at TPL\_APPLICATION.  2. Call Open() to re‑open the deleted directory handle. The return code should be EFI\_NOT\_FOUND. |
| 5.7.3.3.8 | 0xb9c79e4e, 0x187f, 0x46c6, 0x8d, 0x0a, 0x71, 0x70, 0xca, 0x99, 0x31, 0xa7 | EFI\_FILE\_PROTOCOL.Delete – Re‑open deleted directory handle returns EFI\_NOT\_FOUND at TPL\_CALLBACK. | 1. Call Delete() to delete directory handles at TPL\_CALLBACK.  2. Call Open() to re‑open the deleted directory handle. The return code should be EFI\_NOT\_FOUND. |
| 5.7.3.3.9 | 0x12f4e25b, 0x77c7, 0x4b47, 0x98, 0xb3, 0xd4, 0xf1, 0x54, 0x24, 0x68, 0x98 | EFI\_FILE\_PROTOCOL.Delete – Delete() nonempty directory returns EFI\_WARN\_DELETE\_FAILURE at TPL\_APPLICATION. | 1. Call Open() to create directory and file handles under the directory at TPL\_APPLICATION.  2. Call Delete() to delete the directory handle. The return code should be EFI\_WARN\_DELETE\_FAILURE |
| 5.7.3.3.10 | 0x11860155, 0x016e, 0x4c07, 0x83, 0x7a, 0xf1, 0x27, 0x59, 0x2b, 0xf0, 0x75 | EFI\_FILE\_PROTOCOL.Delete – Delete() nonempty directory returns EFI\_WARN\_DELETE\_FAILURE at TPL\_CALLBACK. | 1. Call Open() to create directory and file handles under the directory at TPL\_CALLBACK.  2. Call Delete() to delete the directory handle. The return code should be EFI\_WARN\_DELETE\_FAILURE. |
| 5.7.3.3.11 | 0x619d8713, 0xd755, 0x4293, 0xbe, 0x3d, 0x19, 0xb0, 0x17, 0xa8, 0xd4, 0x09 | EFI\_FILE\_PROTOCOL.Delete – Re‑open of undeleted directory handle returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Open() to create directory and file handles under the directory at TPL\_APPLICATION.  2. Call Delete() to delete the directory handle. The return code should be EFI\_WARN\_DELETE\_FAILURE.  3. Call Open() to re‑open the directory. The return code should be EFI\_SUCCESS. |
| 5.7.3.3.12 | 0xb9306618, 0x2613, 0x4a6a, 0xaa, 0x72, 0x5e, 0xf7, 0x2e, 0xc8, 0x07, 0xf6 | EFI\_FILE\_PROTOCOL.Delete – Re‑open of undeleted directory handle returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Open() to create directory and file handles under the directory at TPL\_CALLBACK.  2. Call Delete() to delete the directory handle. The return code should be EFI\_WARN\_DELETE\_FAILURE.  3. Call Open() to re‑open the directory. The return code should be EFI\_SUCCESS. |
| 5.7.3.3.13 | 0xca4a0455, 0xee2a, 0x4260, 0x8e, 0xdc, 0x12, 0xb4, 0xd2, 0xc0, 0x1b, 0x79 | EFI\_FILE\_PROTOCOL.Delete – Delete() on root directory returns EFI\_WARN\_DELETE\_FAILURE. | 1. Call Delete() on root directory. The return code should be EFI\_WARN\_DELETE\_FAILURE. |
| 5.7.3.3.14 | 0xda598731, 0xf3da, 0x4f63, 0xa3, 0x49, 0x15, 0x1e, 0x0b, 0x77, 0xe3, 0x6f | EFI\_FILE\_PROTOCOL.Delete – Open() on root directory returns EFI\_SUCCESS after Delete() on root fails. | 1. Call Delete() on root directory. It returns EFI\_WARN\_DELETE\_FAILURE.  2. Call Open() on root. The return code should be EFI\_SUCCESS. |

### Read()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.4.1 | 0xf98a984c, 0x0043, 0x481e, 0x93, 0x3a, 0x24, 0x4d, 0xc8, 0x6e, 0x71, 0x1a | EFI\_FILE\_PROTOCOL.Read – Read() from file handle returns EFI\_SUCCESS except when read position is beyond file end at TPL\_APPLICATION. | 1. Call Read() to read from file handles at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. If read position is beyond file end, the return code should be EFI\_DEVICE\_ERROR. |
| 5.7.3.4.2 | 0x192d00c3, 0x604e, 0x49bb, 0xb0, 0xc2, 0x5b, 0x25, 0x69, 0x6e, 0xa9, 0x2f | EFI\_FILE\_PROTOCOL.Read – Read() from file handle returns EFI\_SUCCESS except when read position is beyond file end at TPL\_CALLBACK. | 1. Call Read() to read from file handles at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. If read position is beyond file end, the return code should be EFI\_DEVICE\_ERROR. |
| 5.7.3.4.3 | 0xfbff4d9d, 0xe021, 0x482b, 0x8d, 0x92, 0x99, 0xd5, 0x26, 0x89, 0xe0, 0xd3 | EFI\_FILE\_PROTOCOL.Read – GetPosition() after read returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Read() to read from file handles at TPL\_APPLICATION.  2. Call GetPosition() to get current file position. The return code should be EFI\_SUCCESS. |
| 5.7.3.4.4 | 0x522c18d5, 0xe922, 0x4844, 0xbb, 0x59, 0x5f, 0xdd, 0x48, 0xf8, 0xfe, 0xbc | EFI\_FILE\_PROTOCOL.Read – GetPosition() after read returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Read() to read from file handles at TPL\_CALLBACK.  2. Call GetPosition() to get current file position. The return code should be EFI\_SUCCESS. |
| 5.7.3.4.5 | 0x69decc47, 0xbc8d, 0x44e9, 0x92, 0x3c, 0x63, 0x02, 0x2d, 0xd2, 0xe1, 0x1f | EFI\_FILE\_PROTOCOL.Read – If read position is beyond file end, buffer size should be truncated at TPL\_APPLICATION. | 1. Call Read() to read from file handles at TPL\_APPLICATION.  2. Call GetPosition() to get current file position.  3. If read position is beyond file end, buffer size should be truncated. |
| 5.7.3.4.6 | 0x4ba1060f, 0xdaba, 0x4d5b, 0xb9, 0xce, 0xf8, 0x59, 0xbb, 0xbf, 0x52, 0x69 | EFI\_FILE\_PROTOCOL.Read – If read position is beyond file end, buffer size should be truncated at TPL\_CALLBACK. | 1. Call Read() to read from file handles at TPL\_CALLBACK.  2. Call GetPosition() to get current file position.  3. If read position is beyond file end, buffer size should be truncated. |
| 5.7.3.4.7 | 0xe0ebe6bd, 0x0fd2, 0x4c14, 0x84, 0xd5, 0xec, 0xd3, 0x96, 0x6a, 0x04, 0xed | EFI\_FILE\_PROTOCOL.Read – If read beyond file end, file position updated to the end of the file at TPL\_APPLICATION. | 1. Call Read() to read from file handles at TPL\_APPLICATION.  2. Call GetPosition() to get current file position.  3. If read beyond file end, file position updated to the end of the file. |
| 5.7.3.4.8 | 0xca6d5592, 0x48a9, 0x46c8, 0xa8, 0xa4, 0x5a, 0xd8, 0x4b, 0x07, 0x68, 0xf1 | EFI\_FILE\_PROTOCOL.Read – If read postion is beyond file end, file position updated to the end of the file at TPL\_CALLBACK. | 1. Call Read() to read from file handles at TPL\_CALLBACK.  2. Call GetPosition() to get current file position.  3. If read position is beyond file end, file position updated to the end of the file. |
| 5.7.3.4.9 | 0x0b158040, 0xb603, 0x49e2, 0xab, 0x3c, 0xfb, 0x75, 0x31, 0xdf, 0x68, 0xaa | EFI\_FILE\_PROTOCOL.Read – **BufferSize** is equal to the number of bytes read at TPL\_APPLICATION. | 1. Call Read() to read from file handles at TPL\_APPLICATION. **BufferSize** is equal to the number of bytes read. |
| 5.7.3.4.10 | 0xad1fd527, 0xf8d7, 0x4875, 0xab, 0x3d, 0x9c, 0x1f, 0xe0, 0x7e, 0x52, 0x41 | EFI\_FILE\_PROTOCOL.Read – BufferSize is equal to the number of bytes read at TPL\_CALLBACK. | 1. Call Read() to read from file handles at TPL\_CALLBACK. **BufferSize** is equal to the number of bytes read. |
| 5.7.3.4.11 | 0x3ee4c586, 0x9f92, 0x4cc0, 0x9f, 0x32, 0xba, 0xaa, 0xb9, 0x56, 0xce, 0x7b | EFI\_FILE\_PROTOCOL.Read – If read position is within file size, file position is updated to the start position plus read bytes at TPL\_APPLICATION. | 1. Call Read() to read from file handles at TPL\_APPLICATION.  2. Call GetPosition() to get current file position.  3. If read position is within file size, file position is updated to the start position plus read bytes. |
| 5.7.3.4.12 | 0x2fa03a35, 0x34d7, 0x4ede, 0x94, 0xfa, 0xca, 0x2b, 0x78, 0xe1, 0xe5, 0xd6 | EFI\_FILE\_PROTOCOL.Read – If read within file size, file position updated to the start position plus read bytes at TPL\_CALLBACK. | 1. Call Read() to read from file handles at TPL\_CALLBACK.  2. Call GetPosition() to get current file position.  3. If read position is within file size, file position is updated to the start position plus read bytes. |
| 5.7.3.4.13 | 0x54013277, 0xde8a, 0x4f8b, 0xa5, 0x8a, 0x60, 0xe4, 0x17, 0x4c, 0xcd, 0xa2 | EFI\_FILE\_PROTOCOL.Read – Read content should be the same as written at TPL\_APPLICATION. | 1. Call Write() to write bytes into file handle.  2. Call Read() to read from file handles at TPL\_APPLICATION. Read content should be the same as written. |
| 5.7.3.4.14 | 0x74ab30a4, 0xcb1b, 0x4d9b, 0x8c, 0x69, 0x3d, 0xf0, 0xda, 0x61, 0x16, 0x32 | EFI\_FILE\_PROTOCOL.Read – Read content should be the same as written at TPL\_CALLBACK. | 1. Call Write() to write bytes into file handle.  2. Call Read() to read from file handles at TPL\_CALLBACK. Read content should be the same as written. |
| 5.7.3.4.15 | 0x2ff71629, 0x8548, 0x4f11, 0x92, 0x24, 0x43, 0x0e, 0xd1, 0x8e, 0xe9, 0x82 | EFI\_FILE\_PROTOCOL.Read – Read() from directory handle with too small buffer returns EFI\_BUFFER\_TOO\_SMALL at TPL\_APPLICATION. | 1. Call Read() to read from directory handle with too small buffer at TPL\_APPLICATION. The return code should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.7.3.4.16 | 0x3b46d893, 0x289e, 0x4186, 0x9d, 0x13, 0x94, 0xcc, 0x4e, 0x96, 0x1b, 0xd4 | EFI\_FILE\_PROTOCOL.Read – Read() from directory handle with too small buffer returns EFI\_BUFFER\_TOO\_SMALL at TPL\_CALLBACK. | 1. Call Read() to read from directory handle with too small buffer at TPL\_CALLBACK. The return code should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.7.3.4.17 | 0x19979967, 0xf6cb, 0x4043, 0xba, 0x15, 0xdd, 0x80, 0x5e, 0x9f, 0x62, 0xe8 | EFI\_FILE\_PROTOCOL.Read – Read() from directory handle with valid parameter returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Read() to read from directory handle with valid parameter at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.4.18 | 0x25c7de0c, 0x56b7, 0x4e8b, 0x94, 0x9e, 0x59, 0x83, 0x0b, 0x46, 0x4a, 0xf4 | EFI\_FILE\_PROTOCOL.Read – Read() from directory handle with valid parameter returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Read() to read from directory handle with valid parameter at TPL\_APPLICATION. The return code should be EFI\_CALLBACK. |
| 5.7.3.4.19 | 0xacc83dc2, 0x84d4, 0x46fd, 0xa9, 0x51, 0x1c, 0x2f, 0x49, 0xd5, 0x97, 0x9c | EFI\_FILE\_PROTOCOL.Read – Read() at the end of the directory returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Read() to at the end of the directory at TPL\_APPLICATION. The return code should be EFI\_SUCCESS and read buffer size is set to 0. |
| 5.7.3.4.20 | 0x882f4162, 0xb6b9, 0x456f, 0xbe, 0xb9, 0x2e, 0x2b, 0xa4, 0xc9, 0x58, 0x5a | EFI\_FILE\_PROTOCOL.Read – Read() at the end of the directory returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Read() to read at the end of the directory at TPL\_APPLICATION. The return code should be EFI\_CALLBACK and read buffer size is set to 0. |

### 

### Write()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.5.1 | 0x73c93917, 0xad5e, 0x4e21, 0xaa, 0xaa, 0x8e, 0x6a, 0x35, 0x85, 0xe9, 0x51 | EFI\_FILE\_PROTOCOL.Write – Write() to file handle returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Write() to write to file handle at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.5.2 | 0xb58c7d6a, 0x90f6, 0x4a0b, 0xb8, 0x49, 0xdb, 0xba, 0x08, 0x4c, 0x22, 0xb6 | EFI\_FILE\_PROTOCOL.Write – Write() to file handle returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Write() to write to file handle at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.5.3 | 0x9f38fdc4, 0xbbf6, 0x4d1b, 0xae, 0x1c, 0xbb, 0xe8, 0x89, 0xda, 0x8c, 0xc5 | EFI\_FILE\_PROTOCOL.Write – GetPostion() after call of Write() returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Write() to write to file handle at TPL\_APPLICATION.  2. Call GetPosition(). The return code should be EFI\_SUCCESS. |
| 5.7.3.5.4 | 0x1ca546ad, 0xac23, 0x4304, 0xa2, 0xff, 0xec, 0xb0, 0x23, 0xd5, 0xfb, 0x21 | EFI\_FILE\_PROTOCOL.Write – GetPostion() after call of Write() returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Write() to write to file handle at TPL\_CALLBACK.  2. Call GetPosition(). The return code should be EFI\_SUCCESS. |
| 5.7.3.5.5 | 0xca3fdc12, 0x3e36, 0x4a38, 0xb9, 0x46, 0xb5, 0x83, 0x31, 0x67, 0x13, 0xc0 | EFI\_FILE\_PROTOCOL.Write – BufferSize is updated as the number of bytes written at TPL\_APPLICATION. | 1. Call Write() to write to file handle at TPL\_APPLICATION. BufferSize should be updated as the number of bytes written. |
| 5.7.3.5.6 | 0x4e3680d0, 0xf1dc, 0x4736, 0x86, 0xcb, 0x6e, 0xb0, 0xc0, 0xc0, 0x8e, 0xdd | EFI\_FILE\_PROTOCOL.Write – BufferSize is updated as the number of bytes written at TPL\_CALLBACK. | 1. Call Write() to write to file handle at TPL\_CALLBACK. **BufferSize** should be updated as the number of bytes written. |
| 5.7.3.5.7 | 0x99e9e364, 0xeefb, 0x4b2a, 0xb3, 0x29, 0xe4, 0x8b, 0x31, 0x2c, 0xe4, 0x7c | EFI\_FILE\_PROTOCOL.Write – File position is updated after call of Write() at TPL\_APPLICATION. | 1. Call Write() to write to file handle at TPL\_APPLICATION.  2. Call GetPosition(). Current file postion is updated to the end of written contents. |
| 5.7.3.5.8 | 0x06a30897, 0xe2ed, 0x4c76, 0x99, 0xa4, 0xc4, 0x5d, 0x00, 0x9f, 0xef, 0x8d | EFI\_FILE\_PROTOCOL.Write – File position is updated after call of Write() at TPL\_CALLBACK. | 1. Call Write() to write to file handle at TPL\_CALLBACK.  2. Call GetPosition().Current file postion is updated to the end of written contents. |
| 5.7.3.5.9 | 0x0af7cb57, 0x661e, 0x4b4f, 0xb4, 0xa4, 0xf6, 0xe3, 0x16, 0xe1, 0x52, 0x76 | EFI\_FILE\_PROTOCOL.Write – File size is updated after call of Write() at TPL\_APPLICATION. | 1. Call Write() to write to file handle at TPL\_APPLICATION.  2. Call GetInfo(). If write is beyond the end of the file, file size has grown. |
| 5.7.3.5.10 | 0x1d6b4c54, 0x51fe, 0x406e, 0xb5, 0x92, 0x22, 0xe2, 0xa8, 0x74, 0x7e, 0xdc | EFI\_FILE\_PROTOCOL.Write – File size is updated after call of Write() at TPL\_CALLBACK. | 1. Call Write() to write to file handle at TPL\_CALLBACK.  2. Call GetInfo(). If write is beyond the end of the file, file size has grown. |
| 5.7.3.5.11 | 0x67428a37, 0x56f9, 0x400a, 0xb1, 0x00, 0xd7, 0xf7, 0x68, 0x0c, 0x65, 0x5c | EFI\_FILE\_PROTOCOL.Write – Read() after Write() returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Write() to write to file handle at TPL\_APPLICATION.  2. Call Read().The return code should be EFI\_SUCCESS. |
| 5.7.3.5.12 | 0xe5242bc2, 0x0b10, 0x462f, 0x89, 0xa4, 0xfb, 0xe1, 0x41, 0x55, 0xdb, 0x84 | EFI\_FILE\_PROTOCOL.Write - Read() after Write() returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Write() to write to file handle at TPL\_CALLBACK.  2. Call Read().The return code should be EFI\_SUCCESS. |
| 5.7.3.5.13 | 0x4838f93c, 0xd601, 0x4d76, 0x8c, 0x7a, 0x59, 0xfa, 0xfa, 0xa5, 0xb6, 0x6d | EFI\_FILE\_PROTOCOL.Write – Read() after Write() gets the same contents as written at TPL\_APPLICATION. | 1. Call Write() to write to file handle at TPL\_APPLICATION.  2. Call Read(). It should return the same contents as written. |
| 5.7.3.5.14 | 0x3ff81ec0, 0xf7ae, 0x42da, 0x82, 0xdd, 0xb8, 0x59, 0xa1, 0x14, 0x72, 0xe7 | EFI\_FILE\_PROTOCOL.Write - Read() after Write() gets the same contents as written at TPL\_CALLBACK. | 1. Call Write() to write to file handle at TPL\_CALLBACK.  2. Call Read().It should return the same contents as written. |
| 5.7.3.5.15 | 0x29eb1c7e, 0xf4aa, 0x4fc4, 0xa9, 0x68, 0x98, 0xde, 0xa1, 0x75, 0x52, 0x0e | EFI\_FILE\_PROTOCOL.Write – Write() to the directory returns EFI\_UNSUPPORTED. | 1. Call Write() to write to a directory. The return code should be EFI\_UNSUPPORTED. |
| 5.7.3.5.16 | 0xd1b25896, 0x9f3d, 0x467a, 0xbc, 0x92, 0x8a, 0x52, 0x04, 0x96, 0x6a, 0x10 | EFI\_FILE\_PROTOCOL.Write – Write() to a read-only opened file returns EFI\_ACCESS\_DENIED. | 1. Call Write() to write to a read-only opened file. The return code should be EFI\_ACCESS\_DENIED. |

### Flush()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.6.1 | 0xa2070225, 0x0018, 0x4953, 0xb8, 0xfa, 0xbd, 0x17, 0x21, 0xca, 0x68, 0x46 | EFI\_FILE\_PROTOCOL.Flush – Flush() on file handle returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Flush() on file handle at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.6.2 | 0x83b6cdc5, 0xd813, 0x4000, 0xa9, 0x84, 0x07, 0xb6, 0x54, 0xc5, 0x1f, 0xe4 | EFI\_FILE\_PROTOCOL.Flush – Flush() on file handle returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Flush() on file handle at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.6.3 | 0x9f7bfe1e, 0xd617, 0x4920, 0xab, 0x63, 0x2f, 0xae, 0x6d, 0xce, 0x77, 0x5d | EFI\_FILE\_PROTOCOL.Flush – Flush() on directory handle returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call Flush() on directory handle at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.6.4 | 0x42985ef5, 0x8c9b, 0x49df, 0x93, 0x3c, 0x30, 0xf8, 0xc0, 0x22, 0x8e, 0x4b | EFI\_FILE\_PROTOCOL.Flush – Flush() on directory handle returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call Flush() on directory handle at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.6.5 | 0xc7900513, 0xe931, 0x404a, 0xa5, 0xe3, 0xe3, 0x48, 0x40, 0x7a, 0xb2, 0xa2 | EFI\_FILE\_PROTOCOL.Flush - Flush() to a read-only opened file returns EFI\_ACCESS\_DENIED. | 1. Call Flush() to flush a read-only opened file. The return code should be EFI\_ACCESS\_DENIED. |

### SetPosition()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.7.1 | 0x6b383ca4, 0xc8e4, 0x4fe2, 0xa8, 0xdb, 0x8b, 0x87, 0x85, 0x38, 0xb7, 0x7b | EFI\_FILE\_PROTOCOL.SetPosition – SetPosiiton() on file handle returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call SetPosition() on file handle at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.7.2 | 0x546093bf, 0x1ab1, 0x445c, 0x9e, 0x36, 0xb3, 0x90, 0x4c, 0xe5, 0x74, 0x7e | EFI\_FILE\_PROTOCOL.SetPosition – SetPosiiton() on file handle returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call SetPosition() on file handle at TPL\_APPLICATION. The return code should be EFI\_CALLBACK. |
| 5.7.3.7.3 | 0x7fa447e4, 0xae1e, 0x490b, 0x89, 0x53, 0x60, 0xd7, 0xef, 0xd3, 0x0f, 0xed | EFI\_FILE\_PROTOCOL.SetPosition – GetPosiiton() on file handle after call of SetPosition() returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call SetPosition() on file handle at TPL\_APPLICATION.  2. Call GetPosition(). The return code should be EFI\_SUCCESS. |
| 5.7.3.7.4 | 0x89e7eb29, 0xd715, 0x47bc, 0x94, 0xb1, 0xcd, 0xdf, 0x67, 0xd8, 0x44, 0x0a | EFI\_FILE\_PROTOCOL.SetPosition – GetPosiiton() on file handle after call of SetPosition() returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call SetPosition() on file handle at TPL\_APPLICATION.  2. Call GetPosition(). The return code should be EFI\_CALLBACK. |
| 5.7.3.7.5 | 0x19ca2741, 0x4dd5, 0x4df8, 0x9d, 0xb4, 0x84, 0xdb, 0xea, 0xc6, 0x7c, 0x63 | EFI\_FILE\_PROTOCOL.SetPosition – GetPosiiton() on file handle after call of SetPosition() returns the same position as set at TPL\_APPLICATION | 1. Call SetPosition() on file handle at TPL\_APPLICATION.  2. Call GetPosition(). The return position should be the same as set. |
| 5.7.3.7.6 | 0x6e22f1ef, 0x664e, 0x4c58, 0x90, 0xea, 0x32, 0x92, 0x39, 0x02, 0xa6, 0x4f | EFI\_FILE\_PROTOCOL.SetPosition – GetPosiiton() on file handle after call of SetPosition() returns the same position as set at TPL\_CALLBACK. | 1. Call SetPosition() on file handle at TPL\_APPLICATION.  2. Call GetPosition(). The return position should be the same as set. |
| 5.7.3.7.7 | 0xde3f7243, 0xc732, 0x45d7, 0x97, 0x17, 0x4f, 0x85, 0x98, 0x8c, 0x85, 0xd8 | EFI\_FILE\_PROTOCOL.SetPosition – SetPosiiton() on directory handle with 0 position returns EFI\_SUCCESS at TPL\_APPLICATION | 1. Call SetPosition() on directory handle with 0 position at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.7.8 | 0x0f4c0762, 0x9746, 0x42a0, 0xba, 0xbf, 0x64, 0x32, 0xb8, 0xd8, 0x1f, 0xf9 | EFI\_FILE\_PROTOCOL.SetPosition – SetPosiiton() on directory handle with 0 position returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call SetPosition() on directory handle with 0 position at TPL\_APPLICATION. The return code should be EFI\_CALLBACK. |
| 5.7.3.7.9 | 0x5e0586cd, 0x7718, 0x4605, 0x9b, 0xa1, 0x4d, 0xa3, 0xd4, 0x2b, 0xf2, 0x51 | EFI\_FILE\_PROTOCOL.SetPosition – SetPosiiton() on directory handle with non-0 position returns EFI\_UNSUPPORTED. | 1. Call SetPosition() on directory handle with non-0 position. The return code should be EFI\_UNSUPPORTED. |

### GetPosition()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.8.1 | 0x9787af2d, 0xda90, 0x4945, 0xba, 0xaa, 0xe4, 0x13, 0x4e, 0x25, 0xe8, 0x8e | EFI\_FILE\_PROTOCOL.GetPosition – GetPosition() on file handle returns EFI\_SUCCESS at TPL\_APPLICATION | 1. Call GetPosition() on file handle at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.8.2 | 0xfaf1daae, 0x3dbc, 0x484d, 0x9c, 0xe0, 0xd5, 0xc7, 0x30, 0xf3, 0x1d, 0x05 | EFI\_FILE\_PROTOCOL.GetPosition – GetPosition() on file handle returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call GetPosition() on file handle at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.8.3 | 0x50e547cb, 0x0d88, 0x447b, 0xaa, 0x07, 0x22, 0x08, 0x8a, 0xc0, 0x05, 0xb5 | EFI\_FILE\_PROTOCOL.GetPosition – GetPosition() on file handle after call of SetPosition() returns the set value at TPL\_APPLICATION. | 1. Call SetPosition() on file handle.  2. Call GetPosition() on file handle at TPL\_APPLICATION. The return position should be the same value as set. |
| 5.7.3.8.4 | 0x8c1a0c2b, 0x0362, 0x4ba8, 0x90, 0x80, 0x41, 0x66, 0x00, 0xcd, 0x12, 0x87 | EFI\_FILE\_PROTOCOL.GetPosition – GetPosition() on file handle after call of SetPosition() returns the set value at TPL\_CALLBACK. | 1. Call SetPosition() on file handle.  2. Call GetPosition() on file handle at TPL\_CALLBACK. The return position should be the same value as set. |
| 5.7.3.8.5 | 0x9664e456, 0x0e74, 0x4d1f, 0x8e, 0x7b, 0x2c, 0x49, 0xf8, 0x94, 0xdb, 0x49 | EFI\_FILE\_PROTOCOL.GetPosition – GetPosition() on directory handle returns EFI\_UNSUPPORTED. | 1. Call GetPosition() on directory handle. The return code should be EFI\_UNSUPPORTED. |

### GetInfo()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.9.1 | 0xf93e4251, 0x75a8, 0x464e, 0xaa, 0xf9, 0x03, 0xa2, 0x9e, 0x6e, 0xdd, 0x6b | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() on file handle for EFI\_FILE\_INFO returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call GetInfo() on file handle for EFI\_FILE\_INFO at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.9.2 | 0xa40d7f41, 0x959f, 0x4c1a, 0x82, 0x02, 0x83, 0xc5, 0xda, 0x58, 0xbe, 0x9c | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() on file handle for EFI\_FILE\_INFO returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call GetInfo() on file handle for EFI\_FILE\_INFO at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.9.3 | 0x2055cdd1, 0xce8b, 0x4e95, 0xad, 0x6b, 0xfb, 0x5f, 0x95, 0x8c, 0xcc, 0x68 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() on file handle for EFI\_FILE\_INFO returns the same buffer size as FileInfo->Size at TPL\_APPLICATION. | 1. Call GetInfo() on file handle for EFI\_FILE\_INFO at TPL\_APPLICATION. The return buffer size should be the same as FileInfo->Size. |
| 5.7.3.9.4 | 0xe8099e1b, 0x193e, 0x4383, 0x88, 0x67, 0x06, 0xb1, 0xb8, 0x1f, 0x48, 0x38 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() on file handle for EFI\_FILE\_INFO returns the same buffer size as FileInfo->Size at TPL\_CALLBACK. | 1. Call GetInfo() on file handle for EFI\_FILE\_INFO at TPL\_CALLBACK. The return buffer size should be the same as FileInfo->Size. |
| 5.7.3.9.5 | 0x788dc48a, 0xdaac, 0x4c4d, 0x82, 0x5d, 0xe9, 0x46, 0x14, 0x00, 0x71, 0xbe | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() on file handle for EFI\_FILE\_INFO returns the same attribute set by SetInfo() at TPL\_APPLICATION. | 1. Call SetInfo() on file handle.  1. Call GetInfo() on file handle for EFI\_FILE\_INFO at TPL\_APPLICATION. The return attribute should be the same as set value. |
| 5.7.3.9.6 | 0x32abd0a4, 0x768d, 0x45a8, 0x9c, 0xf6, 0x2d, 0xc3, 0x56, 0x80, 0x58, 0xa0 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() on file handle for EFI\_FILE\_INFO returns the same attribute set by SetInfo() at TPL\_CALLBACK. | 1. Call SetInfo() on file handle.  1. Call GetInfo() on file handle for EFI\_FILE\_INFO at TPL\_CALLBACK. The return attribute should be the same as set value. |
| 5.7.3.9.7 | 0xfea5ef36, 0x87e1, 0x4282, 0xbc, 0xb9, 0xde, 0x54, 0x5d, 0x20, 0xb7, 0x06 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() on file handle for EFI\_FILE\_INFO returns the same filename as created at TPL\_APPLICATION. | 1. Call Open() to create file handle.  1. Call GetInfo() on file handle for EFI\_FILE\_INFO at TPL\_APPLICATION. The return filename should be the same as created. |
| 5.7.3.9.8 | 0x93c186a6, 0x4e31, 0x4395, 0x87, 0x1a, 0x90, 0xcc, 0x91, 0xa2, 0x2f, 0xc2 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() on file handle for EFI\_FILE\_INFO returns the same filename as created at TPL\_CALLBACK. | 1. Call Open() to create file handle.  1. Call GetInfo() on file handle for EFI\_FILE\_INFO at TPL\_CALLBACK. The return filename should be the same as created filename. |
| 5.7.3.9.9 | 0x35187534, 0xba64, 0x4be4, 0xaa, 0x9c, 0xc2, 0x07, 0x53, 0x12, 0x0f, 0x57 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_INFO from root returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call GetInfo() to retrieve EFI\_FILE\_INFO from root. The return code should be EFI\_SUCCESS. |
| 5.7.3.9.10 | 0xcd2e69ad, 0xe1ce, 0x42ea, 0x80, 0xfe, 0xfc, 0x2b, 0x1e, 0x72, 0xb0, 0x3b | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_INFO from root returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call GetInfo() to retrieve EFI\_FILE\_INFO from root. The return code should be EFI\_SUCCESS. |
| 5.7.3.9.11 | 0x7fc5deb9, 0xf216, 0x462e, 0xbf, 0x6b, 0xa3, 0x02, 0x6f, 0x13, 0x88, 0xea | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() on root handle for EFI\_FILE\_INFO returns the same buffer size as FileInfo->Size at TPL\_APPLICATION | 1. Call GetInfo() on root handle for EFI\_FILE\_INFO at TPL\_APPLICATION. The return buffer size should be the same as FileInfo->Size. |
| 5.7.3.9.12 | 0x8d390587, 0xe4ff, 0x4c55, 0xa1, 0x55, 0xba, 0x80, 0x7a, 0x19, 0xbe, 0xf1 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() on root handle for EFI\_FILE\_INFO returns the same buffer size as FileInfo->Size at TPL\_CALLBACK. | 1. Call GetInfo() on root handle for EFI\_FILE\_INFO at TPL\_CALLBACK. The return buffer size should be the same as FileInfo->Size. |
| 5.7.3.9.13 | 0xc01d216d, 0x9fdf, 0x4504, 0x99, 0x61, 0x3f, 0x4a, 0x08, 0xb7, 0x61, 0x43 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO. The return code should be EFI\_SUCCESS. |
| 5.7.3.9.14 | 0x04ae8ab0, 0xe2d6, 0x46e6, 0x98, 0x35, 0x63, 0x14, 0x21, 0x09, 0x5f, 0xac | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO returns EFI\_SUCCESS at TPL\_CALLBACK | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO. The return code should be EFI\_SUCCESS. |
| 5.7.3.9.15 | 0x75cd35a8, 0x8f56, 0x441d, 0x8a, 0x4a, 0x2a, 0xcd, 0x8a, 0x79, 0xea, 0x01 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO returns correct buffer size of EFI\_FILE\_SYSTEM\_INFO structure at TPL\_APPLICATION. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO. It should return correct buffer size of EFI\_FILE\_SYSTEM\_INFO structure. |
| 5.7.3.9.16 | 0xe4f4f6a2, 0x7538, 0x4c79, 0xaa, 0x3c, 0x67, 0x18, 0x4e, 0xc7, 0x0e, 0x16 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO returns correct buffer size of EFI\_FILE\_SYSTEM\_INFO structure at TPL\_CALLBACK | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO It should return correct buffer size of EFI\_FILE\_SYSTEM\_INFO structure. |
| 5.7.3.9.17 | 0x59afd349, 0xf5a1, 0x4052, 0x9b, 0xb9, 0x22, 0x51, 0x24, 0x0f, 0xe3, 0x47 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO returns correct file system info set by SetInfo() at TPL\_APPLICATION. | 1. Call SetInfo().  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO. It should return correct file system info set by SetInfo(). |
| 5.7.3.9.18 | 0xbe4e594f, 0x43c4, 0x42fc, 0xbe, 0x9e, 0xdc, 0xb7, 0xa8, 0x5a, 0x76, 0x7d | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO returns correct filesystem info set by SetInfo() at TPL\_CALLBACK | 1. Call SetInfo().  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO It should return returns correct file system info set by SetInfo(). |
| 5.7.3.9.19 | 0x4e8fa0c4, 0x95bc, 0x415b, 0x93, 0x65, 0x12, 0x11, 0xea, 0x40, 0x6b, 0xad | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO after create new file and free space decreases at TPL\_APPLICATION. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO and record free space number.  2. Call Open() to create new file.  3. Call GetInfo() again. Free space should decrease. |
| 5.7.3.9.20 | 0x9fa8a442, 0x572f, 0x4d04, 0xba, 0x0d, 0x17, 0x17, 0x72, 0x8d, 0x7e, 0x27 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO after create new file and free space decreases at TPL\_CALLBACK. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO and record free space number.  2. Call Open() to create new file.  3. Call GetInfo() again. Free space should decrease. |
| 5.7.3.9.21 | 0x2970bb0b, 0xb080, 0x48a9, 0x93, 0x64, 0x5e, 0x78, 0xbe, 0xb3, 0xf1, 0x02 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL. The return code should be EFI\_SUCCESS. |
| 5.7.3.9.22 | 0xf5fe94d3, 0x0269, 0x44ff, 0xb1, 0x3b, 0x23, 0x63, 0xd0, 0x33, 0xfe, 0xd5 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL. The return code should be EFI\_SUCCESS. |
| 5.7.3.9.23 | 0xa5d8f95a, 0x5bba, 0x4f1b, 0x83, 0x35, 0x12, 0x3e, 0x29, 0x6e, 0xda, 0xb2 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO. The return code should be EFI\_SUCCESS. |
| 5.7.3.9.24 | 0xf8dea2ab, 0xef13, 0x4544, 0xbd, 0x76, 0x42, 0xad, 0x6c, 0xd6, 0x17, 0x96 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO. The return code should be EFI\_SUCCESS. |
| 5.7.3.9.25 | 0x22837b7c, 0x46fc, 0x4439, 0x95, 0x3b, 0xb0, 0x18, 0xce, 0xd3, 0xd7, 0x67 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() returns the same volume label for EFI\_FILE\_SYSTEM\_INFO and EFI\_FILE\_SYSTEM\_VOLUME\_LABEL at TPL\_APPLICATION. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL.  3. They should return the same volume label. |
| 5.7.3.9.26 | 0x0772aef8, 0x1c09, 0x47e9, 0x83, 0xef, 0x76, 0xaa, 0x3d, 0x21, 0xfa, 0xa4 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() returns the same volume label for EFI\_FILE\_SYSTEM\_INFO and EFI\_FILE\_SYSTEM\_VOLUME\_LABEL at TPL\_CALLBACK. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL.  3. They should return the same volume label. |
| 5.7.3.9.27 | 0xfeb18200, 0x0904, 0x46cb, 0x81, 0x2b, 0x1e, 0xea, 0x00, 0xc3, 0x29, 0xc3 | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() with unsupported info type for the file returns EFI\_UNSUPPORTED. | 1. Call GetInfo() to retrieve unsupported info type for the file. The return code should be EFI\_UNSUPPORTED. |
| 5.7.3.9.28 | 0xdbdc09cc, 0x03d3, 0x4d56, 0x88, 0x76, 0xab, 0xa1, 0x3b, 0xf6, 0x68, 0xae | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_INFO with too small of a buffer returns EFI\_BUFFER\_TOO\_SMALL. | 1. Call GetInfo() to retrieve EFI\_FILE\_INFO with too small of a buffer. The return code should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.7.3.9.29 | 0x50e087ce, 0x802d, 0x46de, 0xa9, 0x13, 0x29, 0xa1, 0x8d, 0x2c, 0xc2, 0xff | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO with too small of a buffer returns EFI\_BUFFER\_TOO\_SMALL. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO with too small of a buffer. The return code should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.7.3.9.30 | 0x7a60bd66, 0x3b1e, 0x4818, 0xa1, 0x4b, 0xf8, 0x65, 0xf2, 0xc4, 0x76, 0x4e | EFI\_FILE\_PROTOCOL.GetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL with too small of a buffer returns EFI\_BUFFER\_TOO\_SMALL. | 1. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL with too small of a buffer. The return code should be EFI\_BUFFER\_TOO\_SMALL. |

### SetInfo()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.10.1 | 0x5eb09d11, 0x22ee, 0x43f7, 0xa6, 0xc1, 0x95, 0x92, 0xb5, 0x04, 0x70, 0xe7 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI\_FILE\_INFO returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call SetInfo() on file handle to set EFI\_FILE\_INFO at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.10.2 | 0x19f9c6f4, 0x2b6d, 0x4eb3, 0x80, 0xfb, 0x25, 0x55, 0x58, 0xf8, 0x47, 0x2f | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI\_FILE\_INFO returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call SetInfo() on file handle to set EFI\_FILE\_INFO at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.10.3 | 0x99f2a97f, 0xb249, 0x4cc3, 0xa4, 0x50, 0x56, 0x51, 0x7c, 0x2a, 0xfb, 0x35 | EFI\_FILE\_PROTOCOL.SetInfo – GetInfo() on file handle to retrieve EFI\_FILE\_INFO returns values set by SetInfo() at TPL\_APPLICATION. | 1. Call SetInfo() on file handle to set EFI\_FILE\_INFO.  2. Call GetInfo() to retrieve EFI\_FILE\_INFO. It should return the values set by SetInfo(). |
| 5.7.3.10.4 | 0x26615965, 0xe6b3, 0x43cb, 0x90, 0xb1, 0xcb, 0x00, 0x42, 0xe3, 0x34, 0xc5 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() on file handle to retrieve EFI\_FILE\_INFO returns values set by SetInfo() at TPL\_CALLBACK. | 1. Call SetInfo() on file handle to set EFI\_FILE\_INFO.  2. Call GetInfo() to retrieve EFI\_FILE\_INFO. It should return the values set by SetInfo(). |
| 5.7.3.10.5 | 0xb46741e9, 0x3545, 0x4b0e, 0x80, 0x12, 0xc9, 0x56, 0x6a, 0x7f, 0x83, 0xc5 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI\_FILE\_INFO returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call SetInfo() on file handle to set EFI\_FILE\_INFO at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.10.6 | 0xa1cc0c27, 0x55dc, 0x4cd8, 0x96, 0xe8, 0x57, 0x1b, 0x65, 0xc1, 0xdb, 0xde | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI\_FILE\_INFO returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call SetInfo() on file handle to set EFI\_FILE\_INFO at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.10.7 | 0x16494a12, 0xfc45, 0x4e30, 0x91, 0xac, 0x88, 0x1c, 0x9c, 0x88, 0xae, 0x4b | EFI\_FILE\_PROTOCOL.SetInfo – GetInfo() on file handle to retrieve EFI\_FILE\_INFO returns values set by SetInfo() at TPL\_APPLICATION. | 1. Call SetInfo() on file handle to set EFI\_FILE\_INFO.  2. Call GetInfo() to retrieve EFI\_FILE\_INFO. It should return the values set by SetInfo(). |
| 5.7.3.10.8 | 0xd843eacb, 0x2468, 0x4d4b, 0xa3, 0x51, 0xbf, 0x41, 0xd5, 0xdd, 0x9a, 0x16 | EFI\_FILE\_PROTOCOL.SetInfo – GetInfo() on file handle to retrieve EFI\_FILE\_INFO returns values set by SetInfo() at TPL\_CALLBACK. | 1. Call SetInfo() on file handle to set EFI\_FILE\_INFO.  2. Call GetInfo() to retrieve EFI\_FILE\_INFO. It should return the values set by SetInfo(). |
| 5.7.3.10.9 | 0x4be420a1, 0xd7e7, 0x4327, 0x8e, 0x63, 0x59, 0x41, 0xef, 0x4b, 0xfd, 0x2a | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI\_FILE\_SYSTEM\_INFO returns EFI\_SUCCESS at TPL\_APPLICATION | 1. Call SetInfo() on file handle to set EFI\_FILE\_SYSTEM\_INFO at TPL\_APPLICATION. The return code should be EFI\_SUCCESS. |
| 5.7.3.10.10 | 0x57880cd6, 0x6eb1, 0x40b5, 0xa3, 0xef, 0x28, 0x26, 0x94, 0x9a, 0xf8, 0x9c | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() on file handle to set EFI\_FILE\_SYSTEM\_INFO returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call SetInfo() on file handle to set EFI\_FILE\_SYSTEM\_INFO at TPL\_CALLBACK. The return code should be EFI\_SUCCESS. |
| 5.7.3.10.11 | 0x0cf2c5c5, 0xd976, 0x4fd4, 0x85, 0x07, 0x0a, 0x81, 0x88, 0x62, 0x45, 0x78 | EFI\_FILE\_PROTOCOL.SetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO returns volume label set by SetInfo() at TPL\_APPLICATION. | 1. Call SetInfo() for EFI\_FILE\_SYSTEM\_INFO to set volume label.  2. Call GetInfo() to to retrieve EFI\_FILE\_SYSTEM\_INFO. The return volume label should be the same as set by SetInfo(). |
| 5.7.3.10.12 | 0xc68c8288, 0x020f, 0x460f, 0x81, 0xf8, 0x67, 0x35, 0x10, 0xd1, 0xfe, 0x6a | EFI\_FILE\_PROTOCOL.SetInfo – GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO returns volume label set by SetInfo() at TPL\_CALLBACK. | 1. Call SetInfo() for EFI\_FILE\_SYSTEM\_INFO to set volume label.  2. Call GetInfo() to to retrieve EFI\_FILE\_SYSTEM\_INFO. The return volume label should be the same as set by SetInfo(). |
| 5.7.3.10.13 | 0x12b68173, 0x7c8d, 0x4023, 0xaf, 0xcc, 0xf1, 0xc6, 0xbe, 0xb6, 0x1c, 0xef | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to set EFI\_FILE\_SYSTEM\_VOLUME\_LABEL returns EFI\_SUCCESS at TPL\_APPLICATION. | 1. Call SetInfo() on file handle to set EFI\_FILE\_SYSTEM\_VOLUME\_LABEL. The return code should be EFI\_SUCCESS. |
| 5.7.3.10.14 | 0x6e869806, 0x1bc2, 0x40d5, 0xb3, 0x02, 0x61, 0xf8, 0xce, 0x56, 0xeb, 0xfe | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to set EFI\_FILE\_SYSTEM\_VOLUME\_LABEL returns EFI\_SUCCESS at TPL\_CALLBACK. | 1. Call SetInfo() on file handle to set EFI\_FILE\_SYSTEM\_VOLUME\_LABEL. The return code should be EFI\_SUCCESS. |
| 5.7.3.10.15 | 0x725364f6, 0x6a23, 0x424a, 0x82, 0xaf, 0xd0, 0xd0, 0x47, 0xd3, 0xb8, 0x08 | EFI\_FILE\_PROTOCOL.SetInfo – GetInfo() for EFI\_FILE\_SYSTEM\_VOLUME\_LABEL returns the same volume label as set by SetInfo() at TPL\_APPLICATION. | 1. Call SetInfo() on file handle to set EFI\_FILE\_SYSTEM\_VOLUME\_LABEL.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL. The return volume label should be the same as set by SetInfo(). |
| 5.7.3.10.16 | 0xcdab6fd9, 0x93aa, 0x4820, 0xb1, 0xa1, 0x71, 0xea, 0x3e, 0x7f, 0xab, 0x26 | EFI\_FILE\_PROTOCOL.SetInfo – GetInfo() for EFI\_FILE\_SYSTEM\_VOLUME\_LABEL returns the same volume label as set by SetInfo() at TPL\_CALLBACK. | 1. Call SetInfo() on file handle to set EFI\_FILE\_SYSTEM\_VOLUME\_LABEL.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_VOLUME\_LABEL. The return volume label should be the same as set by SetInfo(). |
| 5.7.3.10.17 | 0xf700f5f8, 0xecac, 0x45fb, 0x9d, 0x2d, 0x34, 0xe9, 0x46, 0x66, 0x07, 0x38 | EFI\_FILE\_PROTOCOL.SetInfo – GetInfo() for EFI\_FILE\_SYSTEM\_INFO returns the same volume label as set by SetInfo() at TPL\_APPLICATION. | 1. Call SetInfo() on file handle to set EFI\_FILE\_SYSTEM\_VOLUME\_LABEL.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO. The return volume label should be the same as set by SetInfo(). |
| 5.7.3.10.18 | 0x384840cd, 0x9a3f, 0x44c3, 0x87, 0xd8, 0xcd, 0xd9, 0xab, 0xd2, 0x17, 0x96 | EFI\_FILE\_PROTOCOL.SetInfo – GetInfo() for EFI\_FILE\_SYSTEM\_INFO returns the same volume label as set by SetInfo() at TPL\_CALLBACK. | 1. Call SetInfo() on file handle to set EFI\_FILE\_SYSTEM\_VOLUME\_LABEL.  2. Call GetInfo() to retrieve EFI\_FILE\_SYSTEM\_INFO. The return volume label should be the same as set by SetInfo(). |
| 5.7.3.10.19 | 0x2a58594e, 0xd06a, 0x4f44, 0xa2, 0x6e, 0xa3, 0x49, 0x36, 0xde, 0x05, 0xef | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() with unsupported info type returns EFI\_UNSUPPORTED. | 1. Call SetInfo() with unsupported info type. The return code should be EFI\_UNSUPPORTED. |
| 5.7.3.10.20 | 0x164feeba, 0xf3ed, 0x482a, 0x83, 0xac, 0x89, 0x48, 0x0a, 0x1d, 0x9a, 0xc9 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() with the read-only opened file handle to change file size returns EFI\_ACCESS\_DENIED. | 1. Call Open() to open file handle in read-only mode.  2. Call SetInfo() to change file size. The return code should be EFI\_ACCESS\_DENIED. |
| 5.7.3.10.21 | 0x1a74e8f3, 0x62ad, 0x47ef, 0x92, 0xe6, 0x6d, 0x47, 0x23, 0x21, 0xd2, 0xb0 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() with the read-only opened file handle to change file name returns EFI\_ACCESS\_DENIED. | 1. Call Open() to open file handle in read-only mode.  2. Call SetInfo() to change file name. The return code should be EFI\_ACCESS\_DENIED. |
| 5.7.3.10.22 | 0x75c4d3e4, 0x17fa, 0x4f02, 0xb1, 0x15, 0x72, 0x0c, 0x0f, 0x1a, 0xe2, 0xe1 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to set EFI\_FILE\_INFO with too small of a buffer returns EFI\_BUFFER\_TOO\_SMALL. | 1. Call SetInfo() to set EFI\_FILE\_INFO with too small of a buffer. The return code should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.7.3.10.23 | 0x36d0ed31, 0x21f0, 0x48c2, 0x89, 0x74, 0x6b, 0x6e, 0xca, 0x41, 0x20, 0x3c | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to set EFI\_FILE\_SYSTEM\_INFO with too small of a buffer returns EFI\_BUFFER\_TOO\_SMALL. | 1. Call SetInfo() to set EFI\_FILE\_SYSTEM\_INFO with too small of a buffer. The return code should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.7.3.10.24 | 0xc7bfe9bf, 0x92bf, 0x4301, 0x82, 0x17, 0x75, 0x66, 0x2e, 0xa5, 0x24, 0x37 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to set EFI\_FILE\_SYSTEM\_VOLUME\_LABEL with too small of a buffer returns EFI\_BUFFER\_TOO\_SMALL. | 1. Call SetInfo() to set EFI\_FILE\_SYSTEM\_VOLUME\_LABEL with too small of a buffer. The return code should be EFI\_BUFFER\_TOO\_SMALL. |
| 5.7.3.10.25 | 0x86eb2a14, 0x668a, 0x4ad6, 0xbc, 0x8a, 0x56, 0x67, 0x79, 0x09, 0x94, 0xe5 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to set illegal size, physical size and invalid attribute of EFI\_FILE\_INFO for the file does not change the original settings. | 1. Call GetInfo() to store original size, physical size and attribute of EFI\_FILE\_INFO of the file.  2. Call SetInfo() to set illegal size, physical size and invalid attribute of EFI\_FILE\_INFO for the file.  3. Call GetInfo() again to get current size, physical size and attribute of EFI\_FILE\_INFO of the file. It should return the same value as original settings. |
| 5.7.3.10.26 | 0x63c55abc, 0x16d6, 0x4ac9, 0xb7, 0x8c, 0x45, 0x44, 0xbe, 0x70, 0x81, 0x54 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to set illegal filename of EFI\_FILE\_INFO for the file does not change the filename. | 1. Call Open() to create file handle with valid filename.  2. Call SetInfo() to set illegal filename of EFI\_FILE\_INFO for the file.  3. Call GetInfo() to get current filename of EFI\_FILE\_INFO of the file. It should return the same value as original filename. |
| 5.7.3.10.27 | 0x7ba04c1e, 0xcd95, 0x4a3c, 0xa3, 0xba, 0xa5, 0x82, 0xf1, 0x6c, 0x46, 0xbc | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to set illegal size, physical size and invalid attribute of EFI\_FILE\_INFO for the directory does not change the original settings. | 1. Call GetInfo() to store original size, physical size and attribute of EFI\_FILE\_INFO of the directory.  2. Call SetInfo() to set illegal size, physical size and invalid attribute of EFI\_FILE\_INFO for the directory.  3. Call GetInfo() again to get current size, physical size and attribute of EFI\_FILE\_INFO of the directory. It should return the same value as original settings. |
| 5.7.3.10.28 | 0x6a09725c, 0x51c7, 0x44f3, 0x85, 0x74, 0xda, 0xc3, 0x6e, 0xc7, 0x0f, 0x86 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to set illegal filename of EFI\_FILE\_INFO for the directory does not change the filename. | 1. Call Open() to create directory handle with valid filename.  2. Call SetInfo() to set illegal filename of EFI\_FILE\_INFO for the directory.  3. Call GetInfo() to get current filename of EFI\_FILE\_INFO of the directory. It should return the same value as original filename. |
| 5.7.3.10.29 | 0x5bef76ad, 0x4a40, 0x401c, 0x83, 0xd3, 0x9c, 0x73, 0x72, 0x3b, 0xb4, 0x58 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to change all the fields except the **VolumeLabel** of the EFI\_FILE\_SYSTEM\_INFO does not change the original settings. | 1. Call GetInfo() to store original value of fields of EFI\_FILE\_SYSTEM\_INFO.  2. Call SetInfo() to change all the fields except the **VolumeLabel** of the EFI\_FILE\_SYSTEM\_INFO.  3. Call GetInfo() again to get current value of fields of EFI\_FILE\_SYSTEM\_INFO. It should return the same value as original settings. |
| 5.7.3.10.30 | 0x4857f42c, 0xb998, 0x4667, 0x8f, 0x11, 0xdb, 0xed, 0x7a, 0xd5, 0xe0, 0xac | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to change file attribute to read-only returns EFI\_SUCCESS. | 1. Call Open() to create a file handle.  2. Call SetInfo() to set file attribute to read-only. The return code should be EFI\_SUCCESS. |
| 5.7.3.10.31 | 0xa9df1e64, 0xe769, 0x4d16, 0xa0, 0xd5, 0xb5, 0x59, 0xce, 0x90, 0xcf, 0x2b | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to change file attribute to read-only changes the file attribute from read-write to read-only. | 1. Call Open() to create a file handle.  2. Call SetInfo() to set file attribute to read-only.  3. Call GetInfo() to get file attribute. It should be read-only. |
| 5.7.3.10.32 | 0xb5481965, 0xf157, 0x4037, 0x89, 0xab, 0x14, 0x6e, 0xa6, 0xc9, 0x44, 0x1a | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to change file attribute from read-only to read-write returns EFI\_SUCCESS. | 1. Call Open() with read-only open mode to open a read-only file.  2. Call SetInfo() to set file attribute to read-write. The return code should be EFI\_SUCCESS. |
| 5.7.3.10.33 | 0x3535af93, 0x32df, 0x44bb, 0xa0, 0xaf, 0xce, 0x2d, 0x38, 0xe7, 0xd6, 0xfb | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo() to change file attribute to read-write changes the file attribute from read-only to read-write. | 1. Call Open() with read-only open mode to open a read-only file.  2. Call SetInfo() to set file attribute to read-write.  3. Call GetInfo() to get file attribute. It should be read-write. |
| 5.7.3.10.34 | 0x8821c678, 0xde6e, 0x49bf, 0x94, 0xcd, 0x9f, 0x4b, 0xa0, 0xa2, 0x15, 0x22 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo()to change  file name to an existing file name returns EFI\_ACCESS\_DENIED | 1. Call Open() to create two file handle.  2. Call SetInfo()to set one file name to the other file name. The return code should be EFI\_ACCESS\_DENIED. |
| 5.7.3.10.35 | 0x69afc35a, 0xcf85, 0x4365, 0xac, 0xca, 0xa5, 0x3c, 0x48, 0xcb, 0xd3, 0x51 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo()to change  file system volume info on a read-only media returns EFI\_WRITE\_PROTECTED | 1. Get system volume info to see if it is a read-only media  2. Call **SetInfo()** to change file system volume info. The return code should be EFI\_WRITE\_PROTECTED |
| 5.7.3.10.36 | 0x669bf242, 0xd3ca, 0x4b73, 0xa6, 0xdd, 0x8b, 0x2a, 0xf3, 0xfb, 0xa6, 0x28 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo()to change  file system volume label on a read-only media returns EFI\_WRITE\_PROTECTED | 1. Get system volume info to see if it is a read-only media  2. Call SetInfo()to change file system volume label. The return code should be EFI\_WRITE\_PROTECTED |
| 5.7.3.10.37 | 0x33218d68, 0x5245, 0x4bab, 0x9c, 0x1d, 0xc, 0x4b, 0xca, 0xd9, 0x4, 0x87 | EFI\_FILE\_PROTOCOL.SetInfo – SetInfo()to change  file info on a read-only media returns EFI\_WRITE\_PROTECTED | 1. Get system volume info to see if it is a read-only media  2. Call SetInfo()to change file info of root dir. The return code should be EFI\_WRITE\_PROTECTED |

### OpenEx()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.11.1 | 0xb6cff935, 0x32ef, 0x4865,0x9e, 0xd9, 0x09, 0x62, 0x87, 0xf2, 0x2a, 0x66 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx()to async create file under root directory with pure filename returnsEFI\_SUCCESS. | Async call OpenEx()to create file under root directory with pure filename, the return status should be EFI\_SUCCESS. The status in OpenFileFinishList should be EFI\_SUCCESS. |
| 5.7.3.11.2 | 0x5e767a63, 0x577a, 0x4628,0xb6, 0xe9, 0x91, 0xb7, 0xd9, 0xaa, 0x05, 0xcb | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx()to async create file under root directory with pure filename returnsEFI\_SUCCESS. | Async call OpenEx()to create file under root directory with pure filename, the return status should be EFI\_SUCCESS.  The OpenFileFailList should be empty. |
| 5.7.3.11.3 | 0x611a8daf, 0x274c, 0x4bd5,0xa7, 0xba, 0xc1, 0x85, 0x43, 0xd3, 0x7f, 0x74 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create file under root directory with pure filename returnsEFI\_SUCCESS. | Async call OpenEx()to create file under root directory with pure filename, the return status should be EFI\_SUCCESS.  The OpenFileExecuteList should be empty. |
| 5.7.3.11.4 | 0x0a6985e4, 0xfe17, 0x4740,0x95, 0x7a, 0xe9, 0xc0, 0x5b, 0x45, 0x02, 0xe0 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx()to sync create file under root directory with pure filename returnsEFI\_SUCCESS. | Sync call OpenEx()to create file under root directory with pure filename, the return status should be EFI\_SUCCESS. |
| 5.7.3.11.5 | 0x0c61f052, 0x2ae3, 0x4219,0xad, 0x79, 0x4b, 0xdd, 0x95, 0xc1, 0x78, 0xc3 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx()to async create directory under root directory returnsEFI\_SUCCESS. | Async call OpenEx()to create subdirectory under root directory, the return status should be EFI\_SUCCESS. |
| 5.7.3.11.6 | 0x41087c41, 0xb9a9, 0x4943,0xb8, 0x22, 0x9f, 0x9b, 0x41, 0x78, 0xa6, 0x49 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx()to async create file under root directory with filename containing sub directory name returnsEFI\_SUCCESS. | Async call OpenEx()to create file under root directory with filename containing sub directory name, the return status should be EFI\_SUCCESS. |
| 5.7.3.11.7 | 0x6277ccac, 0x481c, 0x4cb2, 0xac, 0x96, 0x89, 0x96, 0x79, 0xf2, 0xa9, 0x19 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create directory under root directory returns EFI\_SUCCESS. | Async call OpenEx()to create directory under root directory, the return status should be EFI\_SUCCESS. The OpenDirFailList should be empty. |
| 5.7.3.11.8 | 0x518c70d5, 0x4070, 0x4b81,0x9d, 0xb3, 0xcb, 0x20, 0xd6, 0x7f, 0x11, 0x1c | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx()to async create file under root directory with filename containing sub directory name returnsEFI\_SUCCESS. | Async call OpenEx()to create file under root directory with filename containing sub directory name, the return status should be EFI\_SUCCESS. The OpenFileFailList should be empty. |
| 5.7.3.11.9 | 0x44fa0576, 0x08cd, 0x48c2,0x9b, 0x71, 0x5f, 0x63, 0xc2, 0xb3, 0x97, 0x10 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create directory under root directory returnsEFI\_SUCCESS. | Async call OpenEx() to create directory under root directory, the return status should be EFI\_SUCCESS. The OpenDirExecuteList should be empty. |
| 5.7.3.11.10 | 0xef745935, 0x0937, 0x4b11,0xa7, 0xca, 0x65, 0xaf, 0x0b, 0xf0, 0x45, 0x44 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx()to async create file under root directory with filename containing sub directory name returnsEFI\_SUCCESS. | Async call OpenEx() to create file under root directory with filename containing sub directory name, the return status should be EFI\_SUCCESS. The OpenFileExecuteList should be empty. |
| 5.7.3.11.11 | 0x3c64e927, 0x68e7, 0x4668,0xae, 0xa8, 0xc2, 0xc7, 0xdc, 0x15, 0x0c, 0x3f | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to sync create directory under root directory returns EFI\_SUCCESS. | Sync call OpenEx()to create directory under root directory, the return status should be EFI\_SUCCESS. |
| 5.7.3.11.12 | 0x233a928b, 0x8f5d, 0x483a,0xab, 0x03, 0x2d, 0x03, 0xf1, 0xa3, 0xdc, 0x26 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to sync create file under root directory with filename containing sub directory name returns EFI\_SUCCESS. | Sync call OpenEx() to create file under root directory with filename containing sub directory name, the return status should be EFI\_SUCCESS. |
| 5.7.3.11.13 | 0x959a9093, 0xa975, 0x42a9,0x9b, 0x83, 0x32, 0x4a, 0x79, 0xca, 0x2f, 0x1b | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create file under sub directory with pure name returns EFI\_SUCCESS. | Call Open() to create directory(dir1)  under root.  Async call OpenEx() to create file with  pure file name under directory(dir1) , the return status should be EFI\_SUCCESS. |
| 5.7.3.11.14 | 0x43ad5688, 0xbc02, 0x4870,0xb8, 0x85, 0x02, 0x86, 0xdd, 0x54, 0xb2, 0x76 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create file under sub directory with pure name returns EFI\_SUCCESS. | Call Open() to create directory(dir1) under root.  Async call OpenEx() to create file with pure file name under directory(dir1) , the return status should be EFI\_SUCCESS. The OpenFileFailList should be empty. |
| 5.7.3.11.15 | 0x90908639, 0x141f, 0x4632,0x85, 0xca, 0x7d, 0x6e, 0x83, 0xe5, 0x57, 0x47 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create file under sub directory with pure name returns EFI\_SUCCESS. | Call Open() to create directory(dir1)  under root.  Async call OpenEx() to create file with  pure file name under directory(dir1) , the return status should be EFI\_SUCCESS. The OpenFileExecuteList should be empty. |
| 5.7.3.11.16 | 0x8eb7f8cc, 0x6d0d, 0x4c10, 0xbd, 0x94, 0xdc, 0x32, 0x7d, 0x2e, 0x6d, 0x3d | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to sync create file under sub directory with pure name returns EFI\_SUCCESS. | Call Open() to create directory(dir1)  under root.  Sync call OpenEx() to create file with  pure file name under directory(dir1) , the return status should be EFI\_SUCCESS. |
| 5.7.3.11.17 | 0x3431780c, 0x56da, 0x4628,0x86, 0xa2, 0xa3, 0x08, 0xf2, 0xe9, 0x88, 0x27 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create file under sub directory and filename containing sub directory name returns EFI\_SUCCESS. | Call Open() to create directory(dir1)  under root.  Call Open() to create directory(dir2)  under dir1.  Async call OpenEx() to create file with  file name containing sub directory name (dir2/pure name) under directory(dir1) , the return status should be EFI\_SUCCESS. |
| 5.7.3.11.18 | 0x7d9eacf0, 0x0167, 0x4ef7, 0xa7, 0xf2, 0x31, 0xb5, 0x3e, 0xc4, 0xcb, 0x8a | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create file under sub directory and filename containing sub directory name returns EFI\_SUCCESS. | Call Open() to create directory(dir1)  under root.  Call Open() to create directory(dir2)  under dir1.  Async call OpenEx() to create file with  file name containing sub directory name (dir2/pure name) under directory(dir1) , the return status should be EFI\_SUCCESS. The OpenFileFailList should be empty. |
| 5.7.3.11.19 | 0xf9dad61f, 0xfc35, 0x4fd6, 0x86, 0x0b, 0x7b, 0x8b, 0x2e, 0xbf, 0x89, 0x63 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create file under sub directory and filename containing sub directory name returns EFI\_SUCCESS. | Call Open() to create directory(dir1)  under root.  Call Open() to create directory(dir2)  under dir1.  Async call OpenEx() to create file with  file name containing sub directory name (dir2/pure name) under directory(dir1) , the return status should be EFI\_SUCCESS. The OpenFileExecuteList should be empty. |
| 5.7.3.11.20 | 0xf87622cf, 0x13c6, 0x412e,0x86, 0xa6, 0x8e, 0x7f, 0xf2, 0x63, 0x0a, 0x8e | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to sync create file under sub directory and filename containing sub directory name returns EFI\_SUCCESS. | Call Open() to create directory(dir1)  under root.  Call Open() to create directory(dir2)  under dir1.  Sync call OpenEx() to create file with  file name containing sub directory name (dir2/pure name) under directory(dir1) , the return status should be EFI\_SUCCESS. |
| 5.7.3.11.21 | 0xbfc2a163, 0xe8d5, 0x45df, 0x8f, 0x6b, 0x01, 0x0a, 0xee, 0x48, 0x89, 0xc0 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create file with sub directory handle and filename containing absolute file path returns EFI\_SUCCESS. | Call Open() to create directory(dir1)  under root.  Call Open() to create directory(dir2)  under dir1.  Async call OpenEx() to create file  containing absolute file path (/dir1/dir2/pure name) under sub directory(dir2), the return status should be EFI\_SUCCESS. |
| 5.7.3.11.22 | 0x8ffd05e8, 0xaa76, 0x4fcb, 0x93, 0xe4, 0x19, 0xa2, 0x0c, 0x2b, 0xa9, 0x04 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create file with sub directory handle and filename containing absolute file path returns EFI\_SUCCESS. | Call Open() to create directory(dir1)  under root.  Call Open() to create directory(dir2)  under dir1.  Async call OpenEx() to create file  containing absolute file path (/dir1/dir2/pure name) under sub directory(dir2), the return status should be EFI\_SUCCESS. The OpenFileFailList should be empty. |
| 5.7.3.11.23 | 0x41fe9684, 0x113b, 0x415f, 0xaf, 0xbf, 0xee, 0x48, 0x10, 0x8a, 0x70, 0xc2 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async create file with sub directory handle and filename containing absolute file path returns EFI\_SUCCESS. | Call Open() to create directory(dir1)  under root.  Call Open() to create directory(dir2)  under dir1.  Async call OpenEx() to create file  containing absolute file path (/dir1/dir2/pure name) under sub directory(dir2), the return status should be EFI\_SUCCESS. The OpenFileExecuteList should be empty. |
| 5.7.3.11.24 | 0xd5c326a3, 0x07ad, 0x490e,0x9b, 0xdc, 0xa8, 0xe3, 0x4d, 0x7a, 0xae, 0x8a | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to sync create file with sub directory handle and filename containing absolute file path returns EFI\_SUCCESS. | Call Open()to create directory(dir1)  under root.  Call Open()to create directory(dir2)  under dir1.  Sync call OpenEx()to create file  containing absolute file path (/dir1/dir2/pure name) under sub directory(dir2), the return status should be EFI\_SUCCESS. |
| 5.7.3.11.25 | 0x55825138, 0x793d, 0x4aaa, 0xab, 0xcc, 0x4d, 0x4a, 0xbd, 0xb2, 0x17, 0xef | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file under root directory with pure filename returns EFI\_SUCCESS. | Call Open()to create file under root.  Call SetInfo()to set file size to 1.  Async call OpenEx() to open the file  Again, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.26 | 0xa4a53615, 0x7939, 0x4dcf, 0xbf, 0xb6, 0xc7, 0x4e, 0xe3, 0x3e, 0x93, 0x30 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file under root directory with pure filename returns EFI\_SUCCESS. | Call Open() to create file under root.  Call SetInfo() to set file size to 1.  Async call OpenEx() to open the file  Again, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.27 | 0x4014c563, 0x7c95, 0x4323,0xa2, 0xd1, 0xbb, 0x94, 0x26, 0x74, 0xc9, 0xa3 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file under root directory with pure filename returns EFI\_SUCCESS. | Call Open() to create file under root.  Call SetInfo() to set file size to 1.  Async call OpenEx() to open the file  Again, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.28 | 0x5a646037, 0xbe58, 0x41d8,0xb4, 0x91, 0x84, 0x03, 0xb9, 0xf8, 0xa7, 0x44 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to sync open the existing file under root directory with pure filename returns EFI\_SUCCESS. | Call Open()to create file under root.  Call SetInfo()to set file size to 1.  Sync call OpenEx()to open the existing file again, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.29 | 0xa398b24a, 0x568f, 0x4762,0xb1, 0xcb, 0x52, 0x25, 0xa7, 0x0e, 0x2f, 0x1f | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file under root directory with filename containing sub directory name returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create file under dir1.  Call SetInfo() to set file size to 1.  Async call OpenEx() to open the  existing file under root directory with filename containing sub directory name, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.30 | 0xbab0c3fc, 0x8630, 0x43bf, 0x97, 0x88, 0x6d, 0x96, 0xcd, 0x3a, 0x6e, 0x7c | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file under root directory with filename containing sub directory name returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create file under dir1.  Call SetInfo() to set file size to 1.  Async call OpenEx() to open the  existing file under root directory with filename containing sub directory name, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.31 | 0xffc5787b, 0x29a5, 0x4704,0x84, 0xd9, 0xd8, 0xb6, 0x6e, 0x62, 0x9c, 0xc2 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file under root directory with filename containing sub directory name returns EFI\_SUCCESS. | Call Open()to create directory(dir1).  Call Open()to create file under dir1.  Call SetInfo() to set file size to 1.  Async call OpenEx() to open the  existing file under root directory with filename containing sub directory name, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.32 | 0xe9d202ed, 0x2e34, 0x4686,0x9a, 0xe3, 0x9b, 0x41, 0x5b, 0xaa, 0xbc, 0x72 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to sync open the existing file under root directory with filename containing sub directory name returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create file under dir1.  Call SetInfo() to set file size to 1.  Sync call OpenEx() to open the  existing file under root directory with filename containing sub directory name, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.33 | 0x3c57480f, 0xc2f3, 0x4cee, 0xab, 0xef, 0x54, 0x8d, 0x69, 0x56, 0xae, 0x89 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file under sub directory with pure filename returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create file under dir1.  Call SetInfo() to set file size to 1.  Async call OpenEx() to open the  existing file under sub directory with pure filename, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.34 | 0x5850bc3c, 0x1b0f, 0x4bda,0x9e, 0x3c, 0x9c, 0x17, 0xf1, 0x9d, 0xf7, 0x53 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file under sub directory with pure filename returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create file under dir1.  Call SetInfo() to set file size to 1.  Async call OpenEx() to open the  existing file under sub directory with pure filename, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.35 | 0x13ce6d88, 0xd770, 0x470f, 0xb7, 0x3d, 0x60, 0x25, 0x18, 0xc2, 0xd2, 0xbf | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file under sub directory with pure filename returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open()to create file under dir1.  Call SetInfo() to set file size to 1.  Async call OpenEx() to open the  existing file under sub directory with pure filename, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.36 | 0xc2535525, 0xbe07, 0x4980,0xb9, 0x46, 0x7f, 0x87, 0x09, 0xe2, 0x12, 0xbe | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to sync open the existing file under sub directory with pure filename returns EFI\_SUCCESS. | Call Open()to create directory(dir1).  Call Open() to create file under dir1.  Call SetInfo() to set file size to 1.  Sync call OpenEx() to open the  existing file under sub directory with pure filename, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.37 | 0x7b0dcc35, 0xc3ea, 0x43cc, 0xac, 0xa9, 0x6a, 0x60, 0x1c, 0x3d, 0xe5, 0x45 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing sub directory name returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create sub  directory(dir2)under dir1.  Call Open() to create file under dir2.  Call SetInfo() to set file size to 1.  Async call OpenEx() to async open  the existing file with sub directory(dir1) handle and filename(/dir2/filename) containing sub directory name, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.38 | 0x8a6ef609, 0xe8dc, 0x40a2,0xb4, 0x18, 0xd0, 0xa4, 0xdf, 0x4d, 0x3f, 0xa3 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing sub directory name returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create sub  directory(dir2)under dir1.  Call Open() to create file under dir2.  Call SetInfo() to set file size to 1.  Async call OpenEx() to async open  the existing file with sub directory(dir1) handle and filename(/dir2/filename) containing sub directory name, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.39 | 0x5cfc5d39, 0x197c, 0x48dd, 0x9c, 0x7f, 0x98, 0x51, 0x64, 0x96, 0x79, 0xe7 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing sub directory name returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create sub  directory(dir2)under dir1.  Call Open()to create file under dir2.  Call SetInfo() to set file size to 1.  Async call OpenEx() to async open  the existing file with sub directory(dir1) handle and filename(/dir2/filename) containing sub directory name, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.40 | 0x1f3f5ccf, 0xdc02, 0x4200,0x81, 0xd0, 0x02, 0x34, 0x32, 0x60, 0xf2, 0xe5 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to sync open the existing file with sub directory handle and filename containing sub directory name returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create sub  directory(dir2)under dir1.  Call Open()to create file under dir2.  Call SetInfo() to set file size to 1.  Sync call OpenEx() to sync open  the existing file with sub directory(dir1) handle and filename(/dir2/filename) containing sub directory name, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.41 | 0x83351bef, 0x2368, 0x442e,0x89, 0xe6, 0xd2, 0xd5, 0xe9, 0xaf, 0x4a, 0x40 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing absolute file path returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create sub  directory(dir2) under dir1.  Call Open()to create file under dir2.  Call SetInfo() to set file size to 1.  Async call OpenEx() to async open  the existing file with sub directory handle and filename containing absolute file path, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.42 | 0x1e8c1e14, 0x47d8, 0x4a23,0xb2, 0xd6, 0x4b, 0xe0, 0x99, 0xf4, 0xa5, 0xdf | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing absolute file path returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create sub  directory(dir2) under dir1.  Call Open() to create file under dir2.  Call SetInfo() to set file size to 1.  Async call OpenEx() to async open  the existing file with sub directory handle and filename containing absolute file path, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.43 | 0x70486db6, 0x12f9, 0x4f6e, 0xa3, 0xf2, 0xed, 0xb4, 0x21, 0x27, 0x45, 0xbc | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async open the existing file with sub directory handle and filename containing absolute file path returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create sub  directory(dir2) under dir1.  Call Open() to create file under dir2.  Call SetInfo() to set file size to 1.  Async call OpenEx() to async open  the existing file with sub directory handle and filename containing absolute file path, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal  to 1.  Call SetInfo() & Write(), if the Open Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.44 | 0x69996cd2, 0xf087, 0x42e9,0xb7, 0xf6, 0x7c, 0x04, 0x18, 0x76, 0x36, 0xd7 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to sync open the existing file with sub directory handle and filename containing absolute file path returns EFI\_SUCCESS. | Call Open() to create directory(dir1).  Call Open() to create sub  directory(dir2) under dir1.  Call Open() to create file under dir2.  Call SetInfo() to set file size to 1.  Sync call OpenEx() to sync open  the existing file with sub directory handle and filename containing absolute file path, the return status should be EFI\_SUCCESS.  To get the file size, it should be equal to 1.  Call SetInfo() & Write(), if the Open  Mode is read-only, the return status should be EFI\_ACCESS\_DENIED. Otherwise, it should be EFI\_SUCCESS. |
| 5.7.3.11.45 | 0xad02d93d, 0xf2e8, 0x4f25, 0x93, 0xce, 0x94, 0x06, 0x77, 0xb6, 0xe1, 0xb2 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async & sync open with non-existent file name returns EFI\_NOT\_FOUND. | Async & Sync call OpenEx() to open with non-existent file name, the return status should be EFI\_NOT\_FOUND. |
| 5.7.3.11.46 | 0xcab7c260, 0xa290, 0x4845,0xb7, 0x03, 0xb1, 0x9f, 0xed, 0xf9, 0x84, 0xeb | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async & sync open with non-existent file path returns EFI\_NOT\_FOUND. | Async & Sync call OpenEx() to open with non-existent file path, the return status should be EFI\_NOT\_FOUND. |
| 5.7.3.11.47 | 0x33273ae, 0x2471, 0x4c08,0xb0, 0x8d, 0xeb, 0xd9, 0xdd, 0xbd, 0x57, 0x81 | EFI\_FILE\_PROTOCOL.OpenEx - OpenEx() to async & sync open with invalid open-mode returns EFI\_INVALID\_PARAMETER. | Async & Sync call OpenEx() to open with invalid open-mode, the return status should be EFI\_INVALID\_PARAMETER. |

### ReadEx

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.12.1 | 0xce038e00, 0x833c, 0x4b2e, 0x9e, 0x50, 0x79, 0xed, 0xc, 0x74, 0xf2, 0x50 | EFI\_FILE\_PROTOCOL.ReadEx - ReadEx() to async read data from a file returns EFI\_SUCCESS. | Call Open() to create a file.  Call Write() to write data to the  file.  Async Call ReadEx()from valid  setposition & ReadLength, the return status should be EFI\_SUCCESS and ReadLength should be equal to the Token’s BufferSize.  Call GetPosition()the  PositionAfterRead should be equal to the sum of SetPosition and ReadLength.  Compare the content of ReadBuffer  with the data set in step2. |
| 5.7.3.12.2 | 0x05857ebf, 0xc920, 0x474a, 0x97, 0x4d, 0x85, 0x8d, 0x83, 0x98, 0x81, 0x6f | EFI\_FILE\_PROTOCOL.ReadEx - ReadEx() to async read data from a file returns EFI\_SUCCESS. | Call Open() to create a file.  Call Write() to write data to the  file.  Async Call ReadEx()from valid  setposition & ReadLength, the return status should be EFI\_SUCCESS and ReadLength should be equal to the Token’s BufferSize.  Call GetPosition()the  PositionAfterRead should be equal to the sum of SetPosition and ReadLength.  Compare the content of ReadBuffer  with the data set in step2.  The ReadFailList should be empty. |
| 5.7.3.12.3 | 0x858ccc86, 0x9739, 0x437e, 0x82, 0xff, 0x29, 0x8a, 0x34, 0x7f, 0xc4, 0x45 | EFI\_FILE\_PROTOCOL.ReadEx - ReadEx() to async read data from a file returns EFI\_SUCCESS. | Call Open() to create a file.  Call Write() to write data to the  file.  Async Call ReadEx()from valid  setposition & ReadLength, the return status should be EFI\_SUCCESS and ReadLength should be equal to the Token’s BufferSize.  Call GetPosition()the  PositionAfterRead should be equal to the sum of SetPosition and ReadLength.  Compare the content of ReadBuffer  with the data set in step2.  The ReadExecuteList should be empty. |
| 5.7.3.12.4 | 0xccb9106f, 0x79ee, 0x4ec1, 0x98, 0xa5, 0x16, 0x8d, 0xe1, 0xa6, 0xb8, 0xf3 | EFI\_FILE\_PROTOCOL.ReadEx - ReadEx() to sync read data from a file returns  EFI\_SUCCESS. | Call Open()to create a file.  Call Write()to write data to the  file.  Sync Call ReadEx()from valid  setposition & ReadLength, the return status should be EFI\_SUCCESS and ReadLength should be equal to the Token’s BufferSize.  Call GetPosition()the  PositionAfterRead should be equal to the sum of SetPosition and ReadLength.  Compare the content of ReadBuffer  with the data set in step2. |
| 5.7.3.12.5 | 0xd01cdf69, 0x1b1b, 0x42fc, 0x92, 0x3f, 0x1d, 0xc1, 0x90, 0x92, 0x03, 0xc7 | EFI\_FILE\_PROTOCOL.ReadEx - ReadEx() to async read data from a directory. | Call Open() to create a directory.  Call Open() to create a file under the  directory opened in step1.  Async Call ReadEx()from different  setposition & ReadLength, if the Setposition is 0 and the ReadLength is smaller than SIZE\_OF\_EFI\_FILE\_INFO + 4, the return status should be EFI\_BUFFER\_TOO\_SMALL, else if the Setposition is 0 and the ReadLength is not less than SIZE\_OF\_EFI\_FILE\_INFO + 4, the return status should be EFI\_SUCCESS, if the Setposition is at the end of directory, the return status should be EFI\_SUCCESS. |
| 5.7.3.12.6 | 0x05241cbf, 0xf260, 0x41d7, 0xb1, 0x93, 0x3b, 0x27, 0x7f, 0x72, 0x12, 0x4c | EFI\_FILE\_PROTOCOL.ReadEx - ReadEx() to async read data from a directory. | Call Open()to create a directory.  Call Open()to create a file under the  directory opened in step1.  Async Call ReadEx()from different  setposition & ReadLength, if the Setposition is 0 and the ReadLength is smaller than SIZE\_OF\_EFI\_FILE\_INFO + 4, the return status should be EFI\_BUFFER\_TOO\_SMALL, else if the Setposition is 0 and the ReadLength is not less than SIZE\_OF\_EFI\_FILE\_INFO + 4, the return status should be EFI\_SUCCESS, if the Setposition is at the end of directory, the return status should be EFI\_SUCCESS. |
| 5.7.3.12.7 | 0xcfbb86c0, 0xc6c6, 0x40ca, 0x8e, 0xc8, 0x0d, 0x76, 0xd0, 0xef, 0x50, 0xe7 | EFI\_FILE\_PROTOCOL.ReadEx - ReadEx() to async read data from a directory. | Call Open() to create a directory.  Call Open()to create a file under the  directory opened in step1.  Async Call ReadEx()from different  setposition & ReadLength, if the Setposition is 0 and the ReadLength is smaller than SIZE\_OF\_EFI\_FILE\_INFO + 4, the return status should be EFI\_BUFFER\_TOO\_SMALL, else if the Setposition is 0 and the ReadLength is not less than SIZE\_OF\_EFI\_FILE\_INFO + 4, the return status should be EFI\_SUCCESS, if the Setposition is at the end of directory, the return status should be EFI\_SUCCESS.  The ReadExecuteList should be empty. |
| 5.7.3.12.8 | 0xe8e8665c, 0xa44f, 0x491b, 0xb7, 0xe0, 0x56, 0x09, 0xc2, 0xbc, 0x20, 0xee | EFI\_FILE\_PROTOCOL.ReadEx - ReadEx() to sync read data from a directory. | Call Open() to create a directory.  Call Open() to create a file under the  directory opened in step1.  Sync Call ReadEx()from different  setposition & ReadLength, if the Setposition is 0 and the ReadLength is small than SIZE\_OF\_EFI\_FILE\_INFO + 4, the return status should be EFI\_BUFFER\_TOO\_SMALL, else if the Setposition is 0 and the ReadLength is not less than SIZE\_OF\_EFI\_FILE\_INFO + 4, the return status should be EFI\_SUCCESS, if the Setposition is at the end of directory, the return status should be EFI\_SUCCESS. |
| 5.7.3.12.9 | 0x864c9887, 0x7205, 0x4e15,0xad, 0x9f, 0x7a, 0x94, 0xec, 0xf0, 0xc2, 0xd8 | EFI\_FILE\_PROTOCOL.ReadEx - ReadEx() async & sync read data from a file with the fileposition beyond the end of the file returns EFI\_DEVICE\_ERROR. | Async & Sync Call ReadEx()read  data from a file with the fileposition beyond the end of the file, the return status should be EFI\_DEVICE\_ERROR. |
| 5.7.3.12.10 | 0x12bc7ab7, 0x4ac5, 0x4cf3, 0xa5, 0x54, 0x6b, 0x34, 0xc9, 0x5d, 0x0c, 0xea | EFI\_FILE\_PROTOCOL.ReadEx - ReadEx() async & sync read data from a file which has been deleted returns EFI\_DEVICE\_ERROR. | Async & Sync Call ReadEx()read  data from a file which has been deleted, the return status should be EFI\_DEVICE\_ERROR. |

### WriteEX

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.13.1 | 0x077c1f80, 0xa887, 0x417d, 0xa9, 0xd6, 0xd9, 0x54, 0xca, 0x0b, 0x94, 0x7b | EFI\_FILE\_PROTOCOL.WriteEx - WriteEx() to async write data into a normal file returns EFI\_SUCCESS. | Call Open() to create a file.  Async Call WriteEx()from valid  setposition & WriteLength, the return status should be EFI\_SUCCESS and FileHandle’s position after write should be equal to the sum of Setposition and WriteLength.  Call Read(),then compare the  content of ReadBuffer with the data writed to the file in step2, they should be the same. |
| 5.7.3.13.2 | 0xf75bdc5a, 0xfd02, 0x444d, 0x9b, 0xb1, 0xda, 0x70, 0x2e, 0x2a, 0x86, 0x13 | EFI\_FILE\_PROTOCOL.WriteEx - WriteEx() to async write data into a normal file returns EFI\_SUCCESS. | Call Open() to create a file.  Async Call WriteEx()from valid  setposition & WriteLength, the return status should be EFI\_SUCCESS and FileHandle’s position after write should be equal to the sum of Setposition and WriteLength.  Call Read(),then compare the  content of ReadBuffer with the data writed to the file in step2, they should be the same.  The WriteFailList should be empty. |
| 5.7.3.13.3 | 0xc105380e, 0x4c6d, 0x4e49, 0x8d, 0xe8, 0x1a, 0x0c, 0xc0, 0x77, 0x3e, 0xdc | EFI\_FILE\_PROTOCOL.WriteEx - WriteEx() to async write data into a normal file returns EFI\_SUCCESS. | Call Open() to create a file.  Async Call WriteEx()from valid  setposition & WriteLength, the return status should be EFI\_SUCCESS and FileHandle’s position after write should be equal to the sum of Setposition and WriteLength.  Call Read(),then compare the  content of ReadBuffer with the data writed to the file in step2, they should be the same.  The WriteExecuteList should be empty. |
| 5.7.3.13.4 | 0x67e49003, 0xf68c, 0x44bd, 0xb6, 0xee, 0xa5, 0xc8, 0x01, 0x06, 0xe7, 0xc1 | EFI\_FILE\_PROTOCOL.WriteEx - WriteEx() to async write data into a normal file returns EFI\_SUCCESS. | Call Open() to create a file.  Sync Call WriteEx()from valid  setposition & WriteLength, the return status should be EFI\_SUCCESS and FileHandle’s position after write should be equal to the sum of Setposition and WriteLength.  Call Read(),then compare the  content of ReadBuffer with the data writed to the file in step2, they should be the same. |
| 5.7.3.13.5 | 0xbe6ccb33, 0x351f, 0x488c, 0x86, 0x42, 0x65, 0x47, 0xa1, 0x35, 0x79, 0x0c | EFI\_FILE\_PROTOCOL.WriteEx - WriteEx() to async write data into multi files returns EFI\_SUCCESS. | Call Open() to create three file.  Async Call WriteEx()to write data  to different file with different position and write length, the return status should be EFI\_SUCCESS.  Compare the position after write, the  writelength with the expect value.  Call Read(),then compare the  content of ReadBuffer with the data writed to the file in step2, they should be the same. |
| 5.7.3.13.6 | 0x0aaacd7f, 0xeb8b, 0x4e91, 0x9b, 0xcd, 0x30, 0x57, 0xe6, 0x10, 0x57, 0x69 | EFI\_FILE\_PROTOCOL.WriteEx - WriteEx() to async write data into multi files returns EFI\_SUCCESS. | Call Open() to create three file.  Async Call WriteEx()to write data  to different file with different position and write length, the return status should be EFI\_SUCCESS.  Compare the position after write, the  writelength with the expect value.  Call Read(),then compare the  content of ReadBuffer with the data writed to the file in step2, they should be the same.  The WriteMultiFailList should be empty. |
| 5.7.3.13.7 | 0x4c7ec69e, 0x9615, 0x4274, 0xa0, 0x99, 0xb1, 0xd3, 0x48, 0x88, 0xd6, 0x70 | EFI\_FILE\_PROTOCOL.WriteEx - WriteEx() to async write data into multi files returns EFI\_SUCCESS. | Call Open() to create three file.  Async Call WriteEx()to write data  to different file with different position and write length, the return status should be EFI\_SUCCESS.  Compare the position after write, the  writelength with the expect value.  Call Read(),then compare the  content of ReadBuffer with the data writed to the file in step2, they should be the same.  The WriteMultiExecuteList should be empty. |
| 5.7.3.13.8 | 0x03186ac5, 0xb4b2, 0x4d2d, 0xa8, 0x67, 0xb9, 0x10, 0xdd, 0x1f, 0x64, 0xad | EFI\_FILE\_PROTOCOL.WriteEx - WriteEx() to async write data into multi files returns EFI\_SUCCESS. | Call Open() to create three file.  Sync Call WriteEx()to write data  to different file with different position and write length, the return status should be EFI\_SUCCESS.  Compare the position after write, the  writelength with the expect value.  Call Read(),then compare the  content of ReadBuffer with the data writed to the file in step2, they should be the same. |
| 5.7.3.13.9 | 0xc51c0c6d, 0xdfc6, 0x4ea7,0xb4, 0x36, 0x83, 0xae, 0x3a, 0x3f, 0x49, 0xd2 | EFI\_FILE\_PROTOCOL.WriteEx - WriteEx() to async & sync write data to a directory returns EFI\_UNSUPPORTED. | Call WriteEx() to async & sync write data to a directory, the return status should be EFI\_DEVICE\_ERROR. |
| 5.7.3.13.10 | 0xc9af9973, 0x76af, 0x4701,0x88, 0xc0, 0xff, 0x61, 0x0e, 0x37, 0x74, 0x0a | EFI\_FILE\_PROTOCOL.WriteEx - WriteEx() to async & sync write data to a file which was opened read-only returns EFI\_ACCESS\_DENIED. | Async & sync Call WriteEx() to write data to a file which was opened read-only, the return status should be EFI\_ACCESS\_DENIED. |
| 5.7.3.13.11 | 0xa056bcff, 0xdb0b, 0x4733,0x88, 0x9a, 0xb1, 0xca, 0x52, 0xac, 0x58, 0xe9 | EFI\_FILE\_PROTOCOL.WriteEx - WriteEx() to async & sync write data to a file which has been deleted returns EFI\_DEVICE\_ERROR. | Async & sync Call WriteEx() to write data to a file which has been deleted, the return status should be EFI\_DEVICE\_ERROR. |

### FlushEx

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.14.1 | 0x31473e47, 0xa40d, 0x43a0, 0xb7, 0xb8, 0x91, 0xd3, 0x29, 0x41, 0x75, 0x9d | EFI\_FILE\_PROTOCOL.FlushEx - FlushEx () to async flush data into a normal file returns EFI\_SUCCESS. | Call Open() to create a file.  Call Write() to write data to the file.  Async Call FlushEx(), the return  status should be EFI\_SUCCESS. |
| 5.7.3.14.2 | 0x55702a2c, 0x0eef,  0x4ded, 0xa6, 0xd9, 0x2f, 0xd7, 0x9a, 0xbb, 0x88, 0x5f | EFI\_FILE\_PROTOCOL.FlushEx - FlushEx () to async flush data into a normal file returns EFI\_SUCCESS. | Call Open() to create a file.  Call Write() to write data to the file.  Async Call FlushEx(), the return  status should be EFI\_SUCCESS.  The flushFileFailList should be empty. |
| 5.7.3.14.3 | 0x258a6597, 0xd2ef, 0x4711, 0xa9, 0x89, 0xaa, 0xf0, 0xf9, 0x6f, 0x01, 0x0c | EFI\_FILE\_PROTOCOL.FlushEx - FlushEx () to async flush data into a normal file returns EFI\_SUCCESS. | Call Open() to create a file.  Call Write() to write data to the file.  Async Call FlushEx(), the return  status should be EFI\_SUCCESS.  The FlushFileExecuteList should be empty. |
| 5.7.3.14.4 | 0xafd40ec9, 0x5027, 0x42a8, 0xb0, 0x2c, 0x0c, 0xb5, 0x80, 0x86, 0xd7, 0x9c | EFI\_FILE\_PROTOCOL.FlushEx - FlushEx () to sync flush data into a normal file returns EFI\_SUCCESS. | Call Open() to create a file.  Call Write() to write data to the file.  Sync Call FlushEx(), the return  status should be EFI\_SUCCESS. |
| 5.7.3.14.5 | 0x6aa8b399, 0x1b2f, 0x48d7, 0xa5, 0x34, 0x56, 0xc9, 0x68, 0xd6, 0xae, 0x11 | EFI\_FILE\_PROTOCOL.FlushEx - FlushEx () to async flush data into a normal directory returns EFI\_SUCCESS. | Call Open() to create a directory.  Call Open() to Create files under the directory.  Async Call FlushEx(), the return status should be EFI\_SUCCESS. |
| 5.7.3.14.6 | 0xac3897ad, 0xd9c1, 0x4442, 0x84, 0x4b, 0x5c, 0xa1, 0x5c, 0x32, 0x80, 0x0b | EFI\_FILE\_PROTOCOL.FlushEx - FlushEx () to async flush data into a normal directory returns EFI\_SUCCESS. | Call Open() to create a directory.  Call Open() to Create files under the directory.  Async Call FlushEx(), the return  status should be EFI\_SUCCESS.  The FlushDirFailList should be empty. |
| 5.7.3.14.7 | 0x3b9ed07d, 0xa0ea, 0x4719, 0xa2, 0xc9, 0xad, 0x54, 0x57, 0xc1, 0x5a, 0x73 | EFI\_FILE\_PROTOCOL.FlushEx - FlushEx () to async flush data into a normal directory returns EFI\_SUCCESS. | Call Open() to create a directory.  Call Open() to Create files under the  directory.  Async Call FlushEx(), the return  status should be EFI\_SUCCESS.  The FlushDirExecuteList should be empty. |
| 5.7.3.14.8 | 0x93ebe8a5, 0xf66b, 0x4532, 0x95, 0x77, 0x51, 0xe9, 0xdc, 0xda, 0xb6, 0x81 | EFI\_FILE\_PROTOCOL.FlushEx - FlushEx () to sync flush data into a normal directory returns EFI\_SUCCESS. | Call Open() to create a directory.  Call Open() to Create files under the  directory.  Sync Call FlushEx(), the return  status should be EFI\_SUCCESS. |
| 5.7.3.14.9 | 0xce7774fa, 0xd04c, 0x45a6, 0xb7, 0x0b, 0xcd, 0x91, 0xa2, 0x76, 0xf9, 0x15 | EFI\_FILE\_PROTOCOL.FlushEx - FlushEx () to async & sync flush data to a file whose open mode was read-only returns EFI\_ACCESS\_DENIED. | Call Open() to create a directory.  Call Open() to open the file in the  mode of Read-Only.  Async & Sync Call FlushEx(), the  return status should be EFI\_ACCESS\_DENIED. |

### Read-Only File System check points

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.3.15.1 | 0xad3516c1, 0xbf24, 0x4923, 0xb8, 0x84, 0x53, 0x8b, 0x04, 0x2f, 0xb8, 0x25 | EFI\_FILE\_PROTOCOL.GetInfo - GetInfo () get the consistent ReadOnly attribute from EFI\_FILE\_INFO and EFI\_FILE\_SYSTEM\_INFO. | Call GetInfo() to check the ReadOnly attribute from EFI\_FILE\_INFO and EFI\_FILE\_SYSTEM\_INFO. The value should be consistent. |
| 5.7.3.15.2 | 0x5b704b82, 0xe081, 0x4c4a, 0x9d, 0x65, 0x71, 0x00, 0x79, 0xd1, 0x1f, 0x64 | EFI\_FILE\_PROTOCOL.SetPosition - SetPosition  () return EFI\_UNSUPPORTED when the position is not 0 and file handle is the root directory on a volume. | Call SetPosition() when the position is not 0 and file handle is the root directory on a volume, the return status should be EFI\_UNSUPPORTED. |
| 5.7.3.15.3 | 0xd9cbe15a, 0x956a, 0x4e54, 0xa3, 0x50, 0xdf, 0x53, 0xdc, 0x7d, 0xe2, 0x5b | EFI\_FILE\_PROTOCOL.SetPosition - SetPosition () return EFI\_SUCCESS when the position is 0 and file handle is the root directory on a volume. | Call SetPosition ()  return EFI\_SUCCESS when the position is 0 and file handle is the root directory on a volume. |
| 5.7.3.15.4 | 0xa8aadad0, 0x8545, 0x4098, 0x8a, 0x34, 0x2a, 0x03, 0xc2, 0x2b, 0xc0, 0xf6 | EFI\_FILE\_PROTOCOL.GetPosition - GetPosition () return EFI\_UNSUPPORTED when the file handle is the root directory on a volume. | Call GetPosition ()  return EFI\_UNSUPPORTED when the file handle is the root directory on a volume. |
| 5.7.3.15.5 | 0xb20660fc, 0xb957, 0x49d7, 0x8d, 0x93, 0x5c, 0x3f, 0x73, 0x6e, 0xd5, 0xf5 | EFI\_FILE\_PROTOCOL.SetInfo - SetInfo () return EFI\_WRITE\_PROTECTED when the InformationType is  EFI\_FILE\_SYSTEM\_VOLUME\_LABEL\_ID or EFI\_FILE\_PROTOCOL\_SYSTEM\_INFO\_ID  and the media is read-only. | Call SetInfo()  return EFI\_WRITE\_PROTECTED when the InformationType is  EFI\_FILE\_SYSTEM\_VOLUME\_LABEL\_ID or EFI\_FILE\_PROTOCOL\_SYSTEM\_INFO\_ID  and the media is read-only. |
| 5.7.3.15.6 | 0x04d6b761, 0xdeac, 0x4801, 0xb7, 0x39, 0xdb, 0x81, 0x8f, 0x46, 0xcf, 0x11 | EFI\_FILE\_PROTOCOL.Write - Write () return EFI\_UNSUPPORTED when the file handle is one directory. | Call Write()  return EFI\_UNSUPPORTED when the file handle is one directory. |
| 5.7.3.15.7 | 0xbe5cddad, 0x2d54, 0x463e, 0xaf, 0xde, 0x68, 0x1c, 0xb9, 0x08, 0xa8, 0xa0 | EFI\_FILE\_PROTOCOL.Read - Read ()  return EFI\_BUFFER\_TOO\_SMALL when the file handle is one directory and buffer  is not large enough to hold the directory entry. | Call Read() return EFI\_BUFFER\_TOO\_SMALL when the file handle is one directory and buffer is not large enough to hold the directory entry. |
| 5.7.3.15.8 | 0x06950775, 0xa32a, 0x421e, 0x8f, 0xce, 0xd8, 0xb4, 0xc1, 0x43, 0x17, 0xd1 | EFI\_FILE\_PROTOCOL.Open - Open ()  return EFI\_WRITE\_PROTECTED when try to open the file with EFI\_FILE\_MODE\_READ|EFI\_FILE\_MODE\_WRITE or EFI\_FILE\_MODE\_READ|EFI\_FILE\_MODE\_WRITE|EFI\_FILE\_MODE\_CREATE attribute while the file is on the read-only media. | Call Open() return EFI\_WRITE\_PROTECTED when try to open the file with EFI\_FILE\_MODE\_READ|EFI\_FILE\_MODE\_WRITE or EFI\_FILE\_MODE\_READ|EFI\_FILE\_MODE\_WRITE|EFI\_FILE\_MODE\_CREATE attribute while the file is on the read-only media.. |
| 5.7.3.15.9 | 0xd529dfd8, 0x23cb, 0x4548, 0xa2, 0x81, 0x6f, 0x59, 0x1f, 0x9c, 0x54, 0x8d | EFI\_FILE\_PROTOCOL.Open - Open () return EFI\_NOT\_FOUND when try to open one no-existed file. | Call Open()  return EFI\_NOT\_FOUND when try to open one no-existed file. |
| 5.7.3.15.10 | 0xb0091f09, 0x6121, 0x40e8, 0x93, 0x1d, 0xea, 0x6b, 0xa4, 0x6b, 0xbb, 0x09 | EFI\_FILE\_PROTOCOL.Open - Open ()  return EFI\_SUCCESS when try to open one existed file with EFI\_FILE\_MODE\_READ attribute. | Call Open() return EFI\_SUCCESS when try to open one existed file with EFI\_FILE\_MODE\_READ attribute. |
| 5.7.3.15.11 | 0xa42a8e9c, 0x4a31, 0x4b0a, 0xab, 0x2e, 0x7f, 0xd4, 0x2d, 0x42, 0x45, 0xf1 | EFI\_FILE\_PROTOCOL.GetInfo - GetInfo () return EFI\_UNSUPPORTED when the InformationType is not defined in the UEFI Specification. | Call GetInfo()  return EFI\_UNSUPPORTED when the InformationType is not defined in the UEFI Specification. |
| 5.7.3.15.12 | 0x07dc8d79, 0x8349, 0x4e9e, 0x9b, 0xa4, 0x72, 0x68, 0x92, 0x0d, 0x2e, 0x35 | EFI\_FILE\_PROTOCOL.GetInfo - GetInfo ()  return EFI\_BUFFER\_TOO\_SMALL when the Buffer is not large enough to hold the EFI\_FILE\_INFO. | Call GetInfo() return EFI\_BUFFER\_TOO\_SMALL when the Buffer is not large enough to hold the EFI\_FILE\_INFO. |
| 5.7.3.15.13 | 0x54afc2f4, 0x26bd, 0x4161, 0x90, 0x5e, 0xd9, 0x24, 0xd1, 0x34, 0x24, 0x27 | EFI\_FILE\_PROTOCOL.GetInfo - GetInfo () return EFI\_SUCCESS with the correct parameters. | Call GetInfo()  return EFI\_SUCCESS with the correct parameters. |
| 5.7.3.15.14 | 0xabaea718, 0xe1f9, 0x4edc, 0x98, 0xb2, 0x47, 0x18, 0xe4, 0xf7, 0x6b, 0x70 | EFI\_FILE\_PROTOCOL.SetInfo - SetInfo () return  EFI\_WRITE\_PROTECTED to retrieve the EFI\_FILE\_INFO or EFI\_UNSUPPORTED when InformationType is not defined in the UEFI Specification. | Call SetInfo()  return EFI\_WRITE\_PROTECTED to retrieve the EFI\_FILE\_INFO or EFI\_UNSUPPORTED when InformationType is not defined in the UEFI Specification. |
| 5.7.3.15.15 | 0x68a6c62b, 0xc1e0, 0x44d0, 0xba, 0xdb, 0x08, 0x85, 0x63, 0x37, 0x3f, 0xd7 | EFI\_FILE\_PROTOCOL. GetPosition - GetPosition ()  return EFI\_SUCCESS and one reasonable Position. | Call GetPosition() return EFI\_SUCCESS and one reasonable Position. |
| 5.7.3.15.16 | 0x2f83c19f, 0xc757, 0x4975, 0xa5, 0xea, 0x6a, 0x4e, 0xab, 0xa7, 0xce, 0x48 | EFI\_FILE\_PROTOCOL. Write - Write ()  return EFI\_WRITE\_PROTECTED when the media is read-only. | Call Write() return EFI\_WRITE\_PROTECTED when the media is read-only. |
| 5.7.3.15.17 | 0x3c0a4e4a, 0x43f4, 0x4b24, 0xb7, 0x64, 0xd8, 0x3c, 0x18, 0x63, 0xab, 0x81 | EFI\_FILE\_PROTOCOL. Flush - Flush ()  return EFI\_WRITE\_PROTECTED when the media is read-only. | Call Flush() return EFI\_WRITE\_PROTECTED when the media is read-only. |
| 5.7.3.15.18 | 0xece0ade2, 0x027e, 0x4c21, 0x91, 0x50, 0x33, 0x3c, 0x3e, 0x47, 0xea, 0x0b | EFI\_FILE\_PROTOCOL. Read - Read ()  return EFI\_SUCCESS and the output should be consistent in multi read operations. | 1. Call SetPosition() to set the position at 0.  2. Call Read() to read all content and save to FileBuf and get the file size.  3. Call GetPosition() to get the current position after the read operation.  4. Three returned status should be EFI\_SUCCESS, Position should equal with file size, and the FileSize of EFI\_FILE\_INFO should be equal with file size.  5. Read the file from variable positions, the output should be consistent with file content read from step 2. |
| 5.7.3.15.19 | 0x5ee32a7f, 0x0a63, 0x4803, 0x8a, 0xe8, 0x01, 0x9c, 0x07, 0x2a, 0xed, 0xb1 | EFI\_FILE\_PROTOCOL. Delete - Delete ()  return EFI\_WARN\_DELETE\_FAILURE. | Call Delete () return EFI\_WARN\_DELETE\_FAILURE |
| 5.7.3.15.20 | 0x3f8b11ec, 0x6b9e, 0x440c, 0x92, 0x0b, 0xb5, 0x63, 0xf3, 0xfd, 0x2b, 0xa7 | EFI\_FILE\_PROTOCOL. Close - Close ()  one existed opened file return EFI\_SUCCESS. | 1. Call Open() to open one existed file.  2. Call Close () return EFI\_SUCCESS. |

## EFI\_DISK\_IO\_PROTOCOL Test

Reference Document:

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

### ReadDisk()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.4.1.1 | 0x26912470, 0xf463, 0x4f8e, 0x8a, 0x33, 0xf3, 0x8f, 0x9c, 0xc8, 0x0d, 0x04 | EFI\_DISK\_IO\_PROTOCOL.ReadDisk – ReadDisk() returns EFI\_SUCCESS with valid parameter. | Locate Block I/O interface that is associated with specified Disk I/O interface.  For device with a **EFI\_BLOCK\_IO\_MEDIA.MediaPresent** value of TRUE, and for different valid OffSet parameter and BufferSize parameter:  1. Call ReadDisk() with the OffSet and BufferSize  Expected Behavior:  The return code of ReadDisk() should be EFI\_SUCCESS. |
| 5.7.4.1.2 | 0x9603aba0, 0xb4dd, 0x4ab6, 0x93, 0xcb, 0x52, 0x3a, 0x5b, 0x6f, 0xa5, 0x58 | EFI\_DISK\_IO\_PROTOCOL.ReadDisk – ReadDisk() returns EFI\_MEDIA\_CHANGED with MediaId is not the ID for the current media in the device. | Locate Block I/O interface that is associated with specified Disk I/O interface.  For device with a **EFI\_BLOCK\_IO\_MEDIA.MediaPresent** value of TRUE:  1. Call ReadDisk() with valid parameters and a MediaId value of actual MediaId + 5.  2. Call ReadDisk() with valid parameters and a MediaId value of actual MediaId + 1.  3. Call ReadDisk() with valid parameters and a MediaId value of actual MediaId – 1.  4. Call ReadDisk() with valid parameters and a MediaId value of actual MediaId – 5.  5. Call ReadDisk() with valid parameters and a MediaId value of 0.  Expected Behavior:  For that new MediaId not equal to old MediaId, the return code must be EFI\_MEDIA\_CHANGED. |
| 5.7.4.1.3 | 0x6a6d39d0, 0x311d, 0x410f, 0x96, 0x2e, 0x96, 0xef, 0xfb, 0x39, 0x99, 0x44 | EFI\_DISK\_IO\_PROTOCOL.ReadDisk – ReadDisk() returns EFI\_INVALID\_PARAMETER with invalid device addresses. | Locate Block I/O interface that is associated with specified Disk I/O interface.  For device with a **EFI\_BLOCK\_IO\_MEDIA.MediaPresent** value of TRUE:  1. Call ReadDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize** + 1.  2. Call ReadDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize** + 10.  3. Call ReadDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize –** BufferSize + 1.  4. Call ReadDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize –** BufferSize + 2.  5. Call ReadDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize –** BufferSize + 3.  6. Call ReadDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize –** BufferSize + 4.  Expected Behavior:  The return code must be EFI\_INVALID\_PARAMETER. |
| 5.7.4.1.4 | 0xb0d7a6e7, 0x49f1, 0x40d5, 0xa9, 0x29, 0x1a, 0xd5, 0xd4, 0x27, 0x70, 0xbf | EFI\_DISK\_IO\_PROTOCOL.ReadDisk – ReadDisk() returns EFI\_NO\_MEDIA with no media present in the device. | Locate Block I/O interface that is associated with specified Disk I/O interface.  For device with a **EFI\_BLOCK\_IO\_MEDIA.MediaPresent** value of FALSE:  1. Call ReadDisk() with valid parameter.  Expected Behavior:  The return code must be EFI\_NO\_MEDIA. |

### WriteDisk()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.4.2.1 | 0xc3d66c15, 0xb8ad, 0x45ad, 0xbe, 0xb7, 0x38, 0xfe, 0xc9, 0x52, 0x81, 0x5e | EFI\_DISK\_IO\_PROTOCOL.WriteDisk – WriteDisk() returns EFI\_SUCCESS to write proper data to non-readonly disk with valid parameter. | Locate Block I/O interface that is associated with specified Disk I/O interface.  For non-readonly disk with a **EFI\_BLOCK\_IO\_MEDIA.MediaPresent** value of TRUE and for different valid OffSet parameter and BufferSize parameter:  1. Call ReadDisk() with the OffSet and BufferSize.  2. Call WriteDisk() with same OffSet and BufferSize to write the specified buffer (different to buffer read from the last call of ReadDisk()) to the disk.  3. Call ReadDisk() with same OffSet and BufferSize.  4. Call WriteDisk() with same OffSet and BufferSize to write the buffer data read from the first ReadDisk() call.  5. Call ReadDisk() with same OffSet and BufferSize again.  Expected Behavior:  For each action, the return code should be EFI\_SUCCESS.  For each OffSet and BufferSize, the buffer data read by first and last calling ReadDisk() should be the same.  For each OffSet and BufferSize, the buffer data return in the second call of ReadDisk() should be the same with the originally buffer data written to device in the first call of WriteDisk(). |
| 5.7.4.2.2 | 0x36d696b1, 0x1902, 0x46b7, 0x9a, 0x62, 0x85, 0x25, 0x1d, 0xf5, 0xec, 0x25 | EFI\_DISK\_IO\_PROTOCOL.WriteDisk – WriteDisk() returns EFI\_MEDIA\_CHANGED with MediaId is not the ID for the current media in the device. | Locate Block I/O interface that is associated with specified Disk I/O interface.  For non-readonly device with a **EFI\_BLOCK\_IO\_MEDIA.MediaPresent** value of TRUE:  1. Call WriteDisk() with valid parameters and a MediaId value of actual **EFI\_BLOCK\_IO\_MEDIA.MediaId** + 5.  2. Call WriteDisk() with valid parameters and a MediaId value of actual EFI\_BLOCK\_IO\_MEDIA**.MediaId** + 1.  3. Call WriteDisk() with valid parameters and a MediaId value of actual EFI\_BLOCK\_IO\_MEDIA**.MediaId** – 1.  4. Call WriteDisk() with valid parameters and a MediaId value of actual EFI\_BLOCK\_IO\_MEDIA**.MediaId** – 5.  5. Call WriteDisk() with valid parameters and a MediaId value of 0.  Expected Behavior:  For that new MediaId not equal to old MediaId, the return code must be EFI\_MEDIA\_CHANGED. |
| 5.7.4.2.3 | 0xc6eea54a, 0xde3a, 0x425a, 0xa6, 0x42, 0x79, 0xf4, 0xb7, 0x9a, 0x62, 0x36 | EFI\_DISK\_IO\_PROTOCOL.WriteDisk – WriteDisk() returns EFI\_INVALID\_PARAMETER with invalid device addresses. | Locate Block I/O interface that is associated with specified Disk I/O interface.  For non-readonly device with a **EFI\_BLOCK\_IO\_MEDIA.MediaPresent** value of TRUE:  1. Call WriteDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize** + 1.  2. Call WriteDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize** + 10.  3. Call WriteDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize –** BufferSize + 1.  4. Call WriteDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize –** BufferSize + 2.  5. Call WriteDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize –** BufferSize + 3.  6. Call WriteDisk() with an OffSet value of **EFI\_BLOCK\_IO\_MEDIA.LastBlock \* EFI\_BLOCK\_IO\_MEDIA.BlockSize –** BufferSize + 4.  Expected Behavior:  The return code must be EFI\_INVALID\_PARAMETER. |
| 5.7.4.2.4 | 0x83a99320, 0x0831, 0x42d6, 0x8b, 0xec, 0x8d, 0xfd, 0x3d, 0xe4, 0x63, 0x78 | EFI\_DISK\_IO\_PROTOCOL.WriteDisk – WriteDisk() returns EFI\_WRITE\_PROTECTED with a write-protected device. | Locate Block I/O interface that is associated with specified Disk I/O interface.  For read-only device with a **EFI\_BLOCK\_IO\_MEDIA.MediaPresent** value of TRUE:  1. Call WriteDisk() with valid parameter to write data to device.  Expected Behavior:  The return code must be EFI\_WRITE\_PROTECTED. |
| 5.7.4.2.5 | 0x0299b063, 0x21a8, 0x4811, 0x80, 0xe2, 0x8c, 0x4f, 0xfd, 0x3e, 0xd0, 0xa4 | EFI\_DISK\_IO\_PROTOCOL.WriteDisk – WriteDisk() returns EFI\_NO\_MEDIA with no media in the device. | Locate Block I/O interface that is associated with specified Disk I/O interface.  For device with a **EFI\_BLOCK\_IO\_MEDIA.MediaPresent** value of FALSE:  1. Call WriteDisk() with valid parameter to write data to device.  Expected Behavior:  The return code must be EFI\_NO\_MEDIA. |

## EFI\_BLOCK\_IO\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_BLOCK\_IO\_ PROTOCOL Section.

### Reset()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.5.1.1 | 0x61ee3a34, 0x62a2, 0x4214, 0xb0, 0x76, 0x50, 0x73, 0xb1, 0x77, 0x15, 0x6c | EFI\_BLOCK\_IO\_PROTOCOL.Reset - Reset() returns EFI\_SUCCESS with an ExtendedVerification value of TRUE. | 1. Call Reset() with an ExtendedVerification value of TRUE  Expected Behavior:  The return code should be EFI\_SUCCESS.  The private data for the device, which is stored in Media data structure, should be kept unchanged, |
| 5.7.5.1.2 | 0x98530f3d, 0x8bd8, 0x44a1, 0x9d, 0x06, 0x08, 0x03, 0x9f, 0xdf, 0xec, 0x63 | EFI\_BLOCK\_IO\_PROTOCOL.Reset - Reset() returns EFI\_SUCCESS with an ExtendedVerification value of FALSE. | 1. Call Reset() with an ExtendedVerification value of FALSE  Expected Behavior:  The return code should be EFI\_SUCCESS.  The private data for the device, which is stored in Media data structure, should be kept unchanged. |

### ReadBlocks()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.5.2.1 | 0x9efe26c2, 0xc565, 0x478a, 0xa0, 0xb4, 0x05, 0xa8, 0xfd, 0x2e, 0x7e, 0x3e | EFI\_BLOCK\_IO\_PROTOCOL.ReadBlocks – ReadBlocks() returns EFI\_SUCCESS with valid parameter. | (Can only be invoked when media is present.)  1. Call ReadBlocks() with different LBA and BufferSize. The return code should be EFI\_SUCCESS. |
| 5.7.5.2.2 | 0x6dec8f5c, 0xf6ec, 0x47b4, 0xbb, 0x0c, 0xaa, 0x4a, 0x69, 0x39, 0xe2, 0xf0 | EFI\_BLOCK\_IO\_PROTOCOL.ReadBlocks – ReadBlocks() returns EFI\_MEDIA\_CHANGED with MediaId is not the ID for the current media in the device. | For device with a **MediaPresent** value of TRUE:  1. Call ReadBlocks() with valid parameters and a MediaId value of actual MediaId + 5  2. Call ReadBlocks() with valid parameters and a MediaId value of actual MediaId + 1  3. Call ReadBlocks() with valid parameters and a MediaId value of actual MediaId – 1  4. Call ReadBlocks() with valid parameters and a MediaId value of actual MediaId – 5  5. Call ReadBlocks() with valid parameters and a MediaId value of 0  Expected Behavior:  The return code must be EFI\_MEDIA\_CHANGED. |
| 5.7.5.2.3 | 0x05927e73, 0x8b41, 0x4cc7, 0x8e, 0xf2, 0x7c, 0x7a, 0xfb, 0x78, 0xf5, 0x3e | EFI\_BLOCK\_IO\_PROTOCOL.ReadBlocks – ReadBlocks() returns EFI\_BAD\_BUFFER\_SIZE with invalid BufferSize parameter. (Can only be invoked when media is present.) | For device with a **MediaPresent** value of TRUE and a **BlockSize** value other than 1:  1. Call ReadBlocks() with valid parameters and a BufferSize value of **BlockSize** + 1  2. Call ReadBlocks() with valid parameters and a BufferSize value of 2\*BlockSize – 1  3. Call ReadBlocks() with valid parameters and a BufferSize value of 2\***BlockSize** + 1  4. Call ReadBlocks() with valid parameters and a BufferSize value of 3\***BlockSize** – 1  Expected Behavior:  All return codes must be EFI\_BAD\_BUFFER\_SIZE. |
| 5.7.5.2.4 | 0x09de1965, 0x3719, 0x463b, 0xa8, 0xd1, 0xd2, 0x78, 0xd7, 0xd6, 0x58, 0x2c | EFI\_BLOCK\_IO\_PROTOCOL.ReadBlocks – ReadBlocks() returns EFI\_INVALID\_PARAMETER with invalid *LBA* parameter. | For device with a **MediaPresent** value of TRUE:  1. Call ReadBlocks() with valid parameters and an *LBA* value of **LastBlock** + 1  2. Call ReadBlocks() with valid parameters and an *LBA* value of **LastBlock** + 100  3. Call ReadBlocks() with valid parameters and an *LBA* value of **LastBlock** – BufferSize**/BlockSize** + 1  4. Call ReadBlocks() with valid parameters and an *LBA* value of **LastBlock** – BufferSize**/BlockSize** + 2  5. Call ReadBlocks() with valid parameters and an *LBA* value of **LastBlock** – BufferSize**/BlockSize** + 3  6. Call ReadBlocks() with valid parameters and an *LBA* value of **LastBlock** – BufferSize**/BlockSize** + 100  Expected Behavior:  The return code must be EFI\_INVALID\_PARAMETER. |
| 5.7.5.2.5 | 0x91cfde2c, 0x619e, 0x4c88, 0x80, 0x0d, 0x99, 0xce, 0x53, 0xad, 0x3b, 0x25 | EFI\_BLOCK\_IO\_PROTOCOL.ReadBlocks – ReadBlocks() returns EFI\_NO\_MEDIA with no media present in the device. | For device with a **MediaPresent** value of FALSE:  1. Call ReadBlocks() with valid parameter.  Expected Behavior:  The return code must be EFI\_NO\_MEDIA. |
| 5.7.5.2.6 | 0x8cf48053, 0x8e2e, 0x40c9, 0x90, 0xfa, 0x65, 0x33, 0x0b, 0xbf, 0x33, 0x69 | EFI\_BLOCK\_IO\_PROTOCOL.ReadBlocks – ReadBlocks() returns EFI\_INVALID\_PARAMETER with Buffer is not on proper lower alignment. (Can only be invoked when media present and **IoAlign** is larger than 1.) | For device with a **MediaPresent** value of TRUE and **IoAlign** more than 1:  1. Call ReadBlocks() with valid parameter and a Buffer value of (Buffer/**IoAlign**) \* **IoAlign** + Remainder (Remainder goes from 1 to Min(**IoAlign**-1, 5)).  Expected Behavior:  The return code must be EFI\_INVALID\_PARAMETER. |
| 5.7.5.2.7 | 0x9284cf69, 0x7570, 0x4da4, 0xa7, 0xa2, 0x40, 0x5d, 0x27, 0x9d, 0x0c, 0xa7 | EFI\_BLOCK\_IO\_PROTOCOL.ReadBlocks – ReadBlocks() returns EFI\_INVALID\_PARAMETER with Buffer is not on proper alignment. (Can only be invoked when media present and **IoAlign** is larger than 1.) | For device with a **MediaPresent** value of TRUE and **IoAlign** more than 1:  1. Call ReadBlocks() with valid parameter and a Buffer value of (Buffer/**IoAlign**) \* **IoAlign** + Remainder (Remainder goes from **IoAlign**-1 down to Max(**IoAlign**-6, 1)).  Expected Behavior:  The return code must be EFI\_INVALID\_PARAMETER. |

### 

### WriteBlocks()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.5.3.1 | 0x7bbdf28f, 0xb2ea, 0x42c0, 0xa8, 0xfe, 0x6a, 0xdc, 0x00, 0x38, 0x35, 0x77 | EFI\_BLOCK\_IO\_PROTOCOL.Writeblocks – WriteBlocks() returns EFI\_SUCCESS with valid parameters. | (Can only be invoked when media is present and not read-only.)  1. Call ReadBlocks() to get the original data in the media.  2. Call WriteBlocks() with the new data. The return code should be EFI\_SUCCESS. |
| 5.7.5.3.2 | 0x1fb19cbd, 0x7219, 0x4853, 0xa2, 0xaa, 0xeb, 0xe5, 0x17, 0xaa, 0xad, 0xe6 | EFI\_BLOCK\_IO\_PROTOCOL.Writeblocks – ReadBlocks() gets the same data as what are written before. | (Can only be invoked when media is present and not read-only.)  1. Call ReadBlocks() to get the original data in the media.  2. Call WriteBlocks() with the new data.  3. Call ReadBlocks() to get the data in the media.  The data should be the same as the new data written before. |
| 5.7.5.3.3 | 0x48340af1, 0x8425, 0x4847, 0xaa, 0x69, 0x56, 0x52, 0xd6, 0x61, 0x6e, 0x08 | EFI\_BLOCK\_IO\_PROTOCOL.Writeblocks – WriteBlocks() must return EFI\_SUCCESS after being called twice with valid parameters. | (Can only be invoked when media is present and not read-only.)  1. Call ReadBlocks() to get the original data in the media.  2. Call WriteBlocks() with the new data.  3. Call ReadBlocks() to get the data in the media  4. Call WriteBlocks() with the original data. The return code should be EFI\_SUCCESS. |
| 5.7.5.3.4 | 0xa4383f2b, 0xf875, 0x4f57, 0x95, 0xfe, 0xce, 0x65, 0x5a, 0x4d, 0xc6, 0xb0 | EFI\_BLOCK\_IO\_PROTOCOL.Writeblocks – WriteBlocks() returns EFI\_MEDIA\_CHANGED with invalid MediaId. | For non-readonly device with a **MediaPresent** value of TRUE:  1. Call WriteBlocks() with valid parameters and a MediaId value of actual MediaId + 5  2. Call WriteBlocks() with valid parameters and a MediaId value of actual MediaId + 1  3. Call WriteBlocks() with valid parameters and a MediaId value of actual MediaId – 1  4. Call WriteBlocks() with valid parameters and a MediaId value of actual MediaId – 5  5. Call WriteBlocks() with valid parameters and a MediaId value of 0  Expected Behavior:  The return code must be EFI\_MEDIA\_CHANGED. |
| 5.7.5.3.5 | 0xbf9eabdd, 0x1745, 0x4418, 0xaf, 0xf8, 0x12, 0x5e, 0x02, 0x18, 0x94, 0xaa | EFI\_BLOCK\_IO\_PROTOCOL.Writeblocks – ReadBlocks() get the original data written before. | (Can only be invoked when media is present and not read-only.)  1. Call ReadBlocks() to get the original data in the media.  2. Call WriteBlocks() with the new data.  3. Call ReadBlocks() to get the data in the media  4. Call WriteBlocks() with the original data.  5. Call ReadBlocks() to get the data in the media. The data should be the same as the original data written before. |
| 5.7.5.3.6 | 0xa77c46e0, 0x6df6, 0x4d63, 0xaf, 0x8d, 0xae, 0xb7, 0xae, 0x7d, 0x2b, 0x12 | EFI\_BLOCK\_IO\_PROTOCOL.Writeblocks – WriteBlocks() returns EFI\_BAD\_BUFFER\_SIZE with invalid BufferSize. | For non-readonly device with a **MediaPresent** value of TRUE:  1. Call WriteBlocks() with valid parameters and a BufferSize value of **BlockSize** + 1.  2. Call WriteBlocks() with valid parameters and a BufferSize value of 2\***BlockSize** – 1.  3. Call WriteBlocks() with valid parameters and a BufferSize value of 2\***BlockSize** + 1  4. Call WriteBlocks() with valid parameters and a BufferSize value of 3\***BlockSize** – 1.  Expected Behavior:  The return code must be EFI\_BAD\_BUFFER\_SIZE. |
| 5.7.5.3.7 | 0x98e637f8, 0x9a1c, 0x42f9, 0xa6, 0xe2, 0x2e, 0xe8, 0x5f, 0x70, 0x2b, 0x98 | EFI\_BLOCK\_IO\_PROTOCOL.Writeblocks – WriteBlocks() return EFI\_INVALID\_PARAMETER with invalid LBA parameter. | For non-readonly device with a **MediaPresent** value of TRUE:  1. Call WriteBlocks() with valid parameters and an LBA value of **LastBlock** + 1.  2. Call ReadBlocks() with valid parameters and an LBA value of **LastBlock** + 100.  3. Call ReadBlocks() with valid parameters and an LBA value of **LastBlock** – BufferSize**/BlockSize** + 1  4. Call ReadBlocks() with valid parameters and an LBA value of **LastBlock** – BufferSize**/BlockSize** + 2.  5. Call ReadBlocks() with valid parameters and an LBA value of **LastBlock** – BufferSize**/BlockSize** + 3.  6. Call ReadBlocks() with valid parameters and an LBA value of **LastBlock** – BufferSize**/BlockSize** + 100.  Expected Behavior:  The return code must be EFI\_INVALID\_PARAMETER. |
| 5.7.5.3.8 | 0xedb9cf57, 0x1900, 0x45f2, 0x9a, 0x5a, 0xf1, 0x3b, 0x31, 0xdf, 0x36, 0x6a | EFI\_BLOCK\_IO\_PROTOCOL.Writeblocks – WriteBlocks() returns EFI\_WRITE\_PROTECTED with write-protected device. | For read-only device with a **MediaPresent** value of TRUE:  1. Call WriteBlocks() with valid parameter to write data to device.  Expected Behavior:  The return code must be EFI\_WRITE\_PROTECTED. |
| 5.7.5.3.9 | 0x7abcfa31, 0x7456, 0x40ae, 0x93, 0x51, 0x1c, 0xf4, 0x50, 0x1c, 0x08, 0xc9 | EFI\_BLOCK\_IO\_PROTOCOL.Writeblocks – WriteBlocks() returns EFI\_NO\_MEDIA with no media in the device. | For non-readonly device with a **MediaPresent** value of FALSE:  1. Call WriteBlocks() with valid parameter to write data to device.  Expected Behavior:  The return code must be EFI\_NO\_MEDIA. |
| 5.7.5.3.10 | 0x8a7d6ab3, 0x2c11, 0x41e3, 0xa4, 0x30, 0xfe, 0x3c, 0x50, 0xcc, 0x57, 0xad | EFI\_BLOCK\_IO\_PROTOCOL.Writeblocks – WriteBlocks() returns EFI\_INVALID\_PARAMETER with Buffer is not on proper lower alignment. | For non-readonly device with a **MediaPresent** value of TRUE and **IoAlign** more than 1:  1. Call WriteBlocks() with valid parameter and a Buffer value of (Buffer/**IoAlign**) \* **IoAlign** + Remainder (Remainder goes from 1 to Min(**IoAlign**-1, 5)).  Expected Behavior:  The return code must be EFI\_INVALID\_PARAMETER. |
| 5.7.5.3.11 | 0xb9d363bf, 0x9c50, 0x4671, 0x88, 0x55, 0xce, 0xfc, 0xc6, 0xb8, 0x24, 0xaa | EFI\_BLOCK\_IO\_PROTOCOL.Writeblocks – WriteBlocks() returns EFI\_INVALID\_PARAMETER with Buffer is not on proper alignment. | For non-readonly device with a **MediaPresent** value of TRUE and **IoAlign** more than 1:  1. Call WriteBlocks() with valid parameter and a Buffer value of (Buffer/**IoAlign**) \* **IoAlign** + Remainder (Remainder goes from **IoAlign**-1 down to Max(**IoAlign**-6, 1)).  Expected Behavior:  The return code must be EFI\_INVALID\_PARAMETER. |

### FlushBlocks()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.5.4.1 | 0x5f220c61, 0x24b5, 0x4c71, 0x8e, 0x5a, 0x78, 0xbd, 0x0a, 0xc6, 0x77, 0xf6 | EFI\_BLOCK\_IO\_PROTOCOL.FlushBlocks – FlushBlocks() returns EFI\_NO\_MEDIA with no media presented | For device with a **MediaPresent** value of FALSE and a WriteCaching value of TRUE:  1. Call FlushBlocks.  Expected Behavior:  The return code must be EFI\_NO\_MEDIA. |

### Media Info Check

|  |  |  |  |
| --- | --- | --- | --- |
| No | GUID | Assertion | Test Description |
| 5.7.5.5.1 | 0xb8a45208, 0xf7b0, 0x443c, 0x8c, 0xce, 0xeb, 0x81, 0xb6, 0x6c, 0x00, 0x4a | EFI\_BLOCK\_IO\_PROTOCOL. Media–  LogicalBlocksPerPhysicalBlockshould be 0 when LogicalPartitionis TRUE and Revision is greater than or equal to EFI\_BLOCK\_IO\_PROTOCOL\_REVISION2. | LogicalBlocksPerPhysicalBlock should be 0 when LogicalPartition is TRUE and Revision is greater than or equal to EFI\_BLOCK\_IO\_PROTOCOL\_REVISION2**.** |
| 5.7.5.5.2 | 0xe08ff5f4, 0x92de, 0x4cc9, 0x81, 0x22, 0x6b, 0x48, 0x7c, 0x67, 0x0c, 0x9b | EFI\_BLOCK\_IO\_PROTO  COL. Media–  OptimalTransferLengthGranularity should be 0 when LogicalPartitionis TRUE and Revision is greater than or equal to EFI\_BLOCK\_IO\_PROTOCOL\_REVISION3. | OptimalTransferLengthGranularity should be 0 when LogicalPartitionis TRUE and Revision is greater than or equal to EFI\_BLOCK\_IO\_PROTOCOL\_REVISION3. |

## EFI\_UNICODE\_COLLATION\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_UNICODE\_COLLATION\_ PROTOCOL Section.

### StriColl()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.6.1.1 | 0x3bf9028a, 0x599c, 0x44e0, 0xa7, 0xdf, 0xa6, 0x87, 0xcf, 0x9e, 0x15, 0xf4 | EFI\_UNICODE\_COLLATION\_PROTOCOL.StriColl - StriColl() with valid parameter returns correct status of comparison between String1 and String2. | 1. Call StriColl(). The return code should correspond to the string comparison result. |

### MetaiMatch()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.6.2.1 | 0x60291ba4, 0x7170, 0x4f5c, 0x84, 0x20, 0x11, 0x07, 0x85, 0x49, 0x2e, 0x6d | EFI\_UNICODE\_COLLATION\_PROTOCOL.MetaiMatch - MetaiMatch() returns correct status of pattern match. | 1. Call MetaiMatch(). The return code should correspond to the pattern match result. |

### 

### StrLwr()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.6.3.1 | 0x9d69a782, 0x672b, 0x43db, 0xac, 0x24, 0x16, 0x59, 0xa3, 0x9d, 0xa7, 0x5e | EFI\_UNICODE\_COLLATION\_PROTOCOL.StrLwr - StrLwr() convert the string to lowercase. | 1. Call StrLwr(). It should convert the string to lowercase. |
| 5.7.6.3.2 | 0x2e743a2a, 0x52a3, 0x411d, 0x95, 0x2a, 0x42, 0x0c, 0x47, 0x76, 0x90, 0x4c | EFI\_UNICODE\_COLLATION\_PROTOCOL.StrLwr - StrLwr() convert the string to lowercase. | 1. Call StrLwr() to convert string to lowercase and store lowercase string in buffer.  2. Call StrUpr() to convert lower case string to uppercase.  3. Call StrLwr() to convert uppercase string to lowercase. The lowercase string should be equal to lowercase string stored in buffer. |

### StrUpr()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.6.4.1 | 0x1b8390f4, 0xc5ac, 0x4342, 0x85, 0x55, 0x70, 0x74, 0xe5, 0xa2, 0x10, 0x2b | EFI\_UNICODE\_COLLATION\_PROTOCOL.StrUpr - StrUpr() convert the string to Uppercase. | 1. Call StrUpr(). It should convert the string to uppercase. |
| 5.7.6.4.2 | 0x6179f1fb, 0x54c5, 0x4844, 0xba, 0x17, 0x31, 0x4f, 0xe3, 0x57, 0xb4, 0xe3 | EFI\_UNICODE\_COLLATION\_PROTOCOL.StrUpr - StrUpr() convert the string to Uppercase. | 1. Call StrUpr() to convert string to uppercase and store uppercase string in buffer.  2. Call StrLwr() to convert upper case string to lowercase.  3. Call StrUpr() to convert lowercase string to uppercase. The uppercase string should be equal to uppercase string stored in buffer. |

### FatToStr()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.6.5.1 | 0x07f17163, 0x6f7d, 0x428f, 0xad, 0x13, 0xe4, 0xd0, 0x0b, 0x3a, 0x45, 0x64 | EFI\_UNICODE\_COLLATION\_PROTOCOL.FatToStr - FatToStr() with FatSize equal to the size of Fat String converts Fat string to Unicode string correctly. | 1. Call FatToStr() with FatSize equal to the size of Fat String. It should convert Fat string to Unicode string correctly. |
| 5.7.6.5.2 | 0x17ea04a7, 0xa56e, 0x4733, 0x83, 0x20, 0x79, 0x33, 0x09, 0x31, 0xef, 0xac | EFI\_UNICODE\_COLLATION\_PROTOCOL.FatToStr - FatToStr() with FatSize larger than the size of Fat String converts Fat string to Unicode string correctly | 1. Call FatToStr() with FatSize larger than the size of Fat String. It should convert Fat string to Unicode string correctly. |
| 5.7.6.5.3 | 0x2e89ebe3, 0x44bd, 0x4e02, 0xba, 0x50, 0x90, 0x05, 0xc0, 0xfc, 0x08, 0xdd | EFI\_UNICODE\_COLLATION\_PROTOCOL.FatToStr - FatToStr() with FatSize smaller than the size of Fat String converts Fat string to Unicode string correctly. | 1. Call FatToStr() with FatSize smaller than the size of Fat String. It should convert Fat string to Unicode string correctly. |

### StrToFat()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.6.6.1 | 0x6f780647, 0xef48, 0x4c1c, 0x87, 0xa6, 0x95, 0xe2, 0x50, 0x0e, 0x2e, 0x0b | EFI\_UNICODE\_COLLATION\_PROTOCOL.StrToFat - StrToFat() with FatSize equal to the size of Fat String converts Unicode string to Fat string correctly. | 1. Call StrToFat() with FatSize equal to the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns TRUE and characters were substituted with ‘\_’. |
| 5.7.6.6.2 | 0x5eea066e, 0xf73e, 0x4d36, 0x91, 0x25, 0xa0, 0x8a, 0x54, 0x6e, 0xee, 0x27 | EFI\_UNICODE\_COLLATION\_PROTOCOL.StrToFat - StrToFat() with FatSize larger than the size of Fat String converts Unicode string to Fat string correctly. | 1. Call StrToFat() with FatSize larger than the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns TRUE and characters were substituted with ‘\_’. |
| 5.7.6.6.3 | 0x58ae3ae9, 0x3dac, 0x41bf, 0x8d, 0x01, 0xd5, 0x91, 0xe3, 0xef, 0x62, 0x62 | EFI\_UNICODE\_COLLATION\_PROTOCOL.StrToFat - StrToFat() with FatSize smaller than the size of Fat String converts Unicode string to Fat string correctly. | 1. Call StrToFat() with FatSize smaller than the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns TRUE and characters were substituted with ‘\_’. |

## EFI\_UNICODE\_COLLATION2\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_UNICODE\_COLLATION2\_ PROTOCOL Section.

### StriColl()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.7.1.1 | 0x6a69637d, 0x5ada, 0x40fd, 0x93, 0x05, 0xe1, 0x06, 0xc9, 0xff, 0xa1, 0xbd | EFI\_UNICODE\_COLLATION2\_PROTOCOL.StriColl - StriColl() with valid parameter returns correct status of comparison between String1 and String2. | 1. Call StriColl(). The return code should correspond to the string comparison result. |

### MetaiMatch()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.7.2.1 | 0x49f68d03, 0xfef1, 0x460f, 0x8e, 0xdd, 0x27, 0xdb, 0x15, 0x22, 0xa3, 0xa3 | **EFI\_UNICODE\_COLLATION2\_PROTOCOL.MetaiMatch - MetaiMatch()** returns correct status of pattern match. | 1. Call **MetaiMatch()**. The return code should correspond to the pattern match result. |

### StrLwr()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.7.3.1 | 0xa8a08682, 0xf9d9, 0x4471, 0x85, 0x53, 0x48, 0x0b, 0x42, 0x1d, 0x66, 0x5b | EFI\_UNICODE\_COLLATION2\_PROTOCOL.StrLwr - StrLwr() convert the string to lowercase. | 1. Call StrLwr(). It should convert the string to lowercase. |
| 5.7.7.3.2 | 0xfb87853f, 0xa47b, 0x405b, 0x85, 0x5f, 0xc6, 0xbe, 0x18, 0x8b, 0xc3, 0x30 | EFI\_UNICODE\_COLLATION2\_PROTOCOL.StrLwr - StrLwr() convert the string to lowercase. | 1. Call StrLwr() to convert string to lowercase and store lowercase string in buffer.  2. Call StrUpr() to convert lower case string to uppercase.  3. Call StrLwr() to convert uppercase string to lowercase. The lowercase string should be equal to lowercase string stored in buffer. |

### StrUpr()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.7.4.1 | 0x6f390d73, 0xe8c7, 0x4032, 0xb5, 0xcb, 0xc0, 0xf6, 0xa8, 0x18, 0xe1, 0x87 | EFI\_UNICODE\_COLLATION2\_PROTOCOL.StrUpr - StrUpr() convert the string to Uppercase. | 1. Call StrUpr(). It should convert the string to uppercase. |
| 5.7.7.4.2 | 0xf559dbaa, 0xdeb6, 0x4591, 0xbf, 0x16, 0xf1, 0x4b, 0x50, 0x6c, 0xac, 0xae | EFI\_UNICODE\_COLLATION2\_PROTOCOL.StrUpr - StrUpr() convert the string to Uppercase. | 1. Call StrUpr() to convert string to uppercase and store uppercase string in buffer.  2. Call StrLwr() to convert upper case string to lowercase.  3. Call StrUpr() to convert lowercase string to uppercase. The uppercase string should be equal to uppercase string stored in buffer. |

### FatToStr()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number | GUID |  | Assertion |  | Test Description |
| 5.7.7.5.1 | 0x99a47923, 0xd2e9, 0x4114, 0xba, 0xc0, 0x46, 0x2b, 0xaa, 0x5a, 0xe5, 0xf3 | EFI\_UNICODE\_COLLATION2\_PROTOCOL.FatToStr - FatToStr() with FatSize equal to the size of Fat String converts Fat string to Unicode string correctly. | | 1. Call FatToStr() with FatSize equal to the size of Fat String. It should convert Fat string to Unicode string correctly. | |
| 5.7.7.5.2 | 0xd5dc3c74, 0x268a, 0x499d, 0xb3, 0x8b, 0xcb, 0x2d, 0x69, 0x71, 0x19, 0xb3 | EFI\_UNICODE\_COLLATION2\_PROTOCOL.FatToStr - FatToStr() with FatSize larger than the size of Fat String converts Fat string to Unicode string correctly | | 1. Call FatToStr() with FatSize larger than the size of Fat String. It should convert Fat string to Unicode string correctly. | |
| 5.7.7.5.3 | 0x305c644e, 0x002f, 0x466f, 0xae, 0x41, 0x4f, 0x22, 0xff, 0xda, 0x05, 0xfc | EFI\_UNICODE\_COLLATION2\_PROTOCOL.FatToStr - FatToStr() with FatSize smaller than the size of Fat String converts Fat string to Unicode string correctly. | | 1. Call FatToStr() with FatSize smaller than the size of Fat String. It should convert Fat string to Unicode string correctly. | |

### StrToFat()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.7.6.1 | 0x7b8b1cb5, 0xa3b9, 0x410d, 0x96, 0xf0, 0x3d, 0x88, 0x96, 0xe9, 0x03, 0x9a | EFI\_UNICODE\_COLLATION2\_PROTOCOL.StrToFat - StrToFat() with FatSize equal to the size of Fat String converts Unicode string to Fat string correctly. | 1. Call StrToFat() with FatSize equal to the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns TRUE and characters were substituted with ‘\_’. |
| 5.7.7.6.2 | 0x9c40c459, 0x0a09, 0x4382, 0x89, 0x79, 0x01, 0xea, 0x30, 0x35, 0xdd, 0xf4 | EFI\_UNICODE\_COLLATION2\_PROTOCOL.StrToFat - StrToFat() with FatSize larger than the size of Fat String converts Unicode string to Fat string correctly. | 1. Call StrToFat() with FatSize larger than the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns TRUE and characters were substituted with ‘\_’. |
| 5.7.7.6.3 | 0x8d0e58cc, 0x4494, 0x4684, 0xaf, 0x6c, 0xdc, 0xf9, 0x1b, 0x77, 0x6c, 0x6b | EFI\_UNICODE\_COLLATION2\_PROTOCOL.StrToFat - StrToFat() with FatSize smaller than the size of Fat String converts Unicode string to Fat string correctly. | 1. Call StrToFat() with FatSize smaller than the size of Fat String. It should convert Unicode string to Fat string correctly. If one or more conversions failed, it returns TRUE and characters were substituted with ‘\_’. |

## EFI\_ATA\_PASS\_THRU\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_ATA\_PASS\_THRU\_PROTOCOL Section .

### GetNextPort()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.8.1.1 | 0xbad50e59, 0x9423, 0x427d, 0xa7, 0x5d, 0x69, 0x1c, 0x90, 0xb7, 0xf9, 0x75 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.GetNextPort – GetNextPort() should return invalid parameter if input port is invalid. | 1. Call GetNextPort() with Port being a not available port.  2. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.7.8.1.2 | 0xc3e87aa1, 0x6e9c, 0x478f, 0x9b, 0xd5, 0x39, 0x50, 0x8, 0x01, 0x28, 0x96 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.GetNextPort – GetNextPort()  should return invalid parameter if port is not 0xFFFF and port was not returned on a previous call. | 1. Call GetNextPort() when Port is not 0xFFFF and Port was not returned on a previous call.  2. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.7.8.1.3 | 0x5f658292,  0xa409,  0x4d67, 0xba,  0x13, 0x4,  0xc2, 0x51,  0x85, 0xf2, 0x80 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.GetNextPort – GetNextPort()could iterate all available port. | 1. Call GetNextPort() with Port as 0xFFFF to start iterate ports.  2. The iteration should ended up with a return code EFI\_NOT\_FOUND. |

### BuildDevicePath()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.8.2.1 | 0xd72e6a78, 0x5292, 0x4493, 0x90, 0x40, 0xb0, 0x44, 0x5a, 0x9c, 0x17, 0x14 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.BuildDevicePath – BuildDevicePath() with NULL parameter. | 1. Call BuildDevicePath() with with Port and PortMultiplierPort identifying an available device and DevicePath being NULL. The return code should be EFI\_INVALID\_PARAMETER |
| 5.7.8.2.2 | 0xa42a0e01, 0x7b80, 0x46e4, 0xa7, 0x57, 0x86, 0xc4, 0xec, 0x53, 0xf4, 0xe4 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.BuildDevicePath – BuildDevicePath() with invalid port. | 1. Call BuildDevicePath() with with Port being not available and other parameters valid. The return code should be EFI\_NOT\_FOUND |
| 5.7.8.2.3 | 0x322f00c1, 0xf6bf, 0x41ed, 0xae, 0xfd, 0xaa, 0xc4, 0x8f, 0x3f, 0xa9, 0xdb | EFI\_ATA\_PASS\_THRU\_PROTOCOL.BuildDevicePath – BuildDevicePath() with invalid device. | 1. Call BuildDevicePath() with with PortMultiplierPort being not available and other parameters valid. The return code should be EFI\_NOT\_FOUND |
| 5.7.8.2.4 | 0x230d44b6, 0xce53, 0x42b6, 0x9b, 0xa6, 0x3d, 0x11, 0x5d, 0x49, 0x2b, 0x33 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.BuildDevicePath – BuildDevicePath() with available device, device path should be created. | 1. Call BuildDevicePath() with with Port and PortMultiplierPort identifying an available device. The return code should be EFI\_SUCCESS |

### GetDevice()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.8.3.1 | 0x0f2f0849, 0x690b, 0x48ea, 0x8e, 0x35, 0x64, 0x36, 0x3f, 0xaa, 0x8c, 0x5c | EFI\_ATA\_PASS\_THRU\_PROTOCOL.GetDevice – GetDevice() with NULL device path. | 1. Call GetDevice() with the DevicePath being NULL. The Port and PortMultiplierPort are valid, the return code should be EFI\_INVALID\_PARAMETER |
| 5.7.8.3.2 | 0x7602bd0a, 0x1c05, 0x49e5, 0xa8, 0xd4, 0xc6, 0x3, 0x8c, 0x43, 0x9a, 0xf9 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.GetDevice – GetDevice() with NULL port. | 1. Call GetDevice() with the DevicePath being valid. The Port being NULL, PortMultiplierPort is valid, the return code should be EFI\_INVALID\_PARAMETER |
| 5.7.8.3.3 | 0x2b64d49a, 0x1f1b, 0x4610, 0xa2, 0x66, 0xde, 0x32, 0xa1, 0x7, 0x2b, 0x32 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.GetDevice – GetDevice() with NULL device. | 1. Call GetDevice() with the DevicePath being valid. The Port being valid, PortMultiplierPort is NULL, the return code should be EFI\_INVALID\_PARAMETER |
| 5.7.8.3.4 | 0x07830eaf, 0xba30, 0x4224, 0xab, 0xc4, 0x42, 0x42, 0x8b, 0x7a, 0x4, 0x5d | EFI\_ATA\_PASS\_THRU\_PROTOCOL.GetDevice – GetDevice() with invalid device path. | 1. Call GetDevice() with the DevicePath being of type ‘End Device Path’. The Port and PortMultiplierPort are valid, the return code should be EFI\_UNSUPPORTED or EFI\_NOT\_FOUND |
| 5.7.8.3.5 | 0x7ea827e4, 0x522c, 0x44b6, 0x99, 0xe4, 0x25, 0x93, 0x19, 0xba, 0xcc, 0x57 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.GetDevice – GetDevice() with correct device path. The device represented by this device path should be achieved. | 1. Call GetDevice() with the DevicePath that representing one available device. The Port and PortMultiplierPort of the device should be got and the return code should be EFI\_SUCCESS |

### ResetPort()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.8.4.1 | 0x5e0080d2, 0x4065, 0x4b92, 0xa4, 0x61, 0x52, 0x49, 0xf3, 0x8f, 0xaf, 0x55 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.ResetPort – ResetPort() with available port. | 1. Call ResetPort() with Port as one available port. The return code should be EFI\_SUCCESS or EFI\_UNSUPPORTED. |

### ResetDevice()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.8.5.1 | 0x206ae2fc, 0x3f34, 0x4afe, 0x82, 0x44, 0x40, 0x27, 0x57, 0x60, 0x98, 0x31 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.ResetDevice – ResetDevice() with invalid port. | 1. Call ResetDevice() with Port being invalid and PortMultiplierPort as zero. The return code should be EFI\_INVALID\_PARAMETER |
| 5.7.8.5.2 | 0xd9378047, 0x9b4b, 0x4abf, 0xaa, 0x6b, 0xe3, 0xcd, 0xb6, 0xc4, 0x19, 0x39 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.ResetDevice – ResetDevice() with invalid device. | 1. Call ResetDevice() with Port being valid and PortMultiplierPort being invalid. The return code should be EFI\_INVALID\_PARAMETER |
| 5.7.8.5.3 | 0xa400bc81, 0x9e48, 0x469b, 0xa0, 0x97, 0xd0, 0x8, 0x45, 0xb6, 0x69, 0xe8 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.ResetDevice – ResetDevice() with available device. | 1. Call ResetDevice() with Port and PortMultiplierPort as one available device. The return code should be EFI\_SUCCESS or EFI\_UNSUPPORTED |

### GetNextDevice()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.8.6.1 | 0xc564ad60, 0x32ce, 0x4f5f, 0x86, 0x7a, 0xef, 0x9f, 0xef, 0x5e, 0x94, 0xa2 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.GetNextDevice – GetNextDevice() with invalid device number. | 1. Call GetNextPort() with Port as 0xFFFF to start iterate ports.  2. Call GetNextDevice() with Port as one available port and PortMultiplierPort being invalid.  3. The iteration should ended up with a return code EFI\_INVALID\_PARAMETER. |
| 5.7.8.6.2 | 0x0e5c99ba, 0xd36c, 0x4775, 0x91, 0x31, 0x76, 0x6a, 0x6e, 0x8c, 0x53, 0x6b | EFI\_ATA\_PASS\_THRU\_PROTOCOL.GetNextDevice – GetNextDevice() should return invalid parameter if PortMultiplierPort is not 0xFFFF and PortMultiplierPort was not returned on a previous call. | 1. Call GetNextPort()when PortMultiplierPort is not 0xFFFF and PortMultiplierPort was not returned on a previous call.  2. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.7.8.6.3 | 0xd89631f3,  0xbd59,  0x4959, 0xba,  0x10, 0x3f,  0xa9, 0x94,  0x62, 0x02, 0xdf | EFI\_ATA\_PASS\_THRU\_PROTOCOL.GetNextDevice – GetNextDevice()  could iterate all available  devices on one port. | 1.GetNextPort() with Port as  0xFFFFto start iterate ports.  2. Call GetNextPort() with Port  as one available port and  PortMultiplierPort as 0xFFFF to  start iterate devicess.  3. The iteration should ended up with a return code EFI\_NOT\_FOUND. |

### PassThru()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.8.7.1 | 0x7d6fcacd, 0x3463, 0x41c8, 0xa5, 0x1, 0xa2, 0x99, 0x40, 0x44, 0x59, 0xb8 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.PassThru – PassThru() with Non-IoAligned InDataBuffer. | 1. Call PassThru()with Event being NULL Packet.InDataBuffer set to be not aligned with EFI\_ATA\_PASS\_THRU\_PROTOCOL.Mode.IoAlign. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.7.8.7.2 | 0x745295b5, 0xc36b, 0x4b23, 0xaf, 0xc7, 0xd4, 0xcc, 0xc0, 0x1d, 0xb6, 0x4f | EFI\_ATA\_PASS\_THRU\_PROTOCOL.PassThru – PassThru() with Non-IoAligned Asb. | 1. Call PassThru()with Event being NULL Packet.Asb set to be not aligned with EFI\_ATA\_PASS\_THRU\_PROTOCOL.Mode.IoAlign. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.7.8.7.3 | 0xaf9489a2, 0x23f3, 0x4962, 0x9d, 0x8f, 0xd2, 0xc0, 0xa7, 0xcb, 0x2f, 0xb1 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.PassThru – PassThru() with Non-IoAligned OutDataBuffer. | 1. Call PassThru()with Event being NULL Packet.OutDataBuffer set to be not aligned with EFI\_ATA\_PASS\_THRU\_PROTOCOL.Mode.IoAlign. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.7.8.7.4 | 0xd584b074, 0xa8cd, 0x438c, 0xb5, 0x18, 0xb1, 0xec, 0x59, 0xfa, 0xc8, 0xee | EFI\_ATA\_PASS\_THRU\_PROTOCOL.PassThru – PassThru() with invalid port. | 1. Call PassThru()with Event being NULL Packet contents valid, Port as invalid. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.7.8.7.5 | 0x4cd806fd, 0x3742, 0x44e9, 0xa6, 0x19, 0xdf, 0x2d, 0x37, 0x47, 0xe7, 0x8f | EFI\_ATA\_PASS\_THRU\_PROTOCOL.PassThru – PassThru() with invalid device. | 1. Call PassThru()with Event being NULL Packet contents valid, Port as valid, PortMultiplierPort being invalid. The return code should be EFI\_INVALID\_PARAMETER. |
| 5.7.8.7.6 | 0xa648ab45, 0x898b, 0x4b44, 0xab, 0x9e, 0x24, 0x6b, 0xc6, 0x49, 0xc9, 0xfd | EFI\_ATA\_PASS\_THRU\_PROTOCOL.PassThru – PassThru() with too long buffer size. | 1. Call PassThru()with Event being NULL Packet.InDataBufferLength being 0xFFFFFFFF, Port and PortMultiplierPort being valid. The return code should be EFI\_BAD\_BUFFER\_SIZE. |
| 5.7.8.7.7 | 0xe5c8314a, 0xa2b8, 0x42d2, 0xb1, 0x27, 0x97, 0xad, 0x78, 0x74, 0xd5, 0x30 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.PassThru – PassThru() sends ATA command ‘Identify Device’ to an available device with several valid EFI\_ATA\_PASS\_THRU\_COMMAND\_PACKET and EFI\_EVENT inputs. | Below are the three possible separate test procedure that corresponds to this test assertion  1. Call PassThru()with Port and PortMultiplierPort representing one available device. The Packet.Acb.AtaCommand is set to be the value of ‘Identify Device’ command 0xEC, Packet.Asb and Packet.InDataBuffer are allocated and adjusted according to EFI\_ATA\_PASS\_THRU\_PROTOCOL.Mode.IoAlign value, Packet.Timeout set to be 2 seconds, Packet.Length set to be block granularity, Packet.InTransferLength being 1 to indicate one block, Packet.Protocol being EFI\_ATA\_PASS\_THRU\_PROTOCOL\_PIO\_DATA\_IN. The return code should be EFI\_SUCCESS and Packet.Asb.AtaStatus should reflect the ATA command has been executed successfully. |
|  |  |  | 2. Call PassThru()with Port and PortMultiplierPort representing one available device. The Packet.Acb.AtaCommand is set to be the value of ‘Identify Device’ command 0xEC, Packet.Asb and Packet.InDataBuffer are allocated and adjusted according to EFI\_ATA\_PASS\_THRU\_PROTOCOL.Mode.IoAlign value, Packet.Timeout set to be 2 seconds, Packet.Length set to be byte granularity, Packet.InTransferLength being 512 to indicate one block, Packet.Protocol being EFI\_ATA\_PASS\_THRU\_PROTOCOL\_PIO\_DATA\_IN. The return code should be EFI\_SUCCESS and Packet.Asb.AtaStatus should reflect the ATA command has been executed successfully. |
|  |  |  | 3. Call PassThru()with Port and PortMultiplierPort representing one available device. Event being a callback-TPL event with a notification function that updates a global vairable. By checking EFI\_ATA\_PASS\_THRU\_PROTOCOL.Mode.Attributes to determine whether non-blocking IO is supported. The Packet.Acb.AtaCommand is set to be the value of ‘Identify Device’ command 0xEC, Packet.Asb and Packet.InDataBuffer are allocated and adjusted according to EFI\_ATA\_PASS\_THRU\_PROTOCOL.Mode.IoAlign value, Packet.Timeout set to be 2 seconds, Packet.Length set to be block granularity, Packet.InTransferLength being 1 to indicate one block, Packet.Protocol being EFI\_ATA\_PASS\_THRU\_PROTOCOL\_PIO\_DATA\_IN. If non-blocking mode is not supported, the global variable should keep unchanged. The return code should be EFI\_SUCCESS and Packet.Asb.AtaStatus should reflect the ATA command has been executed successfully. |
| 5.7.8.7.8 | 0xeb7841b9, 0x2a4a, 0x45b1, 0xa9, 0x9f, 0x67, 0x7a, 0xb4, 0xcd, 0x79, 0xa2 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.PassThru – PassThru() returns EFI\_SUCCESS. | 1. Call PassThru()with Event being NULL Packet.Length set to be block granularity. The return code should be EFI\_SUCCESS. |
| 5.7.8.7.9 | 0x9662da7d, 0x6f98, 0x4051, 0xb1, 0x87, 0x85, 0xb0, 0xf4, 0xb5, 0x3a, 0xf1 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.PassThru – PassThru() returns EFI\_SUCCESS. | 1. Call PassThru()with Event being NULL Packet.Length set to be byte granularity. The return code should be EFI\_SUCCESS. |
| 5.7.8.7.10 | 0x5787ed6f, 0xa984, 0x4b15, 0xb2, 0xf3, 0xa0, 0xd1, 0xb8, 0xce, 0x61, 0x89 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.PassThru – PassThru() returns EFI\_SUCCESS. | 1. Call PassThru()with Event being a callback-TPL event with a notification function that updates a global variable. By checking EFI\_ATA\_PASS\_THRU\_PROTOCOL.Mode.Attributes to determine whether non-blocking IO is supported. If supported, the global variable will be updated in the event’s notification function and the return code should be EFI\_SUCCESS. |
| 5.7.8.7.11 | 0x202b3252, 0x5c89, 0x41bf, 0x9b, 0x42, 0x94, 0x58, 0x56, 0xc8, 0xcc, 0x7e | EFI\_ATA\_PASS\_THRU\_PROTOCOL.PassThru – PassThru() returns EFI\_SUCCESS. | 1. Call PassThru()with Event being a  callback-TPL event with a notification  function that updates a global variable.  Packet.Length set to block  granularity. By checking  EFI\_ATA\_PASS\_THRU\_PROTOCOL.Mode.Attributes to determine whether nonblockingIO is supported. If supported, the global variable will be updated in the event’s notification function and the return code should be EFI\_SUCCESS. |

### Mode Conformance

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.8.8.1 | 0xdcb2c498, 0x4d12, 0x4351, 0xb4, 0xd7, 0x85, 0x33, 0x2c, 0x51, 0xd8, 0xf7 | EFI\_ATA\_PASS\_THRU\_PROTOCOL.Mode – Mode attributes should be physical, logical or both. | 1. Check Mode.Attributes to be EFI\_ATA\_PASS\_THRU\_ATTRIBUTES\_PHYSICAL , EFI\_ATA\_PASS\_THRU\_ATTRIBUTES\_LOGICAL or EFI\_ATA\_PASS\_THRU\_ATTRIBUTES\_PHYSICAL | EFI\_ATA\_PASS\_THRU\_ATTRIBUTES\_LOGICAL |
| 5.7.8.8.2 | 0x8ccb89ab, 0x2bbe, 0x4766, 0xa9, 0x5, 0x2d, 0x1e, 0xa6, 0xb4, 0x54, 0x6b | EFI\_ATA\_PASS\_THRU\_PROTOCOL.Mode – Mode IoAlign should be 0, 1 or a power of 2. | Check Mode.IoAlign to be 0, 1 or a power of 2. |

## EFI\_BLOCK\_IO2\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_BLOCK\_IO2\_PROTOCOL Section.

### ReadBlocksEx()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.9.1.1 | 0x36a2dbdb, 0x6d88, 0x4807,0xaf, 0xa5, 0x7b, 0xef, 0xc4, 0xb1, 0xfe, 0xaa | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_MEDIA\_CHANGED with invalid MediaID | 1. Sync & Async Call ReadBlockEx() with invalid MediaID. The return code should be EFI\_MEDIA\_CHANGED |
| 5.7.9.1.2 | 0x45d515fd, 0xa64f, 0x47bd, 0x9a, 0x84, 0x1f, 0xe4, 0x86, 0x6a, 0x32, 0x8a | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_BAD\_BUFFER\_SIZE with bad blocksize | 1.Sync & Asnyc Call ReadblockEx()with BufferSize not being a multiple of the intrinsic block size of the device. The return code should be EFI\_BAD\_BUFFER\_SIZE |
| 5.7.9.1.3 | 0x896937aa, 0x65ba, 0x4354, 0xab, 0xf7, 0xd8, 0x4f, 0xe8, 0x9f, 0xbc, 0x8 | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_INVALID\_PARAMETER with invalid LBA parameter | 1. Sync & Async Call ReadblockEx() call with invalid LBA parameter. The return code should be EFI\_INVALID\_PARAMETER |
| 5.7.9.1.4 | 0xd54c2dc4, 0x8fed, 0x4ce1, 0xac, 0x7b, 0xc6, 0x7a, 0x48, 0x4e, 0x2, 0x7 | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_INVALID\_PARAMETER with return data which are smaller thanBufferSize passed in | 1. Sync & Async Call ReadblockEx() returns data which are smaller than BufferSize passed in. The return code should be EFI\_INVALID\_PARAMETER |
| 5.7.9.1.5 | 0xc75d447c, 0x29c5, 0x4882, 0x80, 0xc5, 0x42, 0x67, 0xe0, 0xa2, 0x5c, 0xfc | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_INVALID\_PARAMETERwith block alignment should be power of 2 | 1.Sync & Async Call ReadblockEx() block alignment should be power of 2. The return code should be EFI\_INVALID\_PARAMETER |
| 5.7.9.1.6 | 0x1ce01e1c, 0xedde, 0x4a37, 0x96, 0x2f, 0x3a, 0x32, 0x8a, 0x54, 0xc1, 0xf5 | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_NO\_MEDIA when read from device without media present in the device | 1.Sync & Async Call ReadblockEx()from device without media present in the device. The return code should beEFI\_NO\_MEDIA |
| 5.7.9.1.7 | 0x47b8309d, 0xf783, 0x4679, 0x95, 0xbb, 0x47, 0x58, 0x10, 0x7, 0x2, 0x7c | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_SUCCESS when Async call with proper parameter from valid media | 1. Async Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.1.8 | 0x7abe441a, 0x7118, 0x4394, 0x81, 0xb8, 0xb9, 0x22, 0xa2, 0x87, 0xd2, 0x1f | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_SUCCESS when Async call with proper parameter from valid media | 1. Async Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.1.9 | 0x3167dc14, 0xcf85, 0x4158, 0x9c, 0xec, 0x7a, 0x3, 0xdd, 0xc, 0xfd, 0xa1 | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_SUCCESS when Async call with proper parameter from valid media. The async registered events haven’t be signaled. | 1. Async Call ReadblockEx() from device with proper parameter from valid media. All events should be signaled successfully. |
| 5.7.9.1.10 | 0xc4726d6f, 0x148e, 0x4a06, 0xa0, 0x92, 0xd4, 0x6b, 0xa8, 0x7c, 0x16, 0x63 | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_SUCCESS when Sync call with proper parameter from valid media | 1. Sync Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.1.11 | 0x6e61c6ee, 0x2328, 0x45c5, 0x99, 0xe9, 0xcb, 0x66, 0x99, 0xcf, 0x56, 0xe | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_SUCCESSwhen Batch Async call with proper parameter from valid media | 1. Batch Async Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.1.12 | 0x639fca8b, 0x394e, 0x4c1a, 0x81, 0x15, 0xc2, 0x55, 0xc6, 0xbd, 0x4b, 0x95 | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_SUCCESS when Mixed Sync & Async call with proper parameter from valid media. Async Read Call failed | 1. Mixed Sync & Async Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.1.13 | 0x6d90ad93, 0xf492, 0x4a10, 0xa0, 0xbc, 0xb5, 0x30, 0x54, 0x36, 0x28, 0x54 | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_SUCCESS when Mixed Sync & Async call with proper parameter from valid media. | 1. Mixed Sync & Async Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.1.14 | 0x7ce315f8, 0xb5d1, 0x4691, 0x92, 0x63, 0x66, 0x4b, 0xe0, 0x70, 0x85, 0x47 | EFI\_BLOCK\_IO2\_PROTOCOL. ReadBlocksEx – ReadBlocksEx() returns EFI\_SUCCESS when Mixed Sync & Async call with proper parameter from valid media. Sync Read Call failed | 1. Mixed Sync & Async Call ReadblockEx() from device with proper parameter from valid media. The return code should be EFI\_SUCCESS |

### WriteBlocksEx()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.9.2.1 | 0xbe0e99b7, 0x62a0, 0x45ff, 0x92, 0x11, 0x55, 0xe7, 0xe4, 0xda, 0xa, 0x86 | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx – WriteBlocksEx() returns EFI\_MEDIA\_CHANGED with invalidMediaID | 1. Sync & Async Call WriteBlockEx() with invalid MediaID. The return code should be EFI\_MEDIA\_CHANGED |
| 5.7.9.2.2 | 0x7253b26e, 0xbb34, 0x49fa, 0x92, 0xe, 0xfa, 0x5f, 0xef, 0xa6, 0xf5, 0x5a | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx – WriteBlocksEx() returns EFI\_BAD\_BUFFER\_SIZE with bad blocksize | 1.Sync & Asnyc Call WriteBlocksEx()with BufferSize not being a multiple of the intrinsic block size of the device. The return code should be EFI\_BAD\_BUFFER\_SIZE |
| 5.7.9.2.3 | 0x34009928, 0x8f89, 0x42d3, 0xb0, 0x20, 0x9f, 0x7f, 0x54, 0xef, 0x75, 0xe2 | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx – WriteBlocksEx() returns EFI\_INVALID\_PARAMETERwith invalidLBA parameter | 1. Sync & Async Call WriteBlocksEx() call with invalid LBA parameter. The return code should be EFI\_INVALID\_PARAMETER |
| 5.7.9.2.4 | 0x3ca09c43, 0xfd3f, 0x4e88, 0x92, 0xc3, 0x97, 0xe6, 0xe1, 0x76, 0xb7, 0x3c | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx– WriteBlocksEx() returns EFI\_INVALID\_PARAMETER with block alignment should be power of 2 | 1.Sync & Async Call WriteBlocksEx() block alignment should be power of 2. The return code should be EFI\_INVALID\_PARAMETER |
| 5.7.9.2.5 | 0x27c8f6f8, 0x984d, 0x49ff, 0xa6, 0xab, 0xf9, 0xc1, 0x21, 0x39, 0x2f, 0x7e | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx– WriteBlocksEx() returns EFI\_INVALID\_PARAMETERwith unaligned data buffer | 1.Sync & Async Call WriteBlocksEx() unaligned data buffer. The return code should be EFI\_INVALID\_PARAMETER |
| 5.7.9.2.6 | 0xf3807d6, 0x930e, 0x4ea2, 0xaa, 0xd9, 0x54, 0xc6, 0x73, 0xe9, 0xb5, 0x51 | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx– WriteBlocksEx() returns EFI\_NO\_MEDIA when write to device without media present in the device | 1.Sync & Async Call WriteBlocksEx() to device without media present in the device. The return code should be EFI\_NO\_MEDIA |
| 5.7.9.2.7 | 0xb9c4f106, 0x6658, 0x430f, 0x87, 0xcb, 0xe0, 0xde, 0x2c, 0xbf, 0xb9, 0x5c | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx– WriteBlocksEx() when write to a read-only device | 1.Sync & Async Call WriteBlocksEx() to read-only device. The return code should be EFI\_NO\_MEDIA |
| 5.7.9.2.8 | 0x1a0cf746, 0xe5bf, 0x4b0d, 0x84, 0xf, 0x2e, 0x2b, 0xfe, 0x94, 0xdc, 0xdc | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx – WriteBlocksEx() returns EFI\_SUCCESS when Async call with proper parameter to valid media | 1. Async Call WriteBlocksEx() to device with proper parameter to valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.2.9 | 0xf8a6b15d, 0xc85b, 0x4b59, 0xbe, 0x7f, 0x2f, 0x84, 0xf9, 0x77, 0xe, 0x4a | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx– WriteBlocksEx()returns EFI\_SUCCESS when Async call with proper parameter to valid media | 1. Async Call WriteBlocksEx() to device with proper parameter to valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.2.10 | 0x964c1c44, 0xb693, 0x43ca, 0x88, 0xce, 0xb5, 0x34, 0xa6, 0x42, 0xff, 0x80 | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx– WriteBlocksEx() returns EFI\_SUCCESSwhen Async call with proper parameter to valid media. The async registered events haven’t be signaled. | 1. Async Call WriteBlocksEx() to device with proper parameter to valid media. All events should be signaled successfully. |
| 5.7.9.2.11 | 0xad588be4, 0x138f, 0x4874, 0x92, 0xf, 0xef, 0xa1, 0xd3, 0x79, 0xe5, 0x17 | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx– WriteBlocksEx() returns EFI\_SUCCESS when Sync call with proper parameter to valid media | 1. Sync Call WriteBlocksEx() to device with proper parameter to valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.2.12 | 0x9e70ff56, 0x7e0e, 0x404f, 0xab, 0x10, 0x7f, 0x6a, 0x1e, 0x1f, 0xbb, 0xf7 | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx– WriteBlocksEx() returnsEFI\_SUCCESS when Batch Async call with proper parameter to valid media | 1. Batch Async Call **WriteBlocksEx()** to device with proper parameter to valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.2.13 | 0x6d41db68, 0xffe3, 0x4676, 0x89, 0xba, 0xe4, 0xc8, 0xdb, 0xc6, 0x4f, 0xcc | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx– WriteBlocksEx() returns EFI\_SUCCESS when Mixed Sync & Async call with proper parameter to valid media. Async Read Call failed | 1. Mixed Sync & Async Call WriteBlocksEx() to device with proper parameter to valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.2.14 | 0xe532b760, 0xd561, 0x43be, 0xa5, 0x51, 0x62, 0xb5, 0x63, 0x16, 0x9f, 0xbc | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx– WriteBlocksEx() returns EFI\_SUCCESS when Mixed Sync & Async call with proper parameter to valid media. | 1. Mixed Sync & Async Call WriteBlocksEx() to device with proper parameter to valid media. The return code should be EFI\_SUCCESS |
| 5.7.9.2.15 | 0x11a6bb4a, 0xa943, 0x4006, 0xbc, 0xb0, 0x57, 0x6c, 0xb8, 0x68, 0xae, 0x95 | EFI\_BLOCK\_IO2\_PROTOCOL. WriteBlocksEx– WriteBlocksEx() returns EFI\_SUCCESSwhen Mixed Sync & Async call with proper parameter to valid media. Sync Read Call failed | 1. Mixed Sync & Async Call WriteBlocksEx() to device with proper parameter to valid media. The return code should be EFI\_SUCCESS |

### FlashBlocksEx()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.9.3.1 | 0x457168d, 0x1ded, 0x4c01, 0xb7, 0x84, 0xa7, 0xa9, 0xd6, 0xaa, 0xb, 0x81 | EFI\_BLOCK\_IO2\_PROTOCOL. FlashBlocksEx– FlashBlocksEx() returns EFI\_NO\_MEDIA with a device with no media | 1. Sync & Async Call FlashBlocksEx() with no Media on device The return code should be EFI\_NO\_MEDIA |
| 5.7.9.3.2 | 0x6a1de6c8, 0xe02b, 0x4a50, 0x80, 0x6d, 0x9a, 0x51, 0x5c, 0xc1, 0xe4, 0xe3 | EFI\_BLOCK\_IO2\_PROTOCOL. FlashBlocksEx– FlashBlocksEx() returns EFI\_WRITE\_PROTECTED with a read-only device with media | 1. Sync & Async Call FlashBlocksEx() with a read-only media on device The return code should be EFI\_WRITE\_PROTECTED |
| 5.7.9.3.3 | 0xc97de60f, 0x87cf, 0x45b9, 0x98, 0x9b, 0x8, 0x9d, 0x3, 0xe0, 0xf3, 0xf6 | EFI\_BLOCK\_IO2\_PROTOCOL. FlashBlocksEx– FlashBlocksEx() returns EFI\_SUCCESS with a right device with media & all event signaled should be signaled | 1. Async Call FlashBlocksEx() with a media on a right device The return code should be EFI\_SUCCESS |

### Media Info Check

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.9.4.1 | 0x8251405e, 0xe716, 0x4ecd, 0x83, 0x55, 0xc9, 0xf5, 0x60, 0x4b, 0xf2, 0x4d | EFI\_BLOCK\_IO2\_PROTO  COL. Media–  LogicalBlocksPerPhysicalBlock should be 0 when LogicalPartition is TRUE and Revision is greater than or equal to EFI\_BLOCK\_IO\_PROTOCOL\_REVISION2. | LogicalBlocksPerPhysicalBlock should be 0 when LogicalPartition is TRUE and Revision is greater than or equal to EFI\_BLOCK\_IO\_PROTOCOL\_REVISION2. |
| 5.7.9.4.2 | 0x6739b945, 0x2498, 0x4a1c, 0x87, 0xb0, 0x85, 0xa4, 0xbe, 0xf6, 0x53, 0x7c | EFI\_BLOCK\_IO2\_PROTO  COL. Media–  OptimalTransferLengthGranularity should be 0 when LogicalPartition is TRUE and Revision is greater than or equal to EFI\_BLOCK\_IO\_PROTOCOL\_REVISION3. | OptimalTransferLengthGranularity should be 0 when LogicalPartition is TRUE and Revision is greater than or equal to EFI\_BLOCK\_IO\_PROTOCOL\_REVISION3. |

## EFI\_STORAGE\_SECURITY\_COMMAND\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_STORAGE\_SECURITY\_COMMAND\_PROTOCOL Section.

### ReceiveData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.10.1.1 | 0x35749acf, 0xeed8, 0x4230, 0xbc, 0x18, 0xde, 0x1f, 0x8b, 0x7c, 0xfa, 0xef | EFI\_STORAGE\_SECURITY\_ COMMAND.ReceiveData – ReceiveData() should not return EFI ERROR. When PayloadBufferSize is too small | Call ReceiveData () with PayloadBufferSize =10 & TCG command 0 to return security protocol info  The return status should not be EFI Error Status |
| 5.7.10.1.2 | 0x8e742768, 0x229a, 0x4aaa, 0xb5, 0x9d, 0xc9, 0xb2, 0x6e, 0x32, 0x44, 0x58 | EFI\_STORAGE\_SECURITY\_ COMMAND.ReceiveData – ReceiveData() should return EFI\_MEDIA\_CHANGED. When MediaID is not correct | Call ReceiveData () with Wrong MediaID & TCG command 0 to return security protocol info  The return status should be EFI\_MEDIA\_CHANGED |
| 5.7.10.1.3 | 0x2fe7a174, 0xa8a1, 0x45b3, 0x91, 0x5c, 0x1e, 0xd, 0x59, 0x13, 0x17, 0xa6 | EFI\_STORAGE\_SECURITY\_ COMMAND.ReceiveData – ReceiveData() should return EFI\_INVALID\_PARAMETER. When PayloadBuffer is NULL | Call ReceiveData() with NULL PayloadBuffer & TCG command 0 to return security protocol info  The return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.10.1.4 | 0xa55d41c7, 0x0ca4, 0x4ab1, 0xaa, 0x7b, 0x82, 0xee, 0x74, 0x30, 0xcf, 0x9d | EFI\_STORAGE\_SECURITY\_ COMMAND.ReceiveData – ReceiveData() should return EFI\_INVALID\_PARAMETER. When PayloadTransferSize is not NULL | Call ReceiveData() with NULL PayloadTransferSize & TCG command 0 to return security protocol info  The return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.10.1.5 | 0xcc0223b7, 0xf088, 0x4ea1, 0xa6, 0xcb, 0x73, 0x93, 0x8, 0x4e, 0x9f, 0xf2 | EFI\_STORAGE\_SECURITY\_ COMMAND.ReceiveData – ReceiveData()should return EFI\_NO\_MEDIA. When There is no media present | Call ReceiveData () with TCG command 0 to return security protocol info on a no media device  The return status should be EFI\_NO\_MEDIA. |

### SendData()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.10.2.1 | 0x2e6fddd2, 0xce3b, 0x49fb, 0xa1, 0xd4, 0x43, 0xa2, 0x99, 0xf7, 0xec, 0xc2 | EFI\_STORAGE\_SECURITY\_ COMMAND.SendData – SendData() should return EFI\_MEDIA\_CHANGED. When MediaIDis not correct | Call SendData ()with Wrong MediaID & TCG command 0 to return security protocol info  The return status should be EFI\_MEDIA\_CHANGED |
| 5.7.10.2.2 | 0x2323be1a, 0xf73a, 0x46d5, 0xa2, 0x24, 0xbf, 0x9a, 0x7f, 0x6a, 0x53, 0x96 | **E**FI\_STORAGE\_SECURITY\_ COMMAND.SendData – SendData() should return EFI\_INVALID\_PARAMETER. When PayloadBuffer is NULL | Call ReceiveData () with NULL PayloadBuffer& TCG command 0 to return security protocol info  The return status should be EFI\_INVALID\_PARAMETER |
| 5.7.10.2.3 | 0x68acfb97, 0xcec1, 0x4015, 0xac, 0xca, 0x64, 0xad, 0x8c, 0xde, 0x5e, 0x71 | EFI\_STORAGE\_SECURITY\_ COMMAND.SendData – SendData() should return EFI\_NO\_MEDIA. When There is no media present | Call SendData () with TCG command 0 to return security protocol info on a no media device  The return status should be EFI\_NO\_MEDIA. |

## EFI\_DISK\_IO2\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_DISK\_IO2\_PROTOCOL Section.

### Cancel()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.11.1.1 | 0xd8cc30e4, 0xaac4, 0x415b, 0xb0, 0x12, 0x27, 0x13, 0x29, 0x8d, 0x0f, 0xfa | EFI\_DISK\_IO2\_PROTOCOL.Cancel - Cancel () returns EFI\_SUCCESS. | Call Cancel (),the return status should be EFI\_SUCCESS. |

### ReadDiskEx()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.11.2.1 | 0x9b457a7a, 0x9f63, 0x4627, 0x80, 0x6a, 0xfe, 0x39, 0x30, 0x9e, 0x29, 0xec | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_SUCCESS when Async call with proper parameter. | Async Call ReadDiskEx()with proper parameter, the return status should be  EFI\_SUCCESS. If possible, the data should be same as the result from ReadDisk(). |
| 5.7.11.2.2 | 0xac2b9d8c, 0xc35c, 0x4788, 0xa9, 0xcf, 0xb2, 0x93, 0x2e, 0x7c, 0xe2, 0x85 | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_SUCCESS when Async call with proper parameter. | Async Call ReadDiskEx()with proper parameter, the return status should be  EFI\_SUCCESS. The ReadFaillist should be empty. |
| 5.7.11.2.3 | 0xe3aa41fc, 0x1275, 0x4d74, 0x80, 0x97, 0x85, 0x27, 0x5c, 0xc6, 0x22, 0x5c | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_SUCCESS when Async call with proper parameter. | Async Call ReadDiskEx()with proper parameter, the return status should be  EFI\_SUCCESS. The ReadExecuteList should be empty. |
| 5.7.11.2.4 | 0x979b7b0d, 0x22bb, 0x4507, 0x9d, 0x69, 0x21, 0xd1, 0xb8, 0x50, 0x6c, 0x9e | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_SUCCESS when Sync call with proper parameter. | Sync Call ReadDiskEx()with proper parameter, the return status should be  EFI\_SUCCESS.  If possible, the data should be same as the result from ReadDisk(). |
| 5.7.11.2.5 | 0xe09f04a1, 0x00ee,  0x4a48, 0x90, 0x5f, 0x2d, 0x23, 0xd2, 0xa6, 0x71, 0x2e | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_SUCCESS when Batch Async call with proper parameter. | Batch Async Call ReadDiskEx()with proper parameter, the return status should be EFI\_SUCCESS. If possible, the data should be same as the result from ReadDisk(). |
| 5.7.11.2.6 | 0xe6172d46, 0x3648, 0x4677, 0x8d, 0xde, 0xb1, 0xfd, 0x10, 0x6a, 0xe5, 0xe6 | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_SUCCESS when Mixed Sync & Async call with proper parameter. | Mixed Sync & Async Call ReadDiskEx() with proper parameter, the return status should be EFI\_SUCCESS |
| 5.7.11.2.7 | 0x8048d7d8, 0x3e99, 0x4a32, 0x87, 0xac, 0x1f, 0x61, 0x3c, 0xf9, 0x13, 0x9c | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_SUCCESS when Mixed Sync & Async call with proper parameter. Async call failed. | Mixed Sync & Async Call ReadDiskEx()with proper parameter, the return status should be EFI\_SUCCESS. The MixReadFaillist should be empty. |
| 5.7.11.2.8 | 0x6e6179d0, 0xfbe3, 0x4ef8, 0xb5, 0xed, 0x1b, 0xb1, 0xc0, 0xe7, 0x69, 0xb0 | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_SUCCESS when Mixed Sync & Async call with proper parameter.  Async Read Call failed | Mixed Sync & Async Call ReadDiskEx()with proper parameter, the return status should be EFI\_SUCCESS. The MixReadExecutelist should be empty. |
| 5.7.11.2.9 | 0x870cf02a, 0xb573, 0x40d6,0x91, 0x70, 0x8d, 0x83, 0x3d, 0x45, 0xf4, 0xc3 | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_SUCCESS when Mixed Sync & Async call with proper parameter.  Sync Read Call success. | Mixed Sync & Async Call ReadDiskEx()with proper parameter, the return status should be EFI\_SUCCESS. The data in SyncReadList should be same as the output from Async read. |
| 5.7.11.2.10 | 0xb491381b, 0xf841, 0x44fb, 0x94, 0x62, 0xd5, 0x5c, 0x30, 0xde, 0xb1, 0x85 | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_SUCCESS when Mixed Sync & Async call with proper parameter.  Sync Read Call failed. | Mixed Sync & Async Call ReadDiskEx()with proper parameter, the return status should be EFI\_SUCCESS. The SyncReadFailList should be empty. |
| 5.7.11.2.11 | 0xfb7b94af, 0x0368, 0x4a66, 0xaf, 0x52, 0x31, 0x00, 0x8a, 0xf1, 0xd5, 0xc7 | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_MEDIA\_CHANGED when Sync & Async Read disk with MediaId not being the id for the current media in the device. | Sync & Async call ReadDiskEx() with MediaId not being the id for the current media in the device, the return status should be EFI\_MEDIA\_CHANGED. |
| 5.7.11.2.12 | 0x2e9486a6, 0x51c3, 0x4da7,0xa4, 0x81, 0xab, 0xae, 0x72, 0x35, 0xe9, 0x43 | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_MEDIA\_CHANGED when Sync & Async Read disk with invalid offset. | Sync & Async call ReadDiskEx() with invalid offset, the return status should be EFI\_INVALID\_PARAMETERS. |
| 5.7.11.2.13 | 0x8851b5ee, 0x51ea, 0x4241,0xb8, 0x52, 0x40, 0xbc, 0x49, 0x1c, 0x62, 0x31 | EFI\_DISK\_IO2\_PROTOCOL.ReadDiskEx - ReadDiskEx () returns EFI\_MEDIA\_CHANGED when Sync & Async Read disk from device without media present in the device. | Sync & Async call ReadDiskEx()from device without media present in the device, the return status should be EFI\_NO\_MEDIA. |

### WriteDiskEx()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.11.3.1 | 0x3a74e001, 0x817a, 0x45b2, 0xb3, 0x12, 0x3d, 0x12, 0xbb, 0x36, 0x41, 0xc0 | EFI\_DISK\_IO2\_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI\_SUCCESS when Async call with proper parameter. | Async Call WriteDiskEx () with proper parameter, the return status should be  EFI\_SUCCESS. If possible, the date read from the same address should be same as the date written in. |
| 5.7.11.3.2 | 0xeeb0a39d, 0x6c51, 0x4152, 0xb5, 0x74, 0xa6, 0xec, 0xda, 0x4c, 0xdf, 0x80 | EFI\_DISK\_IO2\_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI\_SUCCESS when Async call with proper parameter.  Async call failed. | Async Call WriteDiskEx () with proper parameter, the return status should be  EFI\_SUCCESS. The WriteFailList should be empty. |
| 5.7.11.3.3 | 0x70b3b8f6, 0x91cf, 0x47a5, 0xbc, 0x12, 0x09, 0xe7, 0xb8, 0x27, 0x5d, 0x41 | EFI\_DISK\_IO2\_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI\_SUCCESS when Async call with proper parameter. | Async Call WriteDiskEx ()with proper parameter, the return status should be EFI\_SUCCESS. The WriteExecuteList should be empty. |
| 5.7.11.3.4 | 0x5107009f, 0xe732, 0x45ad, 0xbe, 0x8d, 0xe6, 0x79, 0xb8, 0x76, 0x6a, 0xf3 | EFI\_DISK\_IO2\_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI\_SUCCESS when Sync call with proper parameter. | Sync Call WriteDiskEx ()with proper parameter, the return status should be  EFI\_SUCCESS. If possible, the date read from the same address should be same as the date written in. |
| 5.7.11.3.5 | 0x72023591, 0x1ad7, 0x468c, 0xb4, 0x75, 0x31, 0xa4, 0x1b, 0x9d, 0x0c, 0x78 | EFI\_DISK\_IO2\_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI\_SUCCESS when Batch Async call with proper parameter. | Batch Async Call WriteDiskEx ()with proper parameter, the return status should be  EFI\_SUCCESS. |
| 5.7.11.3.6 | 0x75a4a0e7, 0x5d73, 0x4809, 0xa4, 0x0e, 0x20, 0x3a, 0x0f, 0xcf, 0x09, 0x94 | EFI\_DISK\_IO2\_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI\_MEDIA\_CHANGED when Sync & Async Write disk with MediaId not being the id for the current media in the device. | Sync & Async Call WriteDiskEx () with MediaId not being the id for the current media in the device, the return status should be  EFI\_MEDIA\_CHANGED. |
| 5.7.11.3.7 | 0xe0540275, 0x032e, 0x4507, 0xb3, 0x03, 0x01, 0xc5, 0xbf, 0x9c, 0xe1, 0x56 | EFI\_DISK\_IO2\_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI\_INVALID\_PARAMETERS when Sync & Async Write disk with invalid Offset & BufferSize. | Sync & Async Call WriteDiskEx () with invalid Offset & BufferSize, the return status should be  EFI\_INVALID\_PARAMETERS. |
| 5.7.11.3.8 | 0x8688e7ad, 0x4f3e, 0x432e, 0xaf, 0x3b, 0x03, 0x93, 0x6b, 0xe3, 0xe5, 0xa6 | EFI\_DISK\_IO2\_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI\_WRITE\_PROTECTED when Sync & Async Write disk to a write-protected device. | Sync & Async Call WriteDiskEx () to a write-protected device, the return status should be EFI\_WRITE\_PROTECTED. |
| 5.7.11.3.9 | 0xee9fa363, 0x2009, 0x429f, 0x92, 0x0b, 0x60, 0x3b, 0xc4, 0xdc, 0x6e, 0x64 | EFI\_DISK\_IO2\_PROTOCOL.WriteDiskEx - WriteDiskEx () returns EFI\_NO\_MEDIA when Sync & Async Write disk without media present in the device. | Sync & Async Call WriteDiskEx () without media present in the device, the return status should be EFI\_NO\_MEDIA. |

### FlushDiskEx()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.11.4.1 | 0x681169b1, 0xb5eb, 0x4cb0, 0x91, 0xc6, 0xfd, 0x2d, 0x9f, 0xe8, 0x24, 0x50 | EFI\_DISK\_IO2\_PROTOCOL.FlushDiskEx - FlushDiskEx () returns EFI\_SUCCESS when Async call with proper parameter. | Async Call FlushDiskEx ()with proper parameter, the return status should be  EFI\_SUCCESS. |
| 5.7.11.4.2 | 0x2cf71e16, 0xa399, 0x4a8c, 0xa2, 0xf8, 0x09, 0x5c, 0x9d, 0xcd, 0x25, 0xbd | EFI\_DISK\_IO2\_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI\_SUCCESS when Async call with proper parameter.  Async call failed. | Async Call FlushDiskEx ()with proper parameter, the return status should be  EFI\_SUCCESS. The FlushFailList should be empty. |
| 5.7.11.4.3 | 0x48a3fb9b, 0xd65f, 0x44fe, 0x94, 0x29, 0x14, 0xa6, 0x7b, 0x94, 0x0d, 0xda | EFI\_DISK\_IO2\_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI\_SUCCESS when Async call with proper parameter. | Async Call FlushDiskEx ()with proper parameter, the return status should be EFI\_SUCCESS. The FlushExecuteList should be empty. |
| 5.7.11.4.4 | 0x0003470c, 0x15a7, 0x468a, 0xa2, 0xb1, 0xd1, 0x03, 0x8c, 0x81, 0x70, 0xb5 | EFI\_DISK\_IO2\_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI\_SUCCESS when Sync call with proper parameter. | Sync Call FlushDiskEx ()with proper parameter, the return status should be  EFI\_SUCCESS. |
| 5.7.11.4.5 | 0x14525c4c, 0x213e, 0x4985, 0xa6, 0x42, 0x75, 0x6f, 0x0a, 0x8b, 0x2e, 0xf1 | EFI\_DISK\_IO2\_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI\_SUCCESS when Batch Async call with proper parameter. | Batch Async call FlushDiskEx ()with proper parameter, the return status should be EFI\_SUCCESS. |
| 5.7.11.4.6 | 0x2f6c3f4b, 0x5e09, 0x4ada, 0xb0, 0xea, 0xb2, 0x99, 0xe1, 0xf3, 0xd3, 0x50 | EFI\_DISK\_IO2\_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI\_MEDIA\_CHANGED when Sync & Async flush disk with MediaId not being the id for the current media in the device. | Sync & Async call FlushDiskEx () with MediaId not being the id for the current media in the device, the return status should be  EFI\_MEDIA\_CHANGED. |
| 5.7.11.4.7 | 0x5243f002, 0x6d2e, 0x4267, 0xa5, 0x7b, 0x1f, 0xff, 0xb0, 0x98, 0x8c, 0x5f | EFI\_DISK\_IO2\_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI\_INVALID\_PARAMETERS when Sync & Async flush disk with invalid Offset. | Sync & Async call FlushDiskEx () with invalid Offset, the return status should be  EFI\_INVALID\_PARAMETERS. |
| 5.7.11.4.8 | 0x0c0c5c6d, 0xd082, 0x4b2b, 0x9e, 0x6b, 0x8f, 0xaa, 0x5d, 0x72, 0xe8, 0xd4 | EFI\_DISK\_IO2\_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI\_WRITE\_PROTECTED when Sync & Async flush disk to a write-protected device. | Sync & Async call FlushDiskEx () to a write-protected device, the return status should be EFI\_WRITE\_PROTECTED. |
| 5.7.11.4.9 | 0x28882b47, 0x5bb8, 0x4d8c, 0x84, 0x5c, 0x33, 0xf7, 0x66, 0x32, 0x44, 0x25 | EFI\_DISK\_IO2\_PROTOCOL. FlushDiskEx - FlushDiskEx () returns EFI\_NO\_MEDIA when Sync & Async flush disk without media present in the device. | Sync & Async call FlushDiskEx () without media present in the device, the return status should be EFI\_NO\_MEDIA. |

### EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL Section.

### PassThru()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.12.1.1 | 0x85ee4a17, 0xd2a1, 0x4857, 0x9d, 0xa1, 0xc, 0xa8, 0x2d, 0x45, 0x70, 0x19 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. PassThru() -PassThru()returns EFI\_INVALID\_PARAMETER when TransferBuffer does not meet the alignment requirement specified by the IoAlign field of the EFI\_NVM\_EXPRESS\_PASS\_THRU\_MODE. | 1. Call PassThru() when TransferBuffer does not meet the alignment requirement specified by the IoAlign field of the EFI\_NVM\_EXPRESS\_PASS\_THRU\_MODE, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.1.2 | 0xd6366b2c, 0x437c, 0x48c5, 0x9b, 0xcd, 0x9f, 0x17, 0x6d, 0xf8, 0x61, 0x93 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. PassThru() -PassThru()returns EFI\_INVALID\_PARAMETER when QueueType is not 0 (Admin Submission Queue) or 1 (I/O Submission Queue). | 1. Call PassThru() when QueueType is not 0 (Admin Submission Queue) or 1 (I/O Submission Queue), the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.1.3 | 0xeed32c13, 0x9232, 0x48aa, 0xb0, 0x44, 0xc9, 0xdc, 0x18, 0x47, 0x77, 0xc0 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. Mode –Modecheckreturns Failurewith neither EFI\_NVM\_EXPRESS\_PASS\_THRU\_ATTRIBUTES\_LOGICAL nor EFI\_NVM\_EXPRESS\_PASS\_THRU\_ATTRIBUTES\_PHYSICAL set is an illegal configuration. | 1. An EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL with neither EFI\_NVM\_EXPRESS\_PASS\_THRU\_ATTRIBUTES\_LOGICAL nor EFI\_NVM\_EXPRESS\_PASS\_THRU\_ATTRIBUTES\_PHYSICAL set in Mode.Attributes is an illegal configuration. |
| 5.7.12.1.4 | 0xe22b3a66, 0xb9c8, 0x479a, 0x9c, 0x80, 0x9, 0xa4, 0x44, 0x9c, 0xaf, 0x2e | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. Mode – Modecheckreturns Failure When Mode.IoAlign is neither the power of 2 nor 0. | 1. Mode.IoAlign is neither the power of 2 nor 0. |
| 5.7.12.1.5 | 0x976d1926, 0x862, 0x4f41, 0x84, 0x42, 0xa5, 0x23, 0xf3, 0xc7, 0x9e, 0x4b | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. PassThru() -PassThru()returns EFI\_SUCCESS with the valid Identify Command and NULL Event. | 1. Call PassThru() with the valid Identify Command and NULL Event, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.1.6 | 0x9c88d95c, 0x228a, 0x48e0, 0xbd, 0x17, 0xd1, 0x87, 0x31, 0x9, 0xf1, 0xfc | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. PassThru() -PassThru()returns EFI\_SUCCESS with the valid Identify Command and Event. | 1. Call PassThru() with the valid Identify Command and Event, the return status should be EFI\_INVALID\_PARAMETER and the corresponding notification function should be signaled if the NON\_BLOCKIO is supported. |
|  |  |  |  |

### GetNextNamespace()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.12.2.1 | 0xd516e8e4, 0x2d06, 0x40b4, 0xb5, 0x36, 0x65, 0xf0, 0x1c, 0x59, 0x28, 0xf9 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL.GetNextNamespace() - GetNextNamespace()returns EFI\_INVALID\_PARAMETER with invalid NameSpaceId. | 1. Call GetNextNamespace() with invalid NameSpaceId, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.2.2 | 0x6f1c4115, 0x1ef7, 0x4ae9, 0x8e, 0x9, 0x85, 0xce, 0xe5, 0x4a, 0xd9, 0xb6 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNextNamespace() - GetNextNamespace()returns EFI\_NOT\_FOUND when no more namespaces are defined on this controller. | 1. Call GetNextNamespace()when no more namespaces are defined on this controller, the return status should be EFI\_NOT\_FOUND. |
|  |  |  |  |

### BuildDevicePath()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.12.3.1 | 0x6f45fc1, 0xa9cd, 0x4889, 0x88, 0x1d, 0x5e, 0x34, 0xb8, 0x12, 0xfa, 0x3d | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. BuildDevicePath() - BuildDevicePath()returns EFI\_INVALID\_PARAMETER with NULL DevicePath. | 1. Call BuildDevicePath() with NULL DevicePath, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.3.2 | 0x2b9446e8, 0xea00, 0x49ee, 0x97, 0x2d, 0xcf, 0x2a, 0xa4, 0x9e, 0xa, 0xd3 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. BuildDevicePath() - BuildDevicePath()returns EFI\_NOT\_FOUND with invalid NameSpaceId. | 1. Call BuildDevicePath() with invalid NameSpaceId, the return status should be EFI\_NOT\_FOUND. |
| 5.7.12.3.3 | 0xa11dede9, 0xe13d, 0x4096, 0x90, 0xc8, 0xa6, 0x2e, 0x16, 0xc5, 0x76, 0xaf | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. BuildDevicePath() - BuildDevicePath()returns EFI\_SUCCESS with valid NameSpaceId. | 1. Call GetNextNamespace() with valid NameSpaceId, the return status should be EFI\_SUCCESS. The member NameSpaceId in the DevicePath should be same as the NameSpaceId. |
|  |  |  |  |

### GetNamespace()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.12.4.1 | 0xbefdcd7a, 0xf32d, 0x4423, 0x87, 0x7e, 0xf8, 0xc4, 0x56, 0x38, 0xd6, 0xd8 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNamespace() - GetNamespace()returns EFI\_INVALID\_PARAMETER with NULL NamespaceId. | 1. Call GetNamespace() with NULL NamespaceId, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.4.2 | 0x38ae6f88, 0x2cf9, 0x497b, 0x94, 0x59, 0x7c, 0xaa, 0x34, 0xb7, 0xed, 0x7f | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNamespace() - GetNamespace()returns EFI\_INVALID\_PARAMETER with NULL DevicePath. | 1. Call GetNamespace() with NULL DevicePath, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.4.3 | 0x365f8fba, 0x3314, 0x4502, 0x89, 0x3e, 0x8e, 0x63, 0xc1, 0xda, 0xfe, 0xbc | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNamespace() -GetNamespace()returns EFI\_UNSUPPORTED with unsupported device path node. | 1. Call GetNamespace() with unsupported device path node, the return status should be EFI\_UNSUPPORTED. |
| 5.7.12.4.4 | 0xe864012d, 0x12b0, 0x4467, 0xa9, 0x7b, 0x5f, 0x72, 0xb4, 0xa9, 0x50, 0x27 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNamespace() - GetNamespace()returns EFI\_NOT\_FOUND with NVME device path node, but the translation from DevicePath to namespace ID failed. | 1. Call GetNamespace() with NVME device path node, but translation from DevicePath to namespace ID failed, the return status should be EFI\_NOT\_FOUND. |
| 5.7.12.4.5 | 0xc72a5f58, 0x742a, 0x4c7f, 0xbc, 0xc1, 0x35, 0xf9, 0xd0, 0x31, 0x32, 0xd | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNamespace() - GetNamespace()returns EFI\_SUCCESS with valid parameters. | 1. Call GetNamespace() with valid parameters, the return status should be EFI\_SUCCESS. |
|  |  |  |  |

## EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL Section.

### PassThru()

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **GUID** | **Assertion** | **Test Description** |
| 5.7.12.1.1 | 0x85ee4a17, 0xd2a1, 0x4857, 0x9d, 0xa1, 0xc, 0xa8, 0x2d, 0x45, 0x70, 0x19 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. PassThru() -PassThru()returns EFI\_INVALID\_PARAMETER when TransferBuffer does not meet the alignment requirement specified by the IoAlign field of the EFI\_NVM\_EXPRESS\_PASS\_THRU\_MODE. | 1. Call PassThru() when TransferBuffer does not meet the alignment requirement specified by the IoAlign field of the EFI\_NVM\_EXPRESS\_PASS\_THRU\_MODE, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.1.2 | 0xd6366b2c, 0x437c, 0x48c5, 0x9b, 0xcd, 0x9f, 0x17, 0x6d, 0xf8, 0x61, 0x93 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. PassThru() -PassThru()returns EFI\_INVALID\_PARAMETER when QueueType is not 0 (Admin Submission Queue) or 1 (I/O Submission Queue). | 1. Call PassThru() when QueueType is not 0 (Admin Submission Queue) or 1 (I/O Submission Queue), the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.1.3 | 0xeed32c13, 0x9232, 0x48aa, 0xb0, 0x44, 0xc9, 0xdc, 0x18, 0x47, 0x77, 0xc0 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. Mode –Modecheckreturns Failurewith neither EFI\_NVM\_EXPRESS\_PASS\_THRU\_ATTRIBUTES\_LOGICAL nor EFI\_NVM\_EXPRESS\_PASS\_THRU\_ATTRIBUTES\_PHYSICAL set is an illegal configuration. | 1. An EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL with neither EFI\_NVM\_EXPRESS\_PASS\_THRU\_ATTRIBUTES\_LOGICAL nor EFI\_NVM\_EXPRESS\_PASS\_THRU\_ATTRIBUTES\_PHYSICAL set in Mode.Attributes is an illegal configuration. |
| 5.7.12.1.4 | 0xe22b3a66, 0xb9c8, 0x479a, 0x9c, 0x80, 0x9, 0xa4, 0x44, 0x9c, 0xaf, 0x2e | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. Mode – Modecheckreturns Failure When Mode.IoAlign is neither the power of 2 nor 0. | 1. Mode.IoAlign is neither the power of 2 nor 0. |
| 5.7.12.1.5 | 0x976d1926, 0x862, 0x4f41, 0x84, 0x42, 0xa5, 0x23, 0xf3, 0xc7, 0x9e, 0x4b | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. PassThru() -PassThru()returns EFI\_SUCCESS with the valid Identify Command and NULL Event. | 1. Call PassThru() with the valid Identify Command and NULL Event, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.1.6 | 0x9c88d95c, 0x228a, 0x48e0, 0xbd, 0x17, 0xd1, 0x87, 0x31, 0x9, 0xf1, 0xfc | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. PassThru() -PassThru()returns EFI\_SUCCESS with the valid Identify Command and Event. | 1. Call PassThru() with the valid Identify Command and Event, the return status should be EFI\_INVALID\_PARAMETER and the corresponding notification function should be signaled if the NON\_BLOCKIO is supported. |

### GetNextNamespace()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.12.2.1 | 0xd516e8e4, 0x2d06, 0x40b4, 0xb5, 0x36, 0x65, 0xf0, 0x1c, 0x59, 0x28, 0xf9 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL.GetNextNamespace() - GetNextNamespace()returns EFI\_INVALID\_PARAMETER with invalid NameSpaceId. | 1. Call GetNextNamespace() with invalid NameSpaceId, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.2.2 | 0x6f1c4115, 0x1ef7, 0x4ae9, 0x8e, 0x9, 0x85, 0xce, 0xe5, 0x4a, 0xd9, 0xb6 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNextNamespace() - GetNextNamespace()returns EFI\_NOT\_FOUND when no more namespaces are defined on this controller. | 1. Call GetNextNamespace()when no more namespaces are defined on this controller, the return status should be EFI\_NOT\_FOUND. |
|  |  |  |  |

### BuildDevicePath()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.12.3.1 | 0x6f45fc1, 0xa9cd, 0x4889, 0x88, 0x1d, 0x5e, 0x34, 0xb8, 0x12, 0xfa, 0x3d | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. BuildDevicePath() - BuildDevicePath()returns EFI\_INVALID\_PARAMETER with NULL DevicePath. | 1. Call BuildDevicePath() with NULL DevicePath, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.3.2 | 0x2b9446e8, 0xea00, 0x49ee, 0x97, 0x2d, 0xcf, 0x2a, 0xa4, 0x9e, 0xa, 0xd3 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. BuildDevicePath() - BuildDevicePath()returns EFI\_NOT\_FOUND with invalid NameSpaceId. | 1. Call BuildDevicePath() with invalid NameSpaceId, the return status should be EFI\_NOT\_FOUND. |
| 5.7.12.3.3 | 0xa11dede9, 0xe13d, 0x4096, 0x90, 0xc8, 0xa6, 0x2e, 0x16, 0xc5, 0x76, 0xaf | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. BuildDevicePath() - BuildDevicePath()returns EFI\_SUCCESS with valid NameSpaceId. | 1. Call GetNextNamespace() with valid NameSpaceId, the return status should be EFI\_SUCCESS. The member NameSpaceId in the DevicePath should be same as the NameSpaceId. |
|  |  |  |  |

### GetNamespace()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.12.4.1 | 0xbefdcd7a, 0xf32d, 0x4423, 0x87, 0x7e, 0xf8, 0xc4, 0x56, 0x38, 0xd6, 0xd8 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNamespace() - GetNamespace()returns EFI\_INVALID\_PARAMETER with NULL NamespaceId. | 1. Call GetNamespace() with NULL NamespaceId, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.4.2 | 0x38ae6f88, 0x2cf9, 0x497b, 0x94, 0x59, 0x7c, 0xaa, 0x34, 0xb7, 0xed, 0x7f | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNamespace() - GetNamespace()returns EFI\_INVALID\_PARAMETER with NULL DevicePath. | 1. Call GetNamespace() with NULL DevicePath, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.12.4.3 | 0x365f8fba, 0x3314, 0x4502, 0x89, 0x3e, 0x8e, 0x63, 0xc1, 0xda, 0xfe, 0xbc | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNamespace() -GetNamespace()returns EFI\_UNSUPPORTED with unsupported device path node. | 1. Call GetNamespace() with unsupported device path node, the return status should be EFI\_UNSUPPORTED. |
| 5.7.12.4.4 | 0xe864012d, 0x12b0, 0x4467, 0xa9, 0x7b, 0x5f, 0x72, 0xb4, 0xa9, 0x50, 0x27 | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNamespace() - GetNamespace()returns EFI\_NOT\_FOUND with NVME device path node, but the translation from DevicePath to namespace ID failed. | 1. Call GetNamespace() with NVME device path node, but translation from DevicePath to namespace ID failed, the return status should be EFI\_NOT\_FOUND. |
| 5.7.12.4.5 | 0xc72a5f58, 0x742a, 0x4c7f, 0xbc, 0xc1, 0x35, 0xf9, 0xd0, 0x31, 0x32, 0xd | EFI\_NVM\_EXPRESS\_PASS\_THRU\_PROTOCOL. GetNamespace() - GetNamespace()returns EFI\_SUCCESS with valid parameters. | 1. Call GetNamespace() with valid parameters, the return status should be EFI\_SUCCESS. |
|  |  |  |  |

## EFI\_ERASE\_BLOCK\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_ERASE\_BLOCK\_PROTOCOL Section.

### EraseBlocks()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.13.1.1 | 0xf62e99e3, 0xcda2, 0x4e44, 0x89, 0xa2, 0x47, 0x3b, 0xd8, 0x61, 0x90, 0xf8 | FI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks(). This optional protocol should be installed on the same handle as the EFI\_BLOCK\_IO\_PROTOCOL or EFI\_BLOCK\_IO2\_PROTOCOL. | 1. EFI\_ERASE\_BLOCK\_PROTOCOL should be installed on the same handle as the EFI\_BLOCK\_IO\_PROTOCOL or EFI\_BLOCK\_IO2\_PROTOCOL. |
| 5.7.13.1.2 | 0x4cfed8bb, 0xb9b1, 0x4c21, 0xb3, 0xb6, 0xa7, 0x5, 0x38, 0x6c, 0xf1, 0xe5 | EFI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI\_NO\_MEDIA when there is no media in the device. | 1. Call EraseBlocks() when there is no media in the device, the return status should be EFI\_NO\_MEDIA. |
| 5.7.13.1.3 | 0x9877f323, 0x8812, 0x40bc, 0xbd, 0x41, 0x71, 0xe, 0x8b, 0xbe, 0xb6, 0x69 | EFI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI\_NO\_MEDIA when there is no media in the device, even if LBA is invalid. | 1. Call EraseBlocks() when there is no media in the device, even if LBA is invalid, the return status should be EFI\_NO\_MEDIA. |
| 5.7.13.1.4 | 0x9877cf0d, 0x3d1b, 0x4ac5, 0x8a, 0x3f, 0x8c, 0xba, 0x95, 0x62, 0xb7, 0x53 | EFI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI\_NO\_MEDIA when there is no media in the device, even if Size is invalid. | 1. Call EraseBlocks() when there is no media in the device, even if Size is invalid, the return status should be EFI\_NO\_MEDIA. |
| 5.7.13.1.5 | 0x61c0575e, 0x742f, 0x4094, 0xa8, 0x73, 0x2, 0x11, 0x4, 0xdb, 0x45, 0x1d | EFI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI\_WRITE\_PROTECTED when there is media in the device, but with the read only attribute. | 1. Call EraseBlocks() when there is media in the device, but with the read only attribute, the return status should be EFI\_WRITE\_PROTECTED. |
| 5.7.13.1.6 | 0x2176fd0d, 0xb211, 0x426d, 0xbf, 0xc, 0x84, 0x65, 0x5f, 0x3e, 0x3c, 0xcd | EFI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI\_MEDIA\_CHANGED when the MediaId is not for the current media. | 1. Call EraseBlocks() when the MediaId is not for the current media, the return status should be EFI\_MEDIA\_CHANGED. |
| 5.7.13.1.7 | 0x5d60ba1c, 0x42da, 0x4a50, 0x82, 0xbc, 0xe5, 0xbe, 0xe2, 0x3f, 0x41, 0x4f | EFI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI\_NO\_MEDIA when the MediaId is not for the current media, even if LBA is invalid. | 1. Call EraseBlocks() when the MediaId is not for the current media, even if LBA is invalid, the return status should be EFI\_NO\_MEDIA. |
| 5.7.13.1.8 | 0x702c5141, 0xc1a8, 0x42ee, 0x8f, 0x9c, 0xe6, 0x8, 0x8e, 0x33, 0x2a, 0xe6 | EFI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI\_NO\_MEDIA when the MediaId is not for the current media, even if Size is invalid. | 1. Call EraseBlocks() when the MediaId is not for the current media, even if Size is invalid, the return status should be EFI\_NO\_MEDIA. |
| 5.7.13.1.9 | 0x2864536a, 0x9aa4, 0x44ac, 0xa9, 0x60, 0x3b, 0x6e, 0x4e, 0x93, 0x47, 0xb5 | EFI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI\_INVALID\_PARAMETER when the LBA is invalid. | 1. Call EraseBlocks() when the LBA is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.13.1.10 | 0xb9ec66f1, 0x41ae, 0x44dc, 0xa6, 0xcc, 0x55, 0xde, 0x3b, 0x0, 0x37, 0xca | 5.7.3.14.4 | 0xafd40ec9, 0x5027, 0x42a8, 0xb0, 0x2c, 0x0c, 0xb5, 0x80, 0x86, 0xd7, 0x9c  2. Call EraseBlocks() to erase the same area,the return status should be EFI\_SUCCESS.  3. CallBlockIo->ReadBlocks() to read the same area, the content should be zero.  4. Call BlockIo->WriteBlocks() to restore the original data back. |
| 5.7.13.1.11 | 0x2af1346c, 0xf3d8, 0x48d9, 0x94, 0x61, 0x6e, 0xef, 0xf6, 0xb2, 0x48, 0x3c | status should be EFI\_SUCCESS. | 1. Call BlockIo2->ReadBlocks() to read the data from the specified area.  2. Call EraseBlocks() to erase the same area,the return status should be EFI\_SUCCESS.  3. CallBlockIo2->ReadBlocks() to read the same area, the content should be zero.  4. Call BlockIo2->WriteBlocks() to restore the original data back. |

## EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL Section.

### PassThru()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.14.1.1 | 0x572e13de, 0xcd2e, 0x43ef, 0xa6, 0x41, 0x37, 0x1, 0x28, 0x18, 0xf8, 0xe4 | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. PassThru() - PassThru() returns EFI\_INVALID\_PARAMETER when Packet is NULL. | 1. Call PassThru() when Packet is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| EFI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI\_NO\_MEDIA when there is no media in the device, even if LBA is invalid. | 1. Call EraseBlocks() when there is no media in the device, even if LBA is invalid, the return status should be EFI\_NO\_MEDIA. | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. PassThru() - PassThru() returns EFI\_INVALID\_PARAMETER when the content of Packet is NULL. | 1. Call PassThru() when the content of Packet is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 0x9877cf0d, 0x3d1b, 0x4ac5, 0x8a, 0x3f, 0x8c, 0xba, 0x95, 0x62, 0xb7, 0x53 | EFI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI\_NO\_MEDIA when there is no media in the device, even if Size is invalid. | 1. Call EraseBlocks() when there is no media in the device, even if Size is invalid, the return status should be EFI\_NO\_MEDIA. | 1. Call PassThru() when Packet defines a data command but both InDataBuffer and OutDataBuffer are NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.14.1.4 | 0x516deffa, 0x25ef, 0x4cb6, 0x95, 0xdf, 0xe0, 0x71, 0x93, 0xf0, 0xc4, 0xb5 | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. PassThru() - PassThru() returns EFI\_INVALID\_PARAMETER when Slot is invalid. | 1. Call PassThru() when Slot is invalid, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.14.1.5 | 0x205e3e70, 0x92b1, 0x4534, 0x80, 0x21, 0xf2, 0x39, 0xcc, 0x21, 0xb5, 0x78 | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. PassThru() - PassThru(). The IoAlign should be 0, 1 or the power of 2. | 1. The IoAlign should be 0, 1 or the power of 2. |
| EFI\_ERASE\_BLOCK\_PROTOCOL. EraseBlocks() - EraseBlocks() returns EFI\_NO\_MEDIA when the MediaId is not for the current media, even if LBA is invalid. | 1. Call EraseBlocks() when the MediaId is not for the current media, even if LBA is invalid, the return status should be EFI\_NO\_MEDIA. | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. PassThru() - PassThru() returns EFI\_SUCCESS when the SD Command Packet was sent by the host. | 1. Call PassThru() when the SD Command Packet was sent by the host, the return status should be EFI\_INVALID\_PARAMETER. |

### GetNextSlot()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.14.2.1 | 0xcd9e89de, 0x9765, 0x4930, 0xa1, 0x88, 0xbc, 0x30, 0xd4, 0x9, 0xa0, 0x92 | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL.  GetNextSlot() - GetNextSlot() returns EFI\_INVALID\_PARAMETER when Slot is not 0xFF and Slot was not returned on a previous call. | 1. Call GetNextSlot() when Slot is not 0xFF and Slot was not returned on a previous call, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.14.2.2 | 0x8f6d644f, 0x2d1e, 0x40b3, 0x91, 0x4a, 0xc6, 0xda, 0x21, 0x3, 0x82, 0x44 | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. GetNextSlot() - GetNextSlot() returns EFI\_NOT\_FOUND when there are no more slots on this SD controller. | 1. Call GetNextSlot() when there are no more slots on this SD controller, the return status should be EFI\_NOT\_FOUND. |

### BuildDevicePath()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.14.3.1 | 0x962accdc, 0x5808, 0x450d, 0xba, 0xea, 0xe3, 0xb7, 0x1a, 0x34, 0x76, 0x22 | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. BuildDevicePath() - BuildDevicePath () returns EFI\_INVALID\_PARAMETER when DevicePath is NULL. | 1. Call BuildDevicePath() when DevicePath is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.14.3.2 | 0x2597450b, 0xab3d, 0x49d6, 0x9c, 0x3f, 0xec, 0xcd, 0x24, 0xcc, 0xb5, 0xf5 | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. BuildDevicePath() - BuildDevicePath () returns EFI\_NOT\_FOUND when the SD card specified by Slot does not exist on the SD controller. | 1. Call BuildDevicePath() when the SD card specified by Slot does not exist on the SD controller, the return status should be EFI\_NOT\_FOUND. |
| 5.7.14.3.3 | 0x871efb1e, 0xdfbe, 0x4a0c, 0x83, 0xc4, 0x21, 0x9c, 0x20, 0x91, 0x8e, 0x91 | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. BuildDevicePath() - BuildDevicePath () returns EFI\_SUCCESS when the device path node that describes the SD card specified by Slot was allocated and returned in DevicePath. | 1. Call BuildDevicePath() when the device path node that describes the SD card specified by Slot was allocated and returned in DevicePath, the return status should be EFI\_SUCCESS. |

### GetSlotNumber()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.14.4.1 | 0xab2880b3, 0x9ac3, 0x4ca4, 0x94, 0x75, 0x4e, 0xbd, 0xd1, 0xbe, 0xa, 0xd8 | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. GetSlotNumber() - GetSlotNumber() returns EFI\_INVALID\_PARAMETER when DevicePath is NULL. | 1. Call GetSlotNumber() when DevicePath is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.14.4.2 | 0xff66737b, 0xad5c, 0x4383, 0xbe, 0x96, 0x9a, 0xff, 0xd7, 0xe2, 0xb3, 0x7a | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. GetSlotNumber() - GetSlotNumber() returns EFI\_INVALID\_PARAMETER when Slot is NULL. | 1. Call GetSlotNumber() when Slot is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.14.4.3 | 0x240951b8, 0xaa03, 0x4517, 0xb0, 0xa7, 0x3a, 0xbc, 0x57, 0x5a, 0xc, 0x3e | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. GetSlotNumber() - GetSlotNumber() returns EFI\_UNSUPPORTED when DevicePath is not a device path node type that the SD PassThru driver supports. | 1. Call GetSlotNumber() when DevicePath is not a device path node type that the SD PassThru driver supports, the return status should be EFI\_UNSUPPORTED. |
| 5.7.14.4.4 | 0xb0631fb9, 0xd1f9, 0x41e6, 0xb1, 0x74, 0x18, 0xea, 0x2, 0x59, 0xd4, 0x7a | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. GetSlotNumber() - GetSlotNumber() returns EFI\_SUCCESS when SD card slot number is returned in Slot. | 1. Call GetSlotNumber() when SD card slot number is returned in Slot, the return status should be EFI\_SUCCESS. |

### ResetDevice()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.14.5.1 | 0x2dbb3a26, 0xb27, 0x4333, 0xa2, 0xec, 0xc3, 0x48, 0xee, 0xf9, 0xc9, 0x3e | EFI\_SD\_MMC\_PASS\_THRU\_PROTOCOL. ResetDevice() - ResetDevice() returns EFI\_INVALID\_PARAMETER when Slot number is invalid or the SD controller does not support a device reset operation. | 1. Call ResetDevice() when Slot number is invalid or the SD controller does not support a device reset operation, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.14.5.2 | 0x70c428ae, 0xf1a6, 0x4d02, 0xa1, 0x26, 0x47, 0x89, 0x14, 0xf5, 0xb5, 0xa2 | EFI\_ERASE\_BLOCK\_PROTOCOL. ResetDevice() - ResetDevice() returns EFI\_SUCCESS when the SD card specified by the Slot is reset. | 1. Call ResetDevice() when the SD card specified by the Slot is reset, the return status should be EFI\_SUCCESS. |

## EFI\_RAM\_DISK\_PROTOCOL Test

Reference Document:

*UEFI Specification*, EFI\_RAM\_DISK\_PROTOCOL Section

### Register()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.15.1.1 | 0xf57e3b87, 0x2b93, 0x4645, 0x86, 0x56, 0x9a, 0x59, 0x53, 0x34, 0x58, 0x4b | EFI\_RAM\_DISK\_PROTOCOL.  Register() -  Register() returns EFI\_INVALID\_PARAMETER when  RamDiskSize is 0. | 1. Call Register() when RamDiskSize is 0, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.15.1.2 | 0x309c7941, 0x13be, 0x43f6, 0x83, 0x33, 0x1c, 0x49, 0x5e, 0x7d, 0xf3, 0x56 | EFI\_RAM\_DISK\_PROTOCOL.Register() - Register() returns EFI\_INVALID\_PARAMETER when RamDiskType is NULL. | 1. Call Register() when RamDiskType is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.15.1.3 | 0x35c6688b, 0x7eb9, 0x4446, 0x94, 0x7f, 0x34, 0x39, 0x16, 0xc5, 0xb9, 0x65 | EFI\_RAM\_DISK\_PROTOCOL.Register() - Register() returns EFI\_INVALID\_PARAMETER when DevicePath is NULL. | 1. Call Register() when DevicePath is NULL, the return status should be EFI\_INVALID\_PARAMETER. |
| 5.7.15.1.4 | 0xbf0432c4, 0x5b9b, 0x42f9, 0x94, 0x62, 0x49, 0x57, 0xb, 0x86, 0x83, 0xe1 | EFI\_RAM\_DISK\_PROTOCOL.Register() - Register() returns EFI\_ALREADY\_STARTED when the created DevicePath instance is already present in the handle database. | 1. Call Register() to register one RAM disk with specified address, size and type.  2. Call Register() with the same parameters again, the return status should be EFI\_ALREADY\_STARTED. |
| 5.7.15.1.5 | 0xb5b749af, 0x5ad3, 0x4e79, 0x88, 0x68, 0x58, 0x64, 0x65, 0x24, 0x91, 0x5f | EFI\_RAM\_DISK\_PROTOCOL.Register() - Register() returns EFI\_SUCCESS with valid parameters. | 1. Call Register() with valid parameters, the return status should be EFI\_SUCCESS. |

### Unregister()

|  |  |  |  |
| --- | --- | --- | --- |
| Number | GUID | Assertion | Test Description |
| 5.7.15.2.1 | 0xf05eae55, 0x1dd7, 0x4a10, 0xba, 0x57, 0x38, 0x8d, 0x38, 0x5, 0x51, 0x10 | EFI\_RAM\_DISK\_PROTOCOL.Unregister() - Unregister() returns EFI\_NOT\_FOUND when DevicePath is not existed. | 1. Call Unregister() when DevicePath is not existed, the return status should be EFI\_NOT\_FOUND. |
| 5.7.15.2.2 | 0x6919f770, 0xf418, 0x4873, 0x81, 0x38, 0xc1, 0x45, 0x36, 0x80, 0x1d, 0x77 | EFI\_RAM\_DISK\_PROTOCOL.Unregister() - Unregister() returns EFI\_INVALID\_PARAMETER when DevicePath is NULL. | 1. Call Unregister() when DevicePath is NULL, the return status should be  EFI\_INVALID\_PARAMETER. |
| 5.7.15.2.3 | 0xbc90d7f7, 0x275d, 0x424f, 0x9c, 0x95, 0x14, 0x6e, 0x24, 0xbd, 0xc3, 0xe6 | EFI\_RAM\_DISK\_PROTOCOL.Unregister() - Unregister() returns EFI\_UNSUPPORTED when DevicePath is not the valid Ramdisk device path. | 1. Call Unregister() when DevicePath is not the valid Ramdisk device path, the return status should be EFI\_UNSUPPORTED. |
| 5.7.15.2.4 | 0xa85e1978, 0x216f, 0x4f52, 0xad, 0x7c, 0x70, 0xc2, 0x65, 0xe6, 0xf7, 0xee | EFI\_RAM\_DISK\_PROTOCOL.Unregister() - Unregister() returns EFI\_SUCCESS with valid parameters. | 1. Call Unregister() with valid parameters, the return status should be EFI\_SUCCESS. |