



# UEFI Shell Specification

*September 25, 2008*

Revision 2.0

The material contained herein is not a license, either expressly or impliedly, to any intellectual property owned or controlled by any of the authors or developers of this material or to any contribution thereto. The material contained herein is provided on an "AS IS" basis and, to the maximum extent permitted by applicable law, this information is provided AS IS AND WITH ALL FAULTS, and the authors and developers of this material hereby disclaim all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of results, of workmanlike effort, of lack of viruses and of lack of negligence, all with regard to this material and any contribution thereto. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." The Unified EFI Forum, Inc. reserves any features or instructions so marked for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THE SPECIFICATION AND ANY CONTRIBUTION THERETO.

IN NO EVENT WILL ANY AUTHOR OR DEVELOPER OF THIS MATERIAL OR ANY CONTRIBUTION THERETO BE LIABLE TO ANY OTHER PARTY FOR THE COST OF PROCURING SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA, OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THIS OR ANY OTHER AGREEMENT RELATING TO THIS DOCUMENT, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Copyright 2008 Unified EFI, Inc. All Rights Reserved

# Contents

---

1	Introduction .....	1
1.1	Overview .....	1
1.2	Related Information .....	1
1.3	Terms .....	1
2	Code Definitions .....	3
2.1	Introduction .....	3
2.2	EFI_SHELL_PROTOCOL .....	3
	EFI_SHELL_PROTOCOL .....	3
	EFI_SHELL_PROTOCOL.BatchIsActive() .....	8
	EFI_SHELL_PROTOCOL.CloseFile() .....	9
	EFI_SHELL_PROTOCOL.CreateFile() .....	10
	EFI_SHELL_PROTOCOL.DeleteFile() .....	12
	EFI_SHELL_PROTOCOL.DeleteFileByName() .....	13
	EFI_SHELL_PROTOCOL.DisablePageBreak() .....	14
	EFI_SHELL_PROTOCOL.EnablePageBreak() .....	15
	EFI_SHELL_PROTOCOL.Execute() .....	16
	EFI_SHELL_PROTOCOL.FindFiles() .....	18
	EFI_SHELL_PROTOCOL.FindFilesInDir() .....	19
	EFI_SHELL_PROTOCOL.FlushFile() .....	20
	EFI_SHELL_PROTOCOL.FreeFileList() .....	21
	EFI_SHELL_PROTOCOL.GetCurDir() .....	22
	EFI_SHELL_PROTOCOL.GetDeviceName() .....	23
	EFI_SHELL_PROTOCOL.GetDevicePathFromMap() .....	25
	EFI_SHELL_PROTOCOL.GetDevicePathFromFilePath() .....	26
	EFI_SHELL_PROTOCOL.GetEnv() .....	27
	EFI_SHELL_PROTOCOL.GetFileInfo() .....	28
	EFI_SHELL_PROTOCOL.GetFilePathFromDevicePath() .....	29
	EFI_SHELL_PROTOCOL.GetFilePosition() .....	30
	EFI_SHELL_PROTOCOL.GetFileSize() .....	31
	EFI_SHELL_PROTOCOL.GetHelpText() .....	32
	EFI_SHELL_PROTOCOL.GetMapFromDevicePath() .....	34
	EFI_SHELL_PROTOCOL.GetPageBreak() .....	35
	EFI_SHELL_PROTOCOL.IsRootShell() .....	36
	EFI_SHELL_PROTOCOL.OpenFileByName() .....	37
	EFI_SHELL_PROTOCOL.OpenFileList() .....	39
	EFI_SHELL_PROTOCOL.OpenRoot() .....	41
	EFI_SHELL_PROTOCOL.OpenRootByHandle() .....	42
	EFI_SHELL_PROTOCOL.ReadFile() .....	43
	EFI_SHELL_PROTOCOL.RemoveDupInFileList() .....	44
	EFI_SHELL_PROTOCOL.SetAlias() .....	45
	EFI_SHELL_PROTOCOL.SetCurDir() .....	46
	EFI_SHELL_PROTOCOL.SetEnv() .....	47
	EFI_SHELL_PROTOCOL.SetFileInfo() .....	48
	EFI_SHELL_PROTOCOL.SetFilePosition() .....	49
	EFI_SHELL_PROTOCOL.SetMap() .....	50
	EFI_SHELL_PROTOCOL.WriteFile() .....	51

2.3	EFI_SHELL_PARAMETERS_PROTOCOL .....	52
	EFI_SHELL_PARAMETERS_PROTOCOL .....	52
3	UEFI Shell Features .....	55
3.1	Levels Of Support .....	55
3.2	Invocation .....	56
3.3	Initialization .....	57
3.3.1	Finding startup.nsh .....	58
3.3.2	Supported Profiles .....	58
3.4	Command-Line .....	58
3.4.1	Special Characters .....	59
3.4.2	Escape Characters .....	59
3.4.3	Quoting .....	59
3.4.4	Redirection .....	60
3.5	Current Directory .....	62
3.6	Variables .....	62
3.6.1	Environment Variables .....	63
3.6.2	Positional Parameters .....	65
3.6.3	Index Parameters .....	65
3.6.4	Aliases .....	65
3.7	File Names .....	66
3.7.1	Wildcard Expansion .....	66
3.7.2	Mappings .....	67
3.7.3	Consistent File System Mapping .....	67
3.8	Scripts .....	67
3.9	Nesting the Shell .....	68
3.10	Interactive Features .....	68
3.10.1	Key History Support .....	68
3.10.2	Execution Interrupt Support .....	68
3.10.3	Output Streaming Control .....	69
3.10.4	Scroll Back Buffer Support .....	69
3.11	Shell Applications .....	69
3.11.1	Installation .....	69
3.11.2	Command-Line Help .....	70
4	Scripts .....	71
4.1	Comments .....	71
4.2	Error Handling .....	71
4.3	Script Nesting .....	72
4.4	Output and Echoing .....	72
4.5	Limitations .....	72
5	Shell Commands .....	73
5.1	Overview .....	73
5.1.1	Explanation of Command Description Layout .....	77
5.1.2	Shell Command-Line Options .....	77
5.2	Shell Command Profiles .....	78
5.3	Shell Commands .....	78
	alias .....	78
	attrib .....	81
	bcfg .....	83
	cd .....	86

cls	88
comp	89
connect	91
cp	93
date	96
dblk	98
del	102
devices	103
devtree	105
dh	106
dir	109
disconnect	110
dmem	112
dmpstore	115
drivers	117
drvcfg	120
drvdiag	124
echo	126
edit	128
eficompress	129
efidecompress	130
exit	131
for	132
getmtc	134
goto	135
guid	136
help	137
hexedit	139
if	140
ifconfig	145
ipconfig	147
load	148
loadpcirom	150
ls	151
map	155
md	158
mem	159
memmap	160
mkdir	163
mm	165
mode	168
mv	170
openinfo	172
parse	173
pause	175
pci	176
ping	180
reconnect	181
reset	183
rm	184
sermode	186
set	188
setsize	190
setvar	191
shift	193

smbiosview .....	194
stall .....	196
time .....	197
timezone .....	199
touch .....	201
type .....	202
unload .....	204
ver .....	205
vol .....	207
Appendix A UEFI Shell Consistent Mapping Design .....	209
A.1 Requirement: .....	209
A.2 Design .....	209
A.2.1 What does consistent mapping mean?.....	209
A.2.2 Hardware configuration change:.....	209
A.2.2.1 Hardware configuration change example .....	210
A.2.2.2 Hardware configuration not change example.....	210
A.2.3 Mapping generated from device path .....	210
A.2.4 Consistent Mapping .....	210
A.2.5 Example (USB Devices) .....	211
A.2.5.1 Step 1: Determine MTD .....	211
A.2.5.2 Step 2: Determine HI .....	212
A.2.5.3 Step 3: Determine CSD .....	213
A.2.5.3.1 Rules for USB device path node .....	213
A.2.5.3.2 Rules for hard drive device path node .....	213
A.2.5.3.3 Corresponding CSDs.....	213
A.2.5.3.4 Step 4: Make the whole mapping .....	213
A.3 Implementation .....	214
A.3.1 Get the MTD.....	216
A.3.2 Get the HI .....	216
A.3.3 Get the CSD.....	216
A.3.3.1 USB Devices .....	216
A.3.3.1.1 Example .....	217
A.3.3.2 Floppy Devices .....	217

A.3.3.2.1	Examples: .....	217
A.3.3.3	CSD of ATAPI device path node.....	217
A.3.3.4	CSD of LUN Device path node.....	217
A.3.3.4.1	Examples: .....	217
A.3.3.5	CSD of SAS Device path node.....	217
A.3.3.6	CSD of SATA Device path node.....	217
A.3.3.7	CSD of hard drive device path node.....	217
A.3.3.7.1	Examples.....	218
A.3.3.8	CSD of SCSI device path node.....	218
A.3.3.8.1	Examples: .....	218
A.3.3.9	Fibre Channel Device Path.....	218
A.3.3.10	1394 Device Path.....	218
A.3.3.11	I2o Device Path.....	218
A.3.3.12	MAC Address Device Path.....	218
A.3.3.13	InfiniBand Device Path.....	218
A.3.3.14	UART Device Path.....	219
A.3.3.15	Vendor-Defined Device Path.....	219
A.3.3.16	Vendor-defined Media Device Path.....	219
A.3.3.17	File Path Media Device Path.....	219
A.4	Function & Structure. ....	219
Appendix B	UEFI Help Manual Page Syntax.....	223
Appendix C	UEFI Shell Status Codes.....	225
Appendix D	UEFI Shell Command Standard Formatted Output.....	227

# Tables

---

Table 1 Support Levels .....	55
Table 2 Standard Command Line Options .....	56
Table 3: UEFI Shell Invocation Options.....	56
Table 4 Output Redirection Syntax .....	60
Table 5 Input Redirection Syntax .....	61
Table 6 Input Redirection Syntax .....	62
Table 7 Environment Variables with Special Meaning to the UEFI Shell.....	64
Table 8 Built-in Aliases for the UEFI Shell .....	66
Table 9 Wildcard Character Expansion.....	67
Table 10 Commands from Default Build Shell.....	74
Table 11 Standard Command Line Options.....	77
Table 12 Standard Profiles.....	78
Table 13 Conventions for Directory Names .....	86
Table 14 Date Command Table .....	97
Table 15 Standard-Format Output for devices .....	103
Table 16 dh Standard Formatted Output (HandlesInfo).....	108
Table 17 Variable command line options.....	115
Table 18 Drivers command table .....	118
Table 19 Comparison Operators .....	142
Table 20 Functions used to convert integers into UEFI, PI or OEM error codes.....	142
Table 21 Boolean Functions .....	143
Table 22 ls Standard Formatted Output (VolumeInfo).....	153
Table 23 ls Standard Formatted Output (FileInfo) .....	153
Table 24 Standard Formatted Output (Mappings).....	157
Table 25 Standard-Format Output for memmap (MemoryMap) .....	161
Table 26 Standard-Format Output for memmap (Summary).....	162
Table 27 Open Protocol Information Layout.....	172
Table 28 How to process each type the device path node: .....	214
Table 29 MTD Naming .....	216
Table 30 Subheadings and descriptions .....	223
Table 31 SHELL_STATUS return codes .....	225



# Revision History

---

Revision Number	Description	Revision Date
2.0	Initial UEFI release	9/25/08

§



# 1 Introduction

---

## 1.1 Overview

The UEFI Shell environment provides an API, a command prompt and a rich set of commands that extend and enhance the UEFI Shell's capability.

## 1.2 Related Information

The following publications and sources of information may be useful or are referred to by this document:

*Extensible Firmware Interface Specification*, Version 1.10, Intel, 2001, <http://developer.intel.com/technology/efi>.

*Unified Extensible Firmware Interface Specification*, Version 2.0, Unified EFI, Inc, 2006, <http://www.uefi.org>.

*Intel® Platform Innovation Framework for EFI Specifications*, Intel, 2006, <http://www.intel.com/technology/framework/>.

## 1.3 Terms

### **EFI**

Generic term that refers to one of the versions of the EFI specification: EFI 1.02, EFI 1.10, or UEFI 2.0.

### **EFI 1.10 Specification**

Intel Corporation published the Extensible Firmware Interface Specification. It has been supplanted by the Unified Extensible Firmware Interface (UEFI), which is controlled by the UEFI Forum.

### **GUID**

Globally Unique Identifier. A 128-bit value used to name entities uniquely. Without the help of a centralized authority, an individual can generate a unique GUID. This allows the generation of names that will never conflict, even among multiple, unrelated parties.

### **Protocol**

An API named by a GUID as defined by the UEFI specification.

**UEFI Application**

An application following the UEFI specification. The only difference between a UEFI application and a UEFI driver is that an application is unloaded from memory when it exits regardless of return status, while a driver that returns a successful return status is not unloaded when its entry point exits.

**UEFI Driver**

A driver following the UEFI specification driver model.

**UEFI Specification Version 2.0**

The first UEFI specification released by the Unified EFI Forum.

**UEFI Specification Version 2.1**

Current version of the UEFI specification released by the Unified EFI Forum.

**Unified EFI Forum**

A non-profit collaborative trade organization formed to promote and manage the UEFI standard. For more information, see [www.uefi.org](http://www.uefi.org).

§

## 2 Code Definitions

---

### 2.1 Introduction

### 2.2 EFI\_SHELL\_PROTOCOL

#### EFI\_SHELL\_PROTOCOL

#### Summary

Provides shell services to UEFI applications.



Formatted: Bullets and Numbering

## GUID

```
#define EFI_SHELL_PROTOCOL_GUID \
{ 0x6302d008, 0x7f9b, 0x4f30, \
  { 0x87, 0xac, 0x60, 0xc9, 0xfe, 0xf5, 0xda, 0x4e } }
```

## Protocol Interface Structure

```
typedef struct _EFI_SHELL_PROTOCOL {
    EFI_SHELL_EXECUTE                Execute;
    EFI_SHELL_GET_ENV                GetEnv;
    EFI_SHELL_SET_ENV                SetEnv;
    EFI_SHELL_SET_ALIAS              SetAlias;
    EFI_SHELL_GET_HELP_TEXT          GetHelpText;
    EFI_SHELL_GET_DEVICE_PATH_FROM_MAP GetDevicePathFromMap;
    EFI_SHELL_GET_MAP_FROM_DEVICE_PATH GetMapFromDevicePath;
    EFI_SHELL_GET_DEVICE_PATH_FROM_FILE_PATH GetDevicePathFromFilePath;
    EFI_SHELL_GET_FILE_PATH_FROM_DEVICE_PATH GetFilePathFromDevicePath;
    EFI_SHELL_SET_MAP                SetMap;

    EFI_SHELL_GET_CUR_DIR            GetCurDir;
    EFI_SHELL_SET_CUR_DIR            SetCurDir;
    EFI_SHELL_OPEN_FILE_LIST         OpenFileList;
    EFI_SHELL_FREE_FILE_LIST         FreeFileList;
    EFI_SHELL_REMOVE_DUP_IN_FILE_LIST RemoveDupInFileList;

    EFI_SHELL_BATCH_IS_ACTIVE        BatchIsActive;
    EFI_SHELL_IS_ROOT_SHELL          IsRootShell;
    EFI_SHELL_ENABLE_PAGE_BREAK      EnablePageBreak;
    EFI_SHELL_DISABLE_PAGE_BREAK     DisablePageBreak;
    EFI_SHELL_GET_PAGE_BREAK         GetPageBreak;
    EFI_SHELL_GET_DEVICE_NAME        GetDeviceName;

    EFI_SHELL_GET_FILE_INFO          GetFileInfo;
    EFI_SHELL_SET_FILE_INFO          SetFileInfo;
    EFI_SHELL_OPEN_FILE_BY_NAME      OpenFileByName;
    EFI_SHELL_CLOSE_FILE             CloseFile;
    EFI_SHELL_CREATE_FILE            CreateFile;
    EFI_SHELL_READ_FILE              ReadFile;
    EFI_SHELL_WRITE_FILE             WriteFile;
    EFI_SHELL_DELETE_FILE            DeleteFile;
    EFI_SHELL_DELETE_FILE_BY_NAME    DeleteFileByName;
    EFI_SHELL_GET_FILE_POSITION      GetFilePosition;
    EFI_SHELL_SET_FILE_POSITION      SetFilePosition;
    EFI_SHELL_FLUSH_FILE             FlushFile;
    EFI_SHELL_FIND_FILES             FindFiles;
    EFI_SHELL_FIND_FILES_IN_DIR      FindFilesInDir;
    EFI_SHELL_GET_FILE_SIZE          GetFileSize;

    EFI_SHELL_OPEN_ROOT              OpenRoot;
    EFI_SHELL_OPEN_ROOT_BY_HANDLE    OpenRootByHandle;

    EFI_EVENT                        ExecutionBreak;

    UINT32                            MajorVersion;
    UINT32                            MinorVersion;
} EFI_SHELL_PROTOCOL;
```

## Members

*Execute*

Causes the shell to parse and execute the command line. See the **Execute()** function description below.

*GetEnv*

Gets the environment variable. See the **GetEnv()** function description below.

*SetEnv*

Changes a specific environment variable. Set the **SetEnv()** function description below.

*SetAlias*

Adds or removes the alias for a specific shell command. See the **SetAlias()** function description below.

*GetDevicePathFromMap*

Returns the device path that corresponds to a mapping. See the **GetDevicePathFromMap()** function description below.

*GetMapFromDevicePath*

Returns the mapping that corresponds to a particular device path. See the **GetMapFromDevicePath()** function description below.

*GetDevicePathFromFilePath*

Converts a file path to a device path, where all mappings have been replaced with the corresponding device paths.

*GetFilePathFromDevicePath*

Converts a device path to a file path, where the portion of the device path corresponding to one of the mappings is replaced with that mapping.

*SetMap*

Creates, updates or deletes a mapping between a device and a device path.

*GetCurDir*

Returns the current directory on a device. See the **GetCurDir()** function description below.

*SetCurDir*

Changes the current directory on a device. Set the **SetCurDir()** function description below.

*OpenFileList*

Opens the files that match the path pattern specified. See the **OpenFileList()** function description below.

*FreeFileList*

Frees the file list that created by **OpenFileList()**. See the **FreeFileList()** function description below.

*RemoveDupInFileList*

Deletes the duplicate files in the given file list. See the **RemoveDupInFileList()** function description below.

*BatchIsActive*

Returns whether any script files are currently being processed. See the **BatchIsActive()** function description below.

#### *IsRootShell*

Judges whether the active Shell is the root shell. See the **IsRootShell()** function description below.

#### *EnablePageBreak*

Enables the page break output mode. See the **EnablePageBreak()** function description below.

#### *DisablePageBreak*

Disables the page break output mode. See the **DisablePageBreak()** function description below.

#### *GetPageBreak*

Gets the enable status of the page break output mode. See the **GetPageBreak()** function description below.

#### *GetDeviceName*

Gets the name of the device specified by the device handle. See the **GetDeviceName()** function description below.

#### *GetFileInfo*

Return information about a specific file handle. See the **GetFileInfo()** function description below.

#### *SetFileInfo*

Change information about a specific file handle. See the **SetFileInfo()** function description below.

#### *OpenFileByName*

Given a file name, open a file and return a file handle. See the **OpenFileByName()** description below.

#### *CloseFile*

Close an open file. See the **CloseFile()** description below.

#### *CreateFile*

Create a new file. See the **CreateFile()** function description.

#### *ReadFile*

Read data from a file. See the **ReadFile()** function description.

#### *WriteFile*

Write data to a file. See the **WriteFile()** function description.

#### *DeleteFile*

Delete a file. See the **DeleteFile()** function description.

#### *DeleteFileByName*

Delete a file by name. See the **DeleteFileByName()** function description.

#### *SetFilePosition*

Change the current read/write position within a file. See the **SetFilePosition()** function description.



#### *GetFilePosition*

Return the current read/write position within a file. See the **GetFilePosition()** function description.

#### *FlushFile*

Write all buffered data to a file. See the **FlushFile()** function description.

#### *FindFiles*

Return all files that match a pattern in a file list. See the **FindFiles()** function description.

#### *FindFilesInDir*

Return all files in a specified directory in a file list. See the **FindFilesInDir()** function description.

#### *GetFileSize*

Return the size of a file. See the **GetFileSize()** function description.

#### *OpenRoot*

Return the root directory of a file system. See the **OpenRoot()** function description.

#### *OpenRootByHandle*

Return the root directory of a file system on a particular handle. See the **OpenRootByHandle()** function description.

#### *ExecutionBreak*

Event signaled by the UEFI Shell when the user presses CTRL-C to indicate that the current UEFI Shell command execution should be interrupted.

#### *MajorVersion*

The major version of the shell environment.

#### *MinorVersion*

The minor version of the shell environment.

## Description

Description that is more detailed goes here.

## EFI\_SHELL\_PROTOCOL.BatchIsActive()

### Summary

Returns whether any script files are currently being processed.

### Prototype

```
typedef
BOOLEAN
(EFI_API *EFI_SHELL_BATCH_IS_ACTIVE) (
    VOID
);
```

### Parameters

None

### Description

This function tells whether any script files are currently being processed

### Status Codes Returned

<b>TRUE</b>	There is at least one script file active.
<b>FALSE</b>	No script files are active now.

## EFI\_SHELL\_PROTOCOL.CloseFile()

### Summary

Closes the file handle.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_CLOSE_FILE) (
    IN EFI_FILE_HANDLE    FileHandle
);
```

### Parameters

*FileHandle*  
The file handle to be closed

### Description

This function closes a specified file handle. All “dirty” cached file data is flushed to the device, and the file is closed. In all cases, the handle is closed.

### Status Codes Returned

<b>EFI_SUCCESS</b>	The file is closed successfully
--------------------	---------------------------------

## EFI\_SHELL\_PROTOCOL.CreateFile()

### Summary

Creates a file or directory by name.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_CREATE_FILE) (
    IN CONST CHAR16 *FileName,
    IN UINT64 FileAttribs,
    OUT EFI_FILE_HANDLE *FileHandle
);
```

### Parameters

*FileName*

Points to the null-terminated file path.

*FileAttribs*

The new file's attributes. The different attributes are described in `EFI_FILE_PROTOCOL.Open()`.

*FileHandle*

On return, points to the created file or directory's handle.

### Description

This function creates an empty new file or directory with the specified attributes and returns the new file's handle. If the file already exists and is read-only, then `EFI_INVALID_PARAMETER` will be returned.

If the file already existed, it is truncated and its attributes updated. If the file is created successfully, the *FileHandle* is the file's handle, else, the *FileHandle* is `NULL`.

If the file name begins with >v, then the file handle which is returned refers to the shell environment variable with the specified name. If the shell environment variable already exists and is non-volatile then `EFI_INVALID_PARAMETER` is returned.

### Status Codes Returned

<code>EFI_SUCCESS</code>	The file was opened. <i>FileHandle</i> points to the new file's handle.
<code>EFI_INVALID_PARAMETER</code>	One of the parameters has an invalid value.
<code>EFI_UNSUPPORTED</code>	Could not open the file path.
<code>EFI_NOT_FOUND</code>	The specified file could not be found on the device, or could not find the file system on the device.
<code>EFI_NO_MEDIA</code>	The device has no medium.

<b>EFI_MEDIA_CHANGED</b>	The device has a different medium in it or the medium is no longer supported.
<b>EFI_DEVICE_ERROR</b>	The device reported an error or can't get the file path according the <i>DirName</i> .
<b>EFI_VOLUME_CORRUPTED</b>	The file system structures are corrupted.
<b>EFI_WRITE_PROTECTED</b>	An attempt was made to create a file, or open a file for write when the media is write-protected.
<b>EFI_ACCESS_DENIED</b>	The service denied access to the file.
<b>EFI_OUT_OF_RESOURCES</b>	Not enough resources were available to open the file.
<b>EFI_VOLUME_FULL</b>	The volume is full.

## EFI\_SHELL\_PROTOCOL.DeleteFile()

### Summary

Deletes the file specified by the file handle.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_DELETE_FILE) (
    IN EFI_FILE_HANDLE FileHandle
);
```

### Parameters

*FileHandle*

The file handle to delete.

### Description

This function closes and deletes a file. In all cases, the file handle is closed. If the file cannot be deleted, the warning code **EFI\_WARN\_DELETE\_FAILURE** is returned, but the handle is still closed.

### Status Codes Returned

<b>EFI_SUCCESS</b>	The file was closed and deleted, and the handle was closed.
<b>EFI_WARN_DELETE_FAILURE</b>	The handle was closed but the file was not deleted.

## EFI\_SHELL\_PROTOCOL.DeleteFileByName()

### Summary

Deletes the file specified by the file handle.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_DELETE_FILE_BY_NAME) (
    IN CONST CHAR16 *FileName
);
```

### Parameters

*FileName*

Points to the null-terminated file name.

### Description

This function deletes a file.

### Status Codes Returned

<b>EFI_SUCCESS</b>	The file was closed and deleted, and the handle was closed.
<b>EFI_WARN_DELETE_FAILURE</b>	The handle was closed but the file was not deleted.

## EFI\_SHELL\_PROTOCOL.DisablePageBreak()

### Summary

Disables the page break output mode.

### Prototype

```
typedef  
VOID  
(EFI_API *EFI_SHELL_DISABLE_PAGE_BREAK) (  
    VOID  
);
```

### Parameters

None

### Description

This function disables the page break output mode.

### Status Codes Returned

None



## EFI\_SHELL\_PROTOCOL.EnablePageBreak()

### Summary

Enables the page break output mode.

### Prototype

```
typedef  
VOID  
(EFI_API *EFI_SHELL_ENABLE_PAGE_BREAK) (  
    VOID  
);
```

### Parameters

None

### Description

This function enables the page break output mode.

### Status Codes Returned

None

## EFI\_SHELL\_PROTOCOL.Execute()

### Summary

Execute the command line.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_EXECUTE) (
    IN EFI_HANDLE    *ParentImageHandle,
    IN CHAR16        *CommandLine OPTIONAL,
    IN CHAR16        **Environment OPTIONAL,
    OUT EFI_STATUS   *StatusCode OPTIONAL
);
```

### Parameters

#### *ParentImageHandle*

A handle of the image that is executing the specified command line.

#### *CommandLine*

Points to the null-terminated UCS-2 encoded string containing the command line. If NULL then the command-line will be empty.

#### *Environment*

Points to a null-terminated array of environment variables with the format 'x=y', where x is the environment variable name and y is the value. If this is NULL, then the current shell environment is used.

#### *ErrorCode*

Points to the status code returned by the command.

### Description

This function creates a nested instance of the shell and executes the specified command (*CommandLine*) with the specified environment (*Environment*). Upon return, the status code returned by the specified command is placed in *StatusCode*.

If *Environment* is **NULL**, then the current environment is used and all changes made by the commands executed will be reflected in the current environment. If the *Environment* is non-**NULL**, then the changes made will be discarded.

The *CommandLine* is executed from the current working directory on the current device.

### Status Codes Returned

<b>EFI_SUCCESS</b>	The command executed successfully. The status code returned by the command is
--------------------	---

	pointed to by <i>StatusCode</i> .
<b>EFI_INVALID_PARAMETER</b>	The parameters are invalid.
<b>EFI_OUT_OF_RESOURCES</b>	Out of resources.
<b>EFI_UNSUPPORTED</b>	Nested shell invocations are not allowed.

## EFI\_SHELL\_PROTOCOL.FindFiles()

### Summary

Find files that match a specified pattern.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_FIND_FILES) (
    IN CONST CHAR16 *FilePattern,
    OUT EFI_SHELL_FILE_INFO **FileList
);
```

### Parameters

*FilePattern*

Points to a null-terminated shell file path, including wildcards.

*FileList*

On return, points to the start of a file list containing the names of all matching files or else points to NULL if no matching files were found.

### Description

This function searches for all files and directories that match the specified *FilePattern*. The *FilePattern* can contain wild-card characters. The resulting file information is placed in the file list *FileList*.

The files in the file list are not opened. The OpenMode field is set to 0 and the FileInfo field is set to NULL.

### Status Codes Returned

<b>EFI_SUCCESS</b>	Files found.
<b>EFI_NOT_FOUND</b>	No files found.
<b>EFI_NO_MEDIA</b>	The device has no media
<b>EFI_DEVICE_ERROR</b>	The device reported an error
<b>EFI_VOLUME_CORRUPTED</b>	The file system structures are corrupted

## EFI\_SHELL\_PROTOCOL.FindFilesInDir()

### Summary

Find all files in a specified directory.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_FIND_FILES_IN_DIR) (
    IN EFI_FILE_HANDLE FileDirHandle,
    OUT EFI_SHELL_FILE_INFO **FileList
);
```

### Parameters

*FileDirHandle*

Handle of the directory to search.

*FileList*

On return, points to the list of files in the directory or NULL if there are no files in the directory.

### Status Codes Returned

<b>EFI_SUCCESS</b>	File information was returned successfully.
<b>EFI_VOLUME_CORRUPTED</b>	The file system structures have been corrupted.
<b>EFI_DEVICE_ERROR</b>	The device reported an error.
<b>EFI_NO_MEDIA</b>	The device media is not present.

## EFI\_SHELL\_PROTOCOL.FlushFile()

### Summary

Flushes data back to a device

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_FLUSH_FILE) (
    IN EFI_FILE_HANDLE FileHandle
);
```

### Parameters

*FileHandle*

The handle of the file to flush.

### Description

This function flushes all modified data associated with a file to a device.

### Status Codes Returned

<b>EFI_SUCCESS</b>	The data was flushed.
<b>EFI_NO_MEDIA</b>	The device has no medium.
<b>EFI_DEVICE_ERROR</b>	The device reported an error.
<b>EFI_VOLUME_CORRUPTED</b>	The file system structures are corrupted.
<b>EFI_WRITE_PROTECTED</b>	The file or medium is write-protected.
<b>EFI_ACCESS_DENIED</b>	The file was opened read-only.
<b>EFI_VOLUME_FULL</b>	The volume is full.

## EFI\_SHELL\_PROTOCOL.FreeFileList()

### Summary

Frees the file list.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_FREE_FILE_LIST) (
    IN EFI_SHELL_FILE_INFO **FileList
);
```

### Parameters

*FileList*

The file list to free. Type EFI\_SHELL\_FILE\_INFO is defined in OpenFileList()

### Description

This function cleans up the file list and any related data structures. It has no impact on the files themselves.

### Status Codes Returned

<b>EFI_SUCCESS</b>	Free the file list successfully.
--------------------	----------------------------------

## EFI\_SHELL\_PROTOCOL.GetCurDir()

### Summary

Returns the current directory on the specified device.

### Prototype

```
typedef
CONST CHAR16 *
(EFI_API *EFI_SHELL_GET_CUR_DIR) (
    IN CONST CHAR16 *FileSystemMapping OPTIONAL
);
```

### Parameters

*FileSystemMapping*

A pointer to the file system mapping. If **NULL**, then the current working directory is returned.

### Description

If *FileSystemMapping* is **NULL**, it returns the current working directory. If the *FileSystemMapping* is not **NULL**, it returns the current directory associated with the *FileSystemMapping*. In both cases, the returned name includes the file system mapping (i.e. **fs0:\current-dir**).

For more information, see "Current Directory", section 3.5.

### Status Codes Returned

≠NULL	The current directory.
NULL	Current directory does not exist.



## EFI\_SHELL\_PROTOCOL.GetDeviceName()

### Summary

Gets the name of the device specified by the device handle.

### Prototype

```
typedef
EFI_STATUS
(*EFI_SHELL_GET_DEVICE_NAME) (
    IN EFI_HANDLE DeviceHandle,
    IN EFI_SHELL_DEVICE_NAME_FLAGS Flags,
    IN CHAR8 *Language,
    OUT CHAR16 **BestDeviceName
);
```

### Parameters

#### *DeviceHandle*

The handle of the device.

#### *Flags*

Determines the possible sources of component names. See “Related Definitions” below for more information.

#### *Language*

A pointer to the language specified for the device name, in the same format as described in the UEFI specification, Appendix M

#### *BestDeviceName*

On return, points to the callee-allocated null-terminated name of the device. If no device name could be found, points to NULL. The name must be freed by the caller..

### Description

This function gets the user-readable name of the device specified by the device handle. If no user-readable name could be generated, then *\*BestDeviceName* will be **NULL** and **EFI\_NOT\_FOUND** will be returned.

The

### Related Definitions

```
typedef UINT32 EFI_DEVICE_NAME_FLAGS;
#define EFI_DEVICE_NAME_USE_COMPONENT_NAME 0x00000001
#define EFI_DEVICE_NAME_USE_DEVICE_PATH 0x00000002
```

If **EFI\_DEVICE\_NAME\_USE\_COMPONENT\_NAME** is set, then the function will return the device's name using the **EFI\_COMPONENT\_NAME2\_PROTOCOL**, if present on *DeviceHandle*.

If `EFI_DEVICE_NAME_USE_DEVICE_PATH` is set, then the function will return the device's name using the `EFI_DEVICE_PATH_PROTOCOL`, if present on *DeviceHandle*.

If both `EFI_DEVICE_NAME_USE_COMPONENT_NAME` and `EFI_DEVICE_NAME_USE_DEVICE_PATH` are set, then `EFI_DEVICE_NAME_USE_COMPONENT_NAME` will have higher priority.

### Status Codes Returned

EFI_SUCCESS	Get the name successfully.
EFI_NOT_FOUND	Fail to get the device name.

## EFI\_SHELL\_PROTOCOL.GetDevicePathFromMap()

### Summary

Gets the device path from the mapping.

### Prototype

```
typedef  
CONST EFI_DEVICE_PATH_PROTOCOL *  
(EFI_API *EFI_SHELL_GET_DEVICE_PATH_FROM_MAP) (  
    IN CONST CHAR16 *Mapping  
);
```

### Parameters

*Mapping*

A pointer to the mapping.

### Description

This function gets the device path associated with a mapping.

### Status Codes Returned

≠NULL	Pointer to the device path that corresponds to the device mapping. The returned pointer does not need to be freed.
NULL	There is no device path associated with the specified mapping.

## EFI\_SHELL\_PROTOCOL.GetDevicePathFromFilePath()

### Summary

Converts a file system style name to a device path.

### Prototype

```
typedef
EFI_DEVICE_PATH_PROTOCOL *
(EFI_API *EFI_SHELL_GET_DEVICE_PATH_FROM_FILE_PATH) (
    IN CONST CHAR16 *Path
);
```

### Parameters

*Path*  
The pointer to the path.

### Description

This function converts a file system style name to a device path, by replacing any mapping references to the associated device path.

### Status Codes Returned

The pointer of the file path. The file path is callee allocated and should be freed by the caller.

## EFI\_SHELL\_PROTOCOL.GetEnv()

### Summary

Gets the environment variable.

### Prototype

```
typedef  
CONST CHAR16 *  
(EFI_API *EFI_SHELL_GET_ENV) (  
    IN CONST CHAR16 *Name  
);
```

### Parameters

*Name*

A pointer to the environment variable name.

### Description

This function returns the current value of the specified environment variable.

### Status Codes Returned

≠NULL	The environment variable's value. The returned pointer does not need to be freed by the caller.
NULL	The environment variable doesn't exist.

## EFI\_SHELL\_PROTOCOL.GetFileInfo()

### Summary

Gets the file information from an open file handle.

### Prototype

```
typedef
EFI_FILE_INFO *
(EFI_API *EFI_SHELL_GET_FILE_INFO) (
    IN EFI_FILE_HANDLE    FileHandle
);
```

### Parameters

*FileHandle*

A file handle

### Description

This function allocates a buffer to store the file's information. It's the caller's responsibility to free the buffer.

### Returns

≠NULL	Cannot get the file info.
NULL	A pointer to a buffer with file information.

## EFI\_SHELL\_PROTOCOL.GetFilePathFromDevicePath()

### Summary

Converts a device path to a file system-style path.

### Prototype

```
typedef
CHAR16 *
(EFI_API *EFI_SHELL_GET_FILE_PATH_FROM_DEVICE_PATH) (
    IN CONST EFI_DEVICE_PATH_PROTOCOL *Path
);
```

### Parameters

*Path*  
The pointer to the device path.

### Description

This function converts a device path to a file system path by replacing part, or all, of the device path with the file-system mapping. If there are more than one application file system mappings, the one that most closely matches *Path* will be used.

### Returned Value

The pointer of the null-terminated file path. The path is callee-allocated and should be freed by the caller.

## EFI\_SHELL\_PROTOCOL.GetFilePosition()

### Summary

Gets a file's current position

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_GET_FILE_POSITION) (
    IN EFI_FILE_HANDLE FileHandle,
    OUT UINT64          *Position
);
```

### Parameters

*FileHandle*

The file handle on which to get the current position.

*Position*

Byte position from the start of the file

### Description

This function returns the current file position for the file handle. For directories, the current file position has no meaning outside of the file system driver and as such, the operation is not supported.

### Status Codes Returns

<b>EFI_SUCCESS</b>	Data was accessed.
<b>EFI_UNSUPPORTED</b>	The request is not valid on open directories.



## EFI\_SHELL\_PROTOCOL.GetFileSize()

### Summary

Gets the size of a file.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_GET_FILE_SIZE) (
    IN EFI_FILE_HANDLE FileHandle,
    OUT UINT64          *Size
);
```

### Parameters

*FileHandle*

The handle of the file.

*Size*

The size of this file.

### Description

This function returns the size of the file specified by *FileHandle*.

### Status Codes Returned

<b>EFI_SUCCESS</b>	Get the file's size.
<b>EFI_DEVICE_ERROR</b>	Can't access the file.

## EFI\_SHELL\_PROTOCOL.GetHelpText()

### Summary

Return help information about a specific command.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_GET_HELP_TEXT) (
    IN CONST CHAR16 *Command,
    IN CONST CHAR16 *Sections,
    OUT CHAR16      **HelpText
);
```

### Parameters

*Command*

Points to the null-terminated UEFI Shell command name.

*Sections*

Points to the null-terminated comma-delimited section names to return. If NULL, then all sections will be returned.

*HelpText*

On return, points to a callee-allocated buffer containing all specified help text.

### Description

This function returns the help information for the specified command. The help text can be internal to the shell or can be from a UEFI Shell manual page, as described in Appendix B

If Sections is specified, then each section name listed will be compared in a case-sensitive manner, to the section names described in Appendix B. If the section exists, it will be appended to the returned help text. If the section does not exist, no information will be returned. If Sections is NULL, then all help text information available will be returned.

### Status Codes Returned

<b>EFI_SUCCESS</b>	The help text was returned.
<b>EFI_OUT_OF_RESOURCES</b>	The necessary buffer could not be allocated to hold the returned help text.
<b>EFI_INVALID_PARAMETER</b>	<i>HelpText</i> is NULL
<b>EFI_NOT_FOUND</b>	There is no help text available for <i>Command</i> .



## EFI\_SHELL\_PROTOCOL.GetMapFromDevicePath()

### Summary

Gets the mapping that most closely matches the device path.

### Prototype

```
typedef
CONST CHAR16 *
(EFI_API *EFI_SHELL_GET_MAP_FROM_DEVICE_PATH) (
    IN OUT EFI_DEVICE_PATH_PROTOCOL **DevicePath
);
```

### Parameters

*DevicePath*

On entry, points to a device path pointer. On exit, updates the pointer to point to the portion of the device path after the mapping.

### Description

This function gets the mapping which corresponds to the device path *\*DevicePath*. If there is no exact match, then the mapping which most closely matches *\*DevicePath* is returned, and *\*DevicePath* is updated to point to the remaining portion of the device path. If there is an exact match, the mapping is returned and *\*DevicePath* points to the end-of-device-path node.

### Returned Value

!NULL	Pointer to null-terminated mapping. The buffer is callee-allocated and should be freed by the caller.
NULL	No mapping was found.

## EFI\_SHELL\_PROTOCOL.GetPageBreak()

### Summary

Gets the enable status of the page break output mode.

### Prototype

```
typedef  
BOOLEAN  
(EFI_API *EFI_SHELL_GET_PAGE_BREAK) (  
    VOID  
);
```

### Parameters

None

### Description

User can use this function to determine current page break mode.

### Status Codes Returned

TRUE	The page break output mode is enabled
FALSE	The page break output mode is disabled

## EFI\_SHELL\_PROTOCOL.IsRootShell()

### Summary

Judges whether the active shell is the root shell.

### Prototype

```
typedef
BOOLEAN
(EFI_API *EFI_SHELL_IS_ROOT_SHELL) (
    VOID
);
```

### Parameters

None

### Description

This function makes the user to know that whether the active Shell is the root shell.

### Status Codes Returned

TRUE	The active Shell is the root Shell.
FALSE	The active Shell is NOT the root Shell.

## EFI\_SHELL\_PROTOCOL.OpenFileByName()

### Summary

Opens a file or a directory by file name.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_OPEN_FILE_BY_NAME) (
    IN CONST CHAR16 *FileName,
    OUT EFI_FILE_HANDLE *FileHandle,
    IN UINT64 OpenMode
);
```

### Parameters

*FileName*

Points to the null-terminated UCS-2 encoded file name.

*FileHandle*

On return, points to the file handle.

*OpenMode*

File open mode. Either `EFI_FILE_MODE_READ` or `EFI_FILE_MODE_WRITE` from section 12.4 of the UEFI Specification.

### Related Definitions

```
typedef VOID *EFI_FILE_HANDLE;
```

### Description

This function opens the specified file in the specified *OpenMode* and returns a file handle.

If the file name begins with >v, then the file handle which is returned refers to the shell environment variable with the specified name. If the shell environment variable exists, is non-volatile and the *OpenMode* indicates `EFI_FILE_MODE_WRITE`, then `EFI_INVALID_PARAMETER` is returned.

If the file name is >i, then the file handle which is returned refers to the standard input. If the *OpenMode* indicates `EFI_FILE_MODE_WRITE`, then `EFI_INVALID_PARAMETER` is returned.

If the file name is >o, then the file handle which is returned refers to the standard output. If the *OpenMode* indicates `EFI_FILE_MODE_READ`, then `EFI_INVALID_PARAMETER` is returned.

If the file name is >e, then the file handle which is returned refers to the standard error. If the *OpenMode* indicates `EFI_FILE_MODE_READ`, then `EFI_INVALID_PARAMETER` is returned.

If the file name is NUL, then the file handle that is returned refers to the standard NUL file. If the *OpenMode* indicates **EFI\_FILE\_MODE\_READ**, then **EFI\_INVALID\_PARAMETER** is returned.

If return **EFI\_SUCCESS**, the *FileHandle* is the opened file's handle, else, the *FileHandle* is **NULL**.

## Status Codes Returned

<b>EFI_SUCCESS</b>	The file was opened. FileHandle has the opened file's handle.
<b>EFI_INVALID_PARAMETER</b>	One of the parameters has an invalid value. FileHandle is NULL.
<b>EFI_UNSUPPORTED</b>	Could not open the file path. FileHandle is NULL.
<b>EFI_NOT_FOUND</b>	The specified file could not be found on the device or the file system could not be found on the device. FileHandle is NULL.
<b>EFI_NO_MEDIA</b>	The device has no medium. FileHandle is NULL.
<b>EFI_MEDIA_CHANGED</b>	The device has a different medium in it or the medium is no longer supported. FileHandle is NULL.
<b>EFI_DEVICE_ERROR</b>	The device reported an error or can't get the file path according the <i>FileName</i> . FileHandle is NULL.
<b>EFI_VOLUME_CORRUPTED</b>	The file system structures are corrupted. FileHandle is NULL.
<b>EFI_WRITE_PROTECTED</b>	An attempt was made to create a file, or open a file for write when the media is write-protected. FileHandle is NULL.
<b>EFI_ACCESS_DENIED</b>	The service denied access to the file. FileHandle is NULL.
<b>EFI_OUT_OF_RESOURCES</b>	Not enough resources were available to open the file. FileHandle is NULL.
<b>EFI_VOLUME_FULL</b>	The volume is full. FileHandle is NULL.



## EFI\_SHELL\_PROTOCOL.OpenFileList()

### Summary

Opens the files that match the path specified.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_OPEN_FILE_LIST) (
    IN      CHAR16      *Path,
    IN      UINT64      OpenMode,
    IN OUT  EFI_SHELL_FILE_INFO **FileList
);
```

### Parameters

*Path*

A pointer to the path string.

*OpenMode*

Specifies the mode used to open each file, `EFI_FILE_MODE_READ` or `EFI_FILE_MODE_WRITE`.

*FileList*

Points to the start of a list of files opened.

### Description

This function opens all of the files specified by *Path*. Wildcards are processed according to the rules specified in 3.7.1. Each matching file has an `EFI_SHELL_FILE_INFO` structure created in a linked list.

### Related Definitions

```
typedef struct _EFI_LIST_ENTRY {
    struct _EFI_LIST_ENTRY *Flink;
    struct _EFI_LIST_ENTRY *Blink;
} EFI_LIST_ENTRY;

typedef struct {
    EFI_LIST_ENTRY      Link;
    EFI_STATUS          Status;
    CONST CHAR16        *FullName;
    CONST CHAR16        *FileName;
    EFI_FILE_HANDLE     Handle;
    EFI_FILE_INFO       *Info;
} EFI_SHELL_FILE_INFO;
```

*Link*

Points to the next and previous entries in the file list. If NULL, then no more files.

*Status*

The status returned when calling `OpenFile()` for the entry in the file list.

Version 2.0

*FullName*

Specifies the full name of the file, including the path.

*Handle*

The file handle of the file after it was opened.

*Info*

The file information for the opened file.

### Status Codes Returned

<b>EFI_SUCCESS</b>	Create the file list successfully.
Others	Can't create the file list.

## EFI\_SHELL\_PROTOCOL.OpenRoot()

### Summary

Opens the root directory of a device.

### Prototype

```
typedef
EFI_STATUS
(EFI_API EFI_SHELL_OPEN_ROOT) (
    IN  EFI_DEVICE_PATH_PROTOCOL *DevicePath
    OUT EFI_FILE_HANDLE          *FileHandle
);
```

### Parameters

*DevicePath*

Points to the device path corresponding to the device where the EFI\_SIMPLE\_FILE\_SYSTEM\_PROTOCOL is installed.

*FileHandle*

On exit, points to the file handle corresponding to the root directory on the device.

### Description

This function opens the root directory of a device and returns a file handle to it.

### Status Codes Returned

<b>EFI_SUCCESS</b>	Root opened successfully.
<b>EFI_NOT_FOUND</b>	EFI_SIMPLE_FILE_SYSTEM could not be found or the root directory could not be opened.
<b>EFI_VOLUME_CORRUPTED</b>	The data structures in the volume were corrupted.
<b>EFI_DEVICE_ERROR</b>	The device had an error

## EFI\_SHELL\_PROTOCOL.OpenRootByHandle()

### Summary

Opens the root directory of a device on a handle

### Prototype

```
typedef
EFI_STATUS
(EFI_API EFI_SHELL_OPEN_ROOT_BY_HANDLE) (
    IN  EFI_HANDLE      DeviceHandle,
    OUT EFI_FILE_HANDLE *FileHandle
);
```

### Parameters

*DeviceHandle*

The handle of the device that contains the volume.

*FileHandle*

On exit, points to the file handle corresponding to the root directory on the device.

### Description

This function opens the root directory of a device and returns a file handle to it.

### Status Codes Returned

<b>EFI_SUCCESS</b>	Root opened successfully.
<b>EFI_NOT_FOUND</b>	EFI_SIMPLE_FILE_SYSTEM could not be found or the root directory could not be opened.
<b>EFI_VOLUME_CORRUPTED</b>	The data structures in the volume were corrupted.
<b>EFI_DEVICE_ERROR</b>	The device had an error

## EFI\_SHELL\_PROTOCOL.ReadFile()

### Summary

Reads data from the file.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_READ_FILE) (
    IN     EFI_FILE_HANDLE  FileHandle,
    IN OUT UINTN            *ReadSize,
    OUT    VOID              *Buffer
);
```

### Parameters

*FileHandle*

The opened file handle for read

*ReadSize*

On input, the size of *Buffer*, in bytes. On output, the amount of data read.

*Buffer*

The buffer in which data is read.

### Description

If *FileHandle* is not a directory, the function reads the requested number of bytes from the file at the file's current position and returns them in *Buffer*. If the read goes beyond the end of the file, the read length is truncated to the end of the file. The file's current position is increased by the number of bytes returned.

If *FileHandle* is a directory, then an error is returned.

### Status Codes Returned

<b>EFI_SUCCESS</b>	Data was read.
<b>EFI_NO_MEDIA</b>	The device has no media
<b>EFI_DEVICE_ERROR</b>	The device reported an error
<b>EFI_VOLUME_CORRUPTED</b>	The file system structures are corrupted
<b>EFI_BUFFER_TOO_SMALL</b>	Buffer is too small. <i>ReadSize</i> contains required size

## EFI\_SHELL\_PROTOCOL.RemoveDupInFileList()

### Summary

Deletes the duplicate file names files in the given file list.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_REMOVE_DUP_IN_FILE_LIST) (
    IN EFI_SHELL_FILE_INFO **FileList
);
```

### Parameters

*FileList*

A pointer to the first entry in the file list.

### Description

This function deletes the reduplicate files in the given file list.

### Status Codes Returned

<b>EFI_SUCCESS</b>	Always success.
--------------------	-----------------

## EFI\_SHELL\_PROTOCOL.SetAlias()

### Summary

Changes a shell command alias.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_SET_ALIAS) (
    IN CONST CHAR16 *Command,
    IN CONST CHAR16 *Alias,
    IN BOOLEAN      Replace
);
```

### Parameters

#### *Command*

Points to the null-terminated shell command or existing alias.

#### *Alias*

Points to the null-terminated alias for the shell command. If this is NULL, and Command refers to an alias, that alias will be deleted.

#### *Replace*

If TRUE and the alias already exists, then the existing alias will be replaced. If FALSE and the alias already exists, then the existing alias is unchanged and EFI\_ACCESS\_DENIED is returned.

### Description

This function creates an alias for a shell command.

### Return Value

<b>EFI_SUCCESS</b>	Alias created or deleted successfully.
<b>EFI_ACCESS_DENIED</b>	The alias is a built-in alias or already existed and <i>Replace</i> was set to <b>FALSE</b> .

## EFI\_SHELL\_PROTOCOL.SetCurDir()

### Summary

Changes the current directory on the specified device.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_SET_CUR_DIR) (
    IN CONST CHAR16 *FileSystem OPTIONAL,
    IN CONST CHAR16 *Dir
);
```

### Parameters

*FileSystem*

A pointer to the file system's mapped name. If **NULL**, then the current working directory is changed.

*Dir*

Points to the null-terminated directory on the device specified by *FileSystem*.

### Description

If the *FileSystem* is **NULL**, and the directory *Dir* does not contain a file system's mapped name, this function changes the current working directory. If *FileSystem* is **NULL** and the directory *Dir* contains a mapped name, then the current file system and the current directory on that file system are changed.

If *FileSystem* is not **NULL**, and *Dir* is **NULL**, then this changes the current working file system.

If *FileSystem* is not **NULL** and *Dir* is not **NULL**, then this function changes the current directory on the specified file system.

If the current working directory or the current working file system is changed then the **%cwd%** environment variable will be updated. For more information, see "Current Directory" , section3.5.

### Status Codes Returned

≠NULL	The current directory.
NULL	Current directory does not exist.



## EFI\_SHELL\_PROTOCOL.SetEnv()

### Summary

Sets the environment variable.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_SET_ENV) (
    IN CONST CHAR16 *Name,
    IN CONST CHAR16 *Value,
    IN BOOLEAN      Volatile
);
```

### Parameters

*Name*

Points to the null-terminated environment variable name.

*Value*

Points to the null-terminated environment variable value. If the value is an empty string then the environment variable is deleted.

*Volatile*

Indicates whether the variable is non-volatile (**FALSE**) or volatile (**TRUE**).

### Description

This function changes the current value of the specified environment variable. If the environment variable exists and the Value is an empty string, then the environment variable is deleted. If the environment variable exists and the Value is not an empty string, then the value of the environment variable is changed. If the environment variable does not exist and the Value is an empty string, there is no action. If the environment variable does not exist and the Value is a non-empty string, then the environment variable is created and assigned the specified value.

For a description of volatile and non-volatile environment variables, see 3.6.1.

### Status Codes Returned

EFI_SUCCESS	The environment variable was successfully updated.
-------------	--

## EFI\_SHELL\_PROTOCOL.SetFileInfo()

### Summary

Sets the file information to an opened file handle.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_SET_FILE_INFO) (
    IN EFI_FILE_HANDLE FileHandle,
    IN CONST EFI_FILE_INFO *FileInfo
);
```

### Parameters

*FileHandle*

A file handle

*FileInfo*

Points to new file information.

### Description

This function changes file information.

### Status Codes Returned

<b>EFI_SUCCESS</b>	The information was set.
<b>EFI_NO_MEDIA</b>	The device has no medium.
<b>EFI_DEVICE_ERROR</b>	The device reported an error.
<b>EFI_VOLUME_CORRUPTED</b>	The file system structures are corrupted.
<b>EFI_WRITE_PROTECTED</b>	The file or medium is write-protected.
<b>EFI_ACCESS_DENIED</b>	The file was opened read-only.
<b>EFI_VOLUME_FULL</b>	The volume is full.
<b>EFI_BAD_BUFFER_SIZE</b>	<i>BufferSize</i> is smaller than the size of <b>EFI_FILE_INFO</b> .

## EFI\_SHELL\_PROTOCOL.SetFilePosition()

### Summary

Sets a file's current position

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_SET_FILE_POSITION) (
    IN EFI_FILE_HANDLE FileHandle,
    IN UINT64           Position
);
```

### Parameters

*FileHandle*

The file handle on which requested position will be set.

*Position*

Byte position from the start of the file

### Description

This function sets the current file position for the handle to the position supplied. With the exception of seeking to position **0xFFFFFFFFFFFFFFFF**, only absolute positioning is supported, and seeking past the end of the file is allowed (a subsequent write would grow the file). Seeking to position **0xFFFFFFFFFFFFFFFF** causes the current position to be set to the end of the file.

### Status Codes Returned

<b>EFI_SUCCESS</b>	Data was written.
<b>EFI_UNSUPPORTED</b>	The seek request for nonzero is not valid on open directories.

## EFI\_SHELL\_PROTOCOL.SetMap()

### Summary

Changes a shell command alias.

### Prototype

```
typedef
EFI_STATUS
(EFI_API *EFI_SHELL_SET_MAP) (
    IN CONST EFI_DEVICE_PATH_PROTOCOL *DevicePath,
    IN CONST CHAR16 *Mapping
);
```

### Parameters

*DevicePath*

Points to the device path. If this is NULL and Mapping points to a valid mapping, then the mapping will be deleted.

*Mapping*

Points to the null-terminated mapping for the device path.

### Description

This function creates a mapping for a device path.

### Return Value

<b>EFI_SUCCESS</b>	Mapping created or deleted successfully.
<b>EFI_NO_MAPPING</b>	There is no handle that corresponds exactly to <i>DevicePath</i> . See the boot service function <code>LocateDevicePath()</code> .
<b>EFI_ACCESS_DENIED</b>	The mapping is a built-in alias.

## EFI\_SHELL\_PROTOCOL.WriteFile()

### Summary

Writes data to the file.

### Prototype

```
typedef
EFI_STATUS
(EFI_API EFI_SHELL_WRITE_FILE) (
    IN     EFI_FILE_HANDLE  FileHandle,
    IN OUT UINTN            *BufferSize,
    OUT    VOID              *Buffer
);
```

### Parameters

*FileHandle*

The opened file handle for writing.

*BufferSize*

On input, size of *Buffer*.

*Buffer*

The buffer in which data to write.

### Description

This function writes the specified number of bytes to the file at the current file position. The current file position is advanced the actual number of bytes written, which is returned in *BufferSize*. Partial writes only occur when there has been a data error during the write attempt (such as "volume space full"). The file automatically grows to hold the data, if required.

Direct writes to opened directories are not supported.

### Status Codes Returned

<b>EFI_SUCCESS</b>	Data was written.
<b>EFI_UNSUPPORTED</b>	Writes to open directory are not supported
<b>EFI_NO_MEDIA</b>	The device has no media
<b>EFI_DEVICE_ERROR</b>	The device reported an error
<b>EFI_VOLUME_CORRUPTED</b>	The file system structures are corrupted
<b>EFI_WRITE_PROTECTED</b>	The device is write-protected
<b>EFI_ACCESS_DENIED</b>	The file was open for read only
<b>EFI_VOLUME_FULL</b>	The volume is full

## 2.3 EFI\_SHELL\_PARAMETERS\_PROTOCOL

### EFI\_SHELL\_PARAMETERS\_PROTOCOL

#### Summary

Shell application's arguments.

#### GUID

```
#define EFI_SHELL_PARAMETERS_PROTOCOL_GUID \
{ 0x752f3136, 0x4e16, 0x4fdc, \
  { 0xa2, 0x2a, 0xe5, 0xf4, 0x68, 0x12, 0xf4, 0xca } };
```

#### Prototype

```
typedef struct _EFI_SHELL_PARAMETERS_PROTOCOL {
    CHAR16          **Argv;
    UINTN           Argc;
    EFI_FILE_HANDLE StdIn;
    EFI_FILE_HANDLE StdOut;
    EFI_FILE_HANDLE StdErr;
} EFI_SHELL_PARAMETERS_PROTOCOL;
```

#### Parameters

##### *Argv*

Points to an Argc-element array of points to null-terminated strings containing the command-line parameters. The first entry in the array is always the full file path of the executable. Any quotation marks that were used to preserve whitespace have been removed.

##### *Argc*

The number of elements in the *Argv* array.

##### *StdIn*

The file handle for the standard input for this executable. This may be different from the ConInHandle in the **EFI\_SYSTEM\_TABLE**.

##### *StdOut*

The file handle for the standard output for this executable. This may be different from the ConOutHandle in the **EFI\_SYSTEM\_TABLE**.

##### *StdErr*

The file handle for the standard error output for this executable. This may be different from the StdErrHandle in the **EFI\_SYSTEM\_TABLE**.

#### Description

An instance of this protocol is installed on each shell application's image handle prior to calling **StartImage()**. It describes all of the command-line parameters passed on the command line, as well as the standard file handles for standard input, output and error output.







## 3 UEFI Shell Features

The UEFI Shell supports an interactive command-line interface, scripting, and a standard set of commands.

All the commands can be invoked by entering the name of the command at the command prompt. For external commands, they must reside in a file system. So, to run them users need to have at least one mapped file system and put those external commands under this file system.

The active drive may be changed by entering the mapped name followed by a ':' at the command prompt.

### 3.1 Levels Of Support

This section describes the different standard levels of shell support. The different standard levels are designed to provide different feature/size tradeoffs for different uses. The levels mentioned here are referenced throughout this specification

**Table 1 Support Levels**

Level	Name	Execute()/Scripting/ startup.nsh	PATH?	ALIAS?	Interactive?	Commands
0	Minimal	No	No	No	No	None
1	Scripting	Yes	Yes	No	No	for, endfor, goto, if, else, endif, shift, exit
2	Basic	Yes	Yes	Yes	No	attrib, cd, cp, date*, time*, del, load, ls, map, mkdir, mv, rm, reset, set, timezone*
3	Interactive	Yes	Yes	Yes	Yes	alias, date, echo, help, pause, time, touch, type, ver, cls, timezone

\*Non-interactive forms only

**Execute()/Scripting/startup.nsh** support indicates whether the Execute() function is supported by the EFI\_SHELL\_PROTOCOL, whether or not scripts are supported and whether the default startup script startup.nsh is supported.

**PATH** support determines whether the PATH environment variable will be used to determine the location of executables.

**ALIAS** support determines whether the ALIAS environment variable will be used to determine alternate names for shell commands.

**Interactive** determines whether or not an interactive session can be started.

For more information on scripting, see UEFI Shell Script (Chapter 4).

For more information on processing of the startup.nsh file, please see UEFI Shell Initialization (section 3.3).

The shell must remain compliant with its advertised *shellsupport* command profile. There can be cases where a shell implementation may not want to expose certain commands to all users. If a shell implementation wants to make a particular command inaccessible to a particular user, they must properly interpret the command request and return an appropriate return code, such as SHELL\_SECURITY\_VIOLATION.

## 3.2 Invocation

The UEFI Shell is a UEFI application. The UEFI Shell takes command-line options that are null-terminated UCS-2 encoded strings. The syntax is:

```
shell.efi [ShellOpt-options] [options] [file-name [file-name-options]]
```

The command-line options are separated by the space or tab character. The options are processed left-to-right retrieved from the *LoadOptions* field of the **EFI\_LOADED\_IMAGE\_PROTOCOL**.

The following table describes the standard command-line options.

**Table 2 Standard Command Line Options**

Option	Description
file-name	The name of a UEFI shell application or script to be executed after initialization is complete. By default, if <i>file-name</i> is specified, then <b>-nostartup</b> is implied. Scripts are not supported by level 0.
file-name-options	The command-line options that are passed to <i>file-name</i> when it is invoked.
options	Options (from table X below) which control the initialization behavior of the shell.
ShellOpt-options	Options (from table X below) which control the initialization behavior of the shell. These options are read from the EFI global variable "ShellOpt" and are processed before <i>options</i> or <i>file-name</i> .

**Table 3: UEFI Shell Invocation Options**

<b>-nostartup</b>	The default startup script <b>startup.nsh</b> will not be executed.
<b>-noconsoleout</b>	Console output from the shell applications will not be displayed. This has no effect for UEFI Shells that do not support an interactive mode.

<b>-noconsolein</b>	Console input will not be accepted from the user. This has no effect for UEFI Shells that do not support an interactive mode.
<b>-delay[:n]</b>	Specifies the integer number of seconds the shell will delay prior to the execution of <b>startup.nsh</b> . Ignored for shell level 0 or if <b>-nostartup</b> is specified. If <i>n</i> is not specified, the default is 5 seconds. If 0 is specified, then there will be no delay. If <b>-nointerrupt</b> is specified, then there will be no delay.
<b>-nointerrupt</b>	Execution interruption (as described in Execution Interrupt Support) is not allowed. This has no effect for UEFI Shells that do not support an interactive mode.
<b>-nomap</b>	The default mappings will not be displayed.
<b>-noversion</b>	The version information will not be displayed.
<b>-startup</b>	The default startup script <b>startup.nsh</b> will be executed. Requires shell support level 1 or higher. This overrides the default behavior when <i>file-name</i> is specified.

### 3.3 Initialization

This section describes the steps taken during shell initialization. The following steps are not exhaustive, but they are executed in order:

1. The command-line options of the shell itself are processed.
2. Default file system (FSx:) mapped names are created. The current directory for each mapped name is set to the root.
3. The default alias settings are read from non-volatile storage. This is only supported in shell level 2.
4. The default environment variable settings are read from non-volatile storage.
5. The profiles are read into the 'profiles' environment variable.
6. If the shell supports interactive mode and console output support is enabled, then the console is cleared.
7. The platform watchdog will be cleared through the UEFI SetWatchdogTimer() API to avoid inadvertent platform resets during long operations within the UEFI Shell environment.
8. If the shell supports interactive mode and console output support is enabled and version support (see **-noversion**) is enabled, then the equivalent of **ver** will be executed.
9. If the shell supports interactive mode and console output support is enabled and map support is enabled (see **-nomap**), then the equivalent of **map -terse** will be executed.
10. If interactive mode is supported and execution interrupt is supported (see **-nointerrupt**) then the shell will wait for the number of seconds specified by the **-delay** option.

Formatted: Bullets and Numbering

11. If `startup.nsh` is supported and enabled (see `-startup` and `-nostartup`), the script will be launched.
12. If a file name was specified among the command-line options, then the image or script (if supported) is launched.

### 3.3.1 Finding `startup.nsh`

When executing `startup.nsh`, the shell will search for it first in the directory where the shell itself was launched. If it cannot find the `startup.nsh` file in that directory or it was not launched from a file system, it will search the execution path defined by the environment variable `PATH`.

### 3.3.2 Supported Profiles

The UEFI Shell may have support for zero or more profiles, such as those described in chapter 5 built in to its executable. Additional profiles are described in the file 'profiles.txt', located in the same directory as the UEFI Shell executable. The contents of the file are carriage-return delimited (one profile name per line) and are read into the UEFI shell environment variable 'profiles', where they are semicolon (;) delimited. Profiles names that begin with 'UEFI' are reserved for use in this specification.

## 3.4 Command-Line

The UEFI Shell implements a programming language that provides control over the execution of individual commands. Command names and keywords in certain commands are all case insensitive.

The UEFI Shell processes the command-line by

1. Substituting aliases. The UEFI Shell supports specifying aliases for UEFI Shell commands (both internal and external). The substitution is performed automatically on the first command-line parameter.
2. Substituting variables. The UEFI Shell finds the % character and substitutes the value of an Environment Variable, Positional Parameter or Index Parameter based on the characters found after the % character. See Variables (section, 3.6) for more information.
3. Setting up input and output redirection. Using special characters on the command-line, the UEFI Shell can get input from a file and send output to a file. See Redirection (section 3.4.4) for more information.
4. Breaking up the command-line into arguments. The arguments are delimited by non-quoted whitespace characters.
5. Launching the shell command or UEFI Shell script.

Formatted: Bullets and Numbering

### 3.4.1 Special Characters

When the shell scans its input, it always treats certain characters (**#**, **>**, **<**, **|**, **%**, **\***, **?**, **^**, **\**, **space**, **[**, **]** and **newline**) specially. The usage of these characters is briefly listed here:

**Table 3 Special Characters in Shell**

Character	Description
newline	Ends a command line.
space	Ends an argument, if it is not in a quotation.
#	Starts a comment.
>	Used for output redirection.
<	Used for input redirection
	Used for pipe command support.
%	Used to delimit a variable or an argument.
"	Used to delimit a quotation.
^	Prevents the next character from being interpreted as having special meaning. Can be used inside quoted strings.
*, ?, [, ]	Wildcards to specify multiple similar file names.

### 3.4.2 Escape Characters

The escaping character **^** is used to prevent interpreting the character that immediately follows it as a special character.

### 3.4.3 Quoting

The UEFI Shell uses quotation marks for argument grouping. Normally, the UEFI Shell will interpret a one or more whitespace character as an argument delimiter. However, if the whitespace character appears between double quotation marks, it will be ignored for the purposes of argument delimiting. Empty strings are treated as valid command line arguments. Substitution of environment variables and positional parameters still occurs within quotation marks.

Double-quotation marks that surround arguments are stripped before they are passed to the entry point of a shell application. For more information, see the *Argv* member of the **EFI\_SHELL\_PARAMETERS\_PROTOCOL**.

Double-quotation marks that surround arguments are not stripped in positional parameters (see Positional Parameters, section 3.6.2) or on the copy of the command line passed in the *LoadOptions* member of the **EFI\_LOADED\_IMAGE\_PROTOCOL** passed to shell applications.

To include a double-quotation mark inside of a quoted string, use **^"**. To include a **^** character inside of a quoted string, use **^^**.

For information about how the quotes are treated in each of the options, see “Shell Commands”, chapter 5).

### 3.4.4 Redirection

#### 3.4.4.1 Output Redirection

Output of EFI Shell commands can be redirected to files. For example:

```
Command > ucs2_output_file_pathname
Command >a ascii_output_file_pathname
Command 1> ucs-2_output_file_pathname
Command 1>a ascii_output_file_pathname
Command 2> ucs-2_output_file_pathname
Command 2>a ascii_output_file_pathname
Command >> ucs-2_output_file_pathname
Command >>a ascii_output_file_pathname
Command 1>> ucs-2_output_file_pathname
Command 1>>a ascii_output_file_pathname
```

Table 4 shows the special character combinations that are used to denote output redirection operations:

**Table 4 Output Redirection Syntax**

Character Sequence	Description
>	Redirect standard output to a UCS-2 encode file.
>a	Redirect standard output to an ASCII file.
1>	Redirect standard output to a UCS-2 encoded file.
1>a	Redirect standard output to an ASCII file.
2>	Redirect standard error to a UCS-2 encoded file.
2>a	Redirect standard error to an ASCII file.
>v	Redirect standard output to an environment variable, encoded as UCS-2.
1>v	Redirect standard output to an environment variable, encoded as UCS-2.
2>v	Redirect standard error to an environment variable, encoded as UCS-2.
>>	Redirect standard output appended to a UCS-2 encoded file.
>>a	Redirect standard output appended to an ASCII file.
1>>	Redirect standard output appended to a UCS-2 encoded file.
1>>a	Redirect standard output appended to an ASCII file.
>>v	Append standard output to an environment variable, encoded as UCS2.
1>>v	Append standard output to an environment variable, encoded as UCS2.

<b>2&gt;&gt;v</b>	Append standard error to an environment variable, encoded as UCS2.
-------------------	--

The UEFI Shell will redirect standard output to a single file or variable and standard error to a single file or variable. Redirecting both standard output and standard error to different files or variables is allowed, but not to the same file or variable. Redirecting standard output to more than one file or variable on the same command is not supported. Similarly, redirecting to multiple files or variables is not supported for standard error either.

When redirecting output to an environment variable, if a new environment variable will be created, then it will be volatile. If the environment variable already exists and is non-volatile, an error will be generated.

All output to UCS-2 encoded files will be prefixed with the Unicode Byte Ordering Character (0xFFFE) if (a) there is at least one other character output and (b) that character is not the Unicode Byte Ordering Character.

"**NUL**" is used as a special output file name. When "**NUL**" is used, the output will not be written into a file. Instead, they are discarded silently.

### 3.4.4.2 Input Redirection

Contents from an existing file or variable can be used as standard input to a UEFI shell command. Any commands coming from an ASCII file will be automatically be converted to the equivalent UCS-2 encoding and passed to the UEFI shell command.

When redirecting input from an environment variable, the environment variable must already exist. If it does not, an error will be generated. The shell will ensure that the first character read from an input redirected environment variable will contain the Unicode Byte Ordering Character (0xFFFE). If the first character in the input source is not the Unicode Byte Ordering Character, the shell will insert this character in the output from the input redirected variable. This is done to ensure that an input redirected variable will be look like a UCS-2 encoded file.

Redirecting input from a non-volatile variable is permitted.

**Table 5 Input Redirection Syntax**

Character Sequence	Description
<b>&lt;</b>	Redirect standard input from a Unicode file.
<b>&lt;a</b>	Redirect standard input from an ASCII file.
<b>&lt;v</b>	Redirect standard input from an environment variable.

### 3.4.4.3 Pipe Support

By using the | character, a data channel is formed that takes the standard Unicode output of a file and feeds the data as standard input to another program.

The format for this support is as follows:

`UEFI_Shell_Command [options] | UEFI_Shell_Command`

When piping from an environment variable, the environment variable must already exist. If it does not, an error will be generated. Any Unicode Byte Ordering Characters are ignored.

**Table 6 Input Redirection Syntax**

Character Sequence	Description
	Pipe output of a command to another program in UCS-2 format.
a	Pipe output of a command to another program in ASCII format.

When piping to an environment variable, if a new environment variable will be created, then it will be volatile. If the environment variable already exists and is non-volatile, an error will be generated.

All output to UCS-2 encoded files will be prefixed with the Unicode Byte Ordering Character (0xFFFE) if (a) there is at least one other character output and (b) that character is not the Unicode Byte Ordering Character.

## 3.5 Current Directory

For each file system, the UEFI Shell maintains a *current directory*, which is the default directory used if a directory is not specified in a file path. The UEFI Shell maintains a *current working file system*, which is the default file system used if one is not supplied in a file path. The current directory in the current working file system is the *current working directory*.

The current directory for any file system and current working file system can be retrieved using the `GetCurDir()` function (see page 34). The current directory for any file system and the current working file system can be changed using the `SetCurDir()` function (see page 57).

The current directory for any file system and current working directory can be retrieved and changed using the `cd` shell command (see page 95).

The current working directory can be found in the standard `%cwd%` environment variable.

## 3.6 Variables

This section describes the different types of variable substitution that happens on the command-line for environment variables, positional parameters, index parameters and aliases.



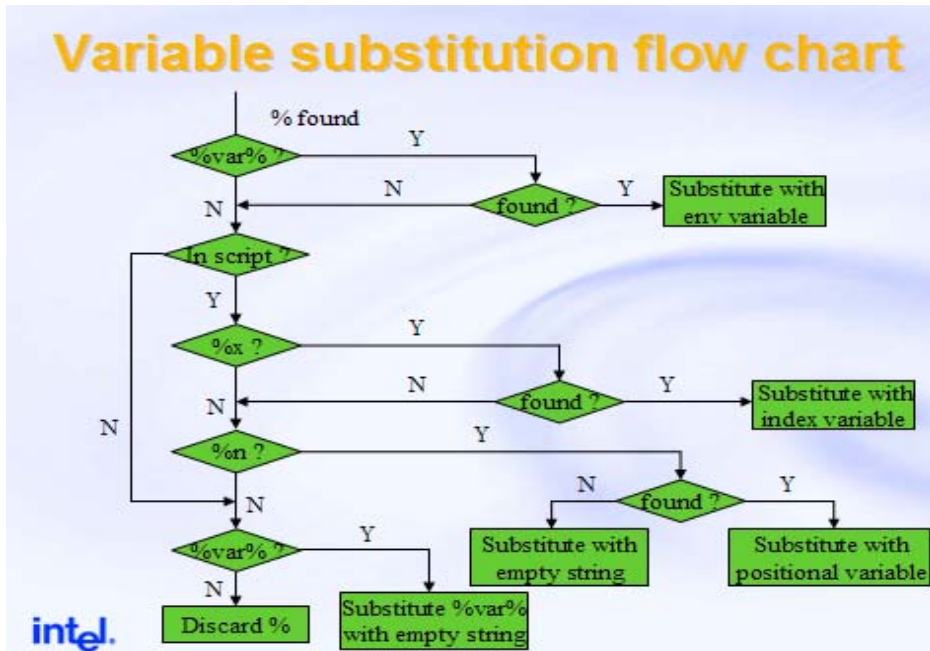


Figure 1 Variable substitution flow chart

### 3.6.1 Environment Variables

Environment variables are variables that can hold the user specified contents and can be used on the command line or in scripts. Each environment variable has a case-sensitive name (a C-style identifier) and a string value. Environment variables can be either volatile (they will lose their value on reset or power-off) or non-volatile (they will maintain their value across reset or power-off).

Environment variables can be used on the command-line by using `%variable-name%` where `variable-name` is the environment variable's name. Variable substitution is not recursive. Environment variables can also be retrieved by a UEFI Shell command by using the `GetEnv()` (see page 39) function.

Environment variables can be displayed or changed using the set shell command. They can also be changed by a UEFI Shell command using the `SetEnv()` function.

The following table lists the environment variables that have special meaning to the UEFI Shell:

**Table 7 Environment Variables with Special Meaning to the UEFI Shell**

Variable	V/NV RO/RW	Description
<b>cwd</b>	V/RO	The current working directory, including the current working file system. See “Current Directory” (page TBD) for more information.
<b>Lasterror</b>	V/RO	Last returned error from a UEFI Shell command or batch script
<b>path</b>	V/RW	<p>The UEFI Shell has a default volatile environment variable path, which contains the default path that UEFI Shell will search if necessary. When user wants to launch an UEFI application, UEFI Shell will first try to search the current directory if it exists, and then search the path list sequentially. If the application is found in one of the paths, it will stop searching and execute that application. If the application is not found in all the paths, UEFI Shell will report the application is not found.</p> <p>If the path variable is empty or it doesn't exist, UEFI Shell will treat current directory as the working directory. In general, paths stored in path variable looks like:</p> <p style="padding-left: 40px;">path: .;fs0:\efi\tools;fs0:\efi\boot;fs0:\;fs1:\efi\tools;fs1:\efi\boot;fs1:\</p> <p>The UEFI Shell supports both absolute paths and relative paths when launching commands. Users can set path to any specified value, but this variable will be refreshed immediately after the execution of command 'map -r' and it's volatile so that the contents will be lost after reset or power off. Typically users can append the paths to this variable in this way:</p> <pre>set -v path %path%;fs0:\test</pre>
<b>profiles</b>	NV/RO	The list of UEFI shell command profiles supported by the shell. Each profile name may only contain alphanumeric characters or the '_' character. Profile names are semicolon (;) delimited.
<b>shellsupport</b>	V/RO	<p>Reflects the current support level enabled by the currently running shell environment (see UEFI Shell Levels of Support, section 3.1). The contents of the variable will reflect the text-based numeric version in the form that looks like:</p> <p style="text-align: center;">3</p> <p>This variable is produced by the shell itself and is intended as read-only, any attempt to modify the contents will be ignored.</p>
<b>uefishellversion</b>	V/RO	<p>Reflects the revision of the UEFI Shell specification that the shell supports. The contents are formatted as text:</p> <p style="text-align: center;">2.00</p>

<b>uefiversion</b>	V/RO	Reflects the revision of the UEFI specification that the underlying firmware supports. The contents will look like this:  2.10
--------------------	------	--

### 3.6.2 Positional Parameters

Positional parameters are the first ten arguments (**%0-%9**) passed from the command line into a UEFI shell script. The first parameter after the UEFI Shell script name becomes **%1**, the second **%2**, the third **%3**, and so on. **%0** is the full path name of the script itself.

The **shift** (see page 162) command can be used delete the contents of **%1** and shift all of the other positional parameters down one place (**%2 -> %1**, **%3 -> %2**, **%4 -> %3**, etc.) There is no way for a UEFI Shell script to access the 10<sup>th</sup> or greater argument without using **shift**.

When executing the UEFI Shell script, the **%n** is replaced by the corresponding argument on the command-line that invoked the script. If a positional parameter is referenced in the UEFI Shell script but that parameter was not present, then an empty string is substituted.

Positional parameters do not have quotation marks removed from them. For more information on how quotation marks are handled, see "Quoting"

### 3.6.3 Index Parameters

Index parameters are the variables created by the **for** command (see page 139) when executing inside of a UEFI Shell script. Each index parameter is in the form of **%x**, where **x** is a case-insensitive letter (**%a-%z**).

When executing the UEFI Shell script, the **%x** is replaced by the value specified by the **for** command. If the specified index parameter has not been defined in the current UEFI Shell script, the script execution will halt with an error.

### 3.6.4 Aliases

An alias creates an additional name for an internal UEFI Shell command or a UEFI Shell application. Aliases can be used to provide alternative commonly used names for existing shell commands or even create shortcuts. An alias is a C-style identifier and may refer to an internal command or else the directory and file name of a UEFI shell application.

During command-line processing, if the 1<sup>st</sup> argument of a command is a defined alias, the shell replaces the alias with its defined value. The alias substitution occurs after

the variable substitution. So if `%myvariable%` is set to `dir` and `dir` is aliased to `ls`, entering `%myvariable%` in command line will cause the `ls` command to be executed. Alias substitution is not recursive.

There are several built-in aliases (sometimes referred to as *synonyms*) provided by the UEFI Shell for the following commands:

**Table 8 Built-in Aliases for the UEFI Shell**

Original Command	Built-In Alias	Description
<code>ls</code>	<code>dir</code>	List directory contents
<code>rm</code>	<code>del</code>	Delete a file
<code>cp</code>	<code>copy</code>	Copy a file.
<code>mkdir</code>	<code>md</code>	Create a directory
<code>dmem</code>	<code>mem</code>	Display memory

## 3.7 File Names

The UEFI Shell supports file names and paths with the following format:

`fs-path` := [`fs-map-name`] [`fs-divider`][`fs-dirs`][`fs-name`]

`fs-map-name` := identifier :

`fs-divider` := \ | /

`fs-dirs` := `fs-dir` |  
`fs-dirs fs-dir`

`fs-dir` := `fs-name fs-divider`

`fs-name` := `fs-file-name .fs-file-ext`

`fs-file-name` := one or more ASCII characters, excluding \*?<>\/"⊕

Both short and long names are supported. The maximum valid length for a file path is 255 characters.

### 3.7.1 Wildcard Expansion

The `*`, `?` and `[]` characters can be used as wildcard characters in file name command-line options certain UEFI shell commands that use the `OpenFileList()` function. In addition, the UEFI Shell `for` and `if` script commands also expand arguments containing wildcard characters to existing file names that matches the pattern. A `^` before the wildcard cannot prevent the wildcard from being expanded.

[ ] can be either wildcard characters or literal file name characters, the UEFI Shell will try to take them as wildcard characters first to match files, if there are files matched, no further interpretation. Otherwise, they will be considered as literal characters in file names.

**Table 9 Wildcard Character Expansion**

Character Sequence	Description
*	Matches zero or more characters in a file name.
?	Matches exactly one character in a file name.
[ ]	Matches one character in a file name with one of the characters in [ ]

### 3.7.2 Mappings

Mappings are C-style identifiers that act as an alias for a device path. These mappings can be used interchangeably with the device path in the **EFI\_SHELL\_PROTOCOL** and on the interactive shell's command line. Default mappings (such as fsx) are created by the UEFI Shell during initialization (see Initialization, section 3.3). Other mappings can be created using the **map** shell command or the **SetMap()** function.

A *mapping* which translates to a device path of a device that has a file system protocol installed on its handle is called a *file system mapping*.

### 3.7.3 Consistent File System Mapping

The UEFI Shell provides consistent mapping for file system mappings. The consistent mappings will not change after reboot or after 'map -r' if the hardware configuration hasn't changed. If two or more computers have the same hardware configurations, the consistent mapping results on these computers should be exactly the same. Hardware configuration changes are defined as the changes of controllers or physical interfaces to which the devices are connected. If you are used to the **fsx** notation style for mapping file systems, then the new consistent mapping convention might look a little different. For example, the GUIDed file system may have a consistent mapping, such as **f0agonennapphibbndlmeaakamjeafdnb**. The **fsx** style mappings facilitates the use of mappings on the command line, but they don't have the consistent mapping characteristics.

Consistent mapping only applies to file system mappings, not other device mappings.

## 3.8 Scripts

The UEFI Shell has the capability of executing commands from a file (script). UEFI Shell script files are named using the **".nsh"** extension. Script files can be either UCS-2 or ASCII format files. UEFI Shell script files are invoked by entering the filename at the command prompt, with or without the filename extension. See "Scripts" (section 4) for more information.

## 3.9 Nesting the Shell

The UEFI Shell supports nested shell execution. The UEFI Shell can run the shell from within itself. The maximum nesting level is determined by how much memory the system has. The command `exit` can be used to exit the current shell instance. If the current shell is a child shell, it will return to the parent shell. Newly launched shell will have a brand new execution environment except for environment variables and aliases.

## 3.10 Interactive Features

Even though the shell design specification primarily discusses aspects of the shell that can run without user interaction, there are some features described which can augment the experience of users that are actively interacting with the console.

### 3.10.1 Key History Support

The UEFI Shell will save commands history that executed from the shell prompt. User can press up-arrow key and down-arrow key to browse the previous commands. Commands that were executed in script will not be saved in the key history.

### 3.10.2 Execution Interrupt Support

The UEFI Shell supports the capability of interrupting the execution of the shell commands and scripts.

There are two kinds of the execution interrupt: script execution interrupt and internal command execution interrupt.

#### **Script Execution Interrupt.**

The user can press CTRL-C to interrupt the execution of the script. The interrupt can only happen between commands in a script. The script supports the nesting of scripts. Once a script is interrupted, all its predecessor scripts are also interrupted.

#### **Shell Command Execution Interrupt.**

The user can press the CTRL-C key sequence to interrupt the execution of some time-consuming UEFI Shell commands (i.e. `ls -r`). The UEFI Shell detects this key sequence and signals the `ExecutionBreak` member of the `EFI_SHELL_PROTOCOL`. Individual UEFI Shell commands check the state of `ExecutionBreak` as often as practical and return the `SHELL_ABORTED` error code.

The UEFI Shell may not support asynchronous execution interrupt for commands or applications.

### 3.10.3 Output Streaming Control

The UEFI Shell supports the ability to pause and resume the streaming of characters to the output device. The user can press the CTRL-S key sequence to pause and any key to resume the output results produced by the current running commands or applications. It is especially useful for the commands and applications that may produce a great deal of the output results.

### 3.10.4 Scroll Back Buffer Support

The UEFI Shell supports the ability to scroll back and forward the output so that consoles can have screen history. The user can press Page Up and Page Down to scroll back and forward the screen history, and press any other key to quit scrolling. However, the user cannot do this while the command, application or script is being executed. The text output history will be at least 3 screens.

## 3.11 Shell Applications

UEFI Shell applications have the extension `.efi` and have the same entry point (`EFI_IMAGE_ENTRY_POINT`) defined in section 4.1 of the UEFI specification. When the entry point to a shell command is called, the image handle (`ImageHandle`) has both the `EFI_LOADED_IMAGE_PROTOCOL` and the `EFI_SHELL_PARAMETERS_PROTOCOL` installed on it.

In the `EFI_LOADED_IMAGE_PROTOCOL`, the `LoadOptions` member points to the NULL-terminated, expanded command line. The first part (which corresponds to `Argv[0]` in the `EFI_SHELL_PARAMETERS_PROTOCOL`) is the file path of the executable after alias substitution.

After this, delimited by a whitespace character, are listed each of the arguments, with all environment variables expanded, and with quotation marks still present. This is different from what appears in `Argv[]` in the `EFI_SHELL_PARAMETERS_PROTOCOL`.

The `EFI_SHELL_PARAMETERS_PROTOCOL` has two members: `Argv`, which points to each of the command-line arguments and `Argc`, which is the number of command-line arguments. There is always at least one command-line argument: the path and file name of the shell command. Any arguments are enumerated in `Argv[1-n]`, with all environment variables expanded and all quotation marks removed.

If `ExecutionBreak` was signaled during the execution of a UEFI Shell application, then it will be cleared upon return to the shell.

### 3.11.1 Installation

During installation, UEFI Shell applications may choose to update certain global files or settings, which are used for detecting installed UEFI Shell profiles or providing help text for UEFI shell applications.

During installation, UEFI Shell applications may update the `profiles` environment variable, which lists all of the command profiles supported by the current

implementation of the shell. Some of these command profiles are standard (see “Shell Command Profiles”, section 5.2) and others can be defined by implementers.

During installation, UEFI Shell applications may provide a help file (as described in “Command-Line Help”, section 3.11.2) to support the standard **help** command.

During installation, UEFI Shell applications may update a help file for the help category/categories to which the application belongs. This consists of creating a **NAME** section for the `<category>.man` file and then copying the **NAME** section from the command’s help file to the end of the **DESCRIPTION** section of the `<command>.man` file, if not already present.

The shell application’s category (or categories) is listed in the **CATEGORY** section of the shell application’s help (.man) file. Help categories are not described as part of this specification.

### 3.11.2 Command-Line Help

The user can get UEFI Shell application command-line either using the **help** command or else by typing in `<command-name> -?`. Both of these use the `GetHelpText()` (page 32) function to retrieve the help text.

The UEFI Shell gets help text for UEFI Shell applications by search the directory where **startup.nsh** was located (highest priority) (if **-nostartup** was not specified) and then the directories specified by the **path** environment variable for a file with the same name as the UEFI Shell application, but with the **.man** extension. The format of these files is described in Appendix B.

The UEFI Shell supports help categories, which have **.man** pages similar to those for UEFI shell applications, except using the category name instead of the application name.



## 4 Scripts

---

UEFI Shell scripts allow user to simplify routine or repetitive tasks. A shell script program is a UCS-2 or ASCII text file that contains one or more commands and has a `.nsh` file name extension. When the file name is typed at the command prompt, commands in the file are executed sequentially.

All shell commands can be executed in scripts. In addition, some script-only commands are also provided to do basic flow control. Script-only means those commands can be only executed in UEFI Shell Script files, and cannot be executed from the shell prompt.

Up to ten positional arguments are supported for scripts. Positional argument substitution is performed before the execution of each line in the script file. Positional arguments are denoted by `%n`, where `n` is a digit between `0` and `9`. By convention, `%0` is the name of the script file currently being executed.

Script file execution can be nested; that is, script files may be executed from within other script files. Recursion is allowed. Shell scripts run in their parent's environment.

Output and input redirection are fully supported in scripts. Output redirection on a command in a script file causes the output for that command to be redirected. Output redirection on the invocation of a script causes the output for all commands executed from that script to be redirected to the file, with the output of each command appended to the end of the file.

By default, both the input and output for all commands executed from a script are echoed to the console. Display of commands read from a script file can be suppressed via the `echo -off` command (see `echo`). Also, additional `@` before a command in a script file can prevent the current command from being echoed.

If output for a command is redirected to a file, then that output is not displayed on the console. Commands executed from a script will not be saved by the shell for key history and these commands cannot be recalled by pressing Up-Arrow key.

### 4.1 Comments

Comments can be embedded in scripts. The `#` character on a line is used to denote that all characters on the same line and to the right of the `#` are to be ignored by the shell. Whether the echo state is on or off, comments are not echoed to the console.

### 4.2 Error Handling

By default, if an error is encountered during the execution of a command in a script, the script will continue to execute. But if an error is encountered when executing the script-only commands which affects the logic of the script, such as `for`, `endfor`, `if`,

*else*, *endif*, and *goto*, the script will exit. If the error arousing script is a called by another script, the caller script will continue to execute.

The `lasterror` shell variable allows scripts to test the results of the most recently executed command using the *if* command. This variable is maintained by the shell, is read-only, and cannot be modified by command *set*.

Script-only commands, as well as *echo*, which are used to control the logic of the script, do not affect the value of variable `lasterror`. The `lasterror` environment variable is not changed by comments.

## 4.3 Script Nesting

Scripts can be nested. A script can call one or more scripts. The embedded script will be executed as a command. After the whole embedded script is executed completely, the next command will be executed.

The UEFI Shell will automatically save and restore the running mode before and after the execution of nested scripts so that the changes of running modes in nested scripts will not affect the running mode of a parent script.

## 4.4 Output and Echoing

Output redirection is supported for scripts. Output redirection on a command in a script file causes the output for that command to be redirected. Output redirection on the invocation of a script causes the output for all commands executed from that script to be redirected to the file, with the output of each command appended to the end of the file. If a command in a script redirects its output to file1, while the output is redirected to file2 on invocation of a whole script, the output of that command will be redirected to file1, but the echo of the command itself (if echo state is on) will appear in file2, as well as output of all other commands.

When a script is launched from the interactive shell, it inherits its echo state from interactive shell or parent script. Changing echo state in the script does not affect the echo state of the interactive shell. When a script calls another script, the called script inherits the caller script's current echo state. But if the called script changes the echo state, after returning, the caller's echo script changes, too.

## 4.5 Limitations

Following are some examples of known limitations with the UEFI scripts:

1. Cannot read and write the same file within a single command, for example,  
`fs0:>type test.nsh > test.nsh`
2. `goto` cannot be used to jump into another loop.
3. Don't use the same index variable in nested `for` statements.
4. Index values cannot be referred outside of the `for` statement that defines it.

Formatted: Bullets and Numbering

## 5 *Shell Commands*

---

### 5.1 **Overview**

This section describes the standard UEFI Shell commands.

The table below lists all standard UEFI Shell commands.

**Table 10 Commands from Default Build Shell**

Command	Description	Required at Shell Level or Profile
<b>alias</b>	Displays, creates, or deletes aliases in the UEFI Shell environment	3
<b>attrib</b>	Displays or changes the attributes of files or directories.	2
<b>bcfg</b>	Manipulate boot order and driver order	Debug1, Install1
<b>cd</b>	Displays or changes the current directory	2
<b>cls</b>	Clears the standard output and optionally changes the background color	3
<b>comp</b>	Compares the contents of two files on a byte for byte basis	Debug1
<b>connect</b>	Binds a driver to a specific device and starts the driver.	Driver1
<b>cp</b>	Copies one or more source files or directories to a destination.	2
<b>date</b>	Displays and sets the current date for the system.	2/3
<b>dblk</b>	Displays the contents of one or more blocks from a block device.	Debug1
<b>del</b>	Deletes one or more files or directories.	2
<b>devices</b>	Displays the list of devices managed by UEFI drivers.	Driver1
<b>devtree</b>	Displays the tree of devices compliant with the UEFI Driver Model.	Driver1
<b>dh</b>	Displays the device handles in the UEFI environment.	Driver1
<b>dir</b>	Lists directory contents or file information.	2
<b>disconnect</b>	Disconnects one or more drivers from the specified devices.	Driver1
<b>dmem</b>	Displays the contents of system or device memory.	Debug1
<b>dmpstore</b>	Manages all UEFI NVRAM variables.	Debug1
<b>drivers</b>	Displays a list of information for drivers that follow the EFI Driver Model in the EFI environment.	Driver1
<b>drvcfg</b>	Configures the driver using the UEFI Configuration Access Protocol.	Driver1
<b>drvdiag</b>	Invokes the Driver Diagnostics Protocol.	Driver1
<b>echo</b>	Controls whether or not script commands are displayed as they are read from the script file and prints the given message to the display.	3

Command	Description	Required at Shell Level or Profile
<b>alias</b>	Displays, creates, or deletes aliases in the UEFI Shell environment	3
<b>attrib</b>	Displays or changes the attributes of files or directories.	2
<b>edit</b>	Full screen editor for ASCII or UCS-2 files.	Debug1
<b>eficompress</b>	Compress a file using EFI Compression Algorithm.	Debug1
<b>efidecompress</b>	Decompress a file using EFI Decompression Algorithm.	Debug1
<b>else</b>	Conditionally execute commands if a previous <b>if</b> condition was false.	1
<b>endfor</b>	End a loop stated with <b>for</b> in a script.	1
<b>endif</b>	End a conditional block started with <b>if</b> .	1
<b>exit</b>	Exits the UEFI Shell environment and returns control to the parent that launched the UEFI Shell.	1
<b>for</b>	Start a loop in a script	1
<b>getmtc</b>	Return current monotonic count.	3
<b>goto</b>	Go to a label in a script	1
<b>guid</b>	Displays all registered EFI GUIDs.	Debug1
<b>help</b>	Displays the list of commands that are built into the UEFI Shell.	3
<b>hexedit</b>	Full screen hex editor for files, block devices, or memory.	Debug1
<b>if</b>	Conditionally execute script statements.	1
<b>ifconfig</b>	Displays or modifies the current IP configuration.	Network1
<b>ipconfig</b>	Displays or modifies the current IP configuration.	Network1
<b>load</b>	Loads a UEFI driver into memory.	2
<b>loadpcirom</b>	Loads a PCI Option ROM from the specified file.	Debug1
<b>ls</b>	Lists a directory's contents or file information.	2
<b>map</b>	Defines a mapping between a user-defined name and a device handle.	2
<b>mem</b>	Displays the contents of system or device memory.	Debug1
<b>memmap</b>	Displays the memory map maintained by the EFI environment.	Debug1

Command	Description	Required at Shell Level or Profile
<b>alias</b>	Displays, creates, or deletes aliases in the UEFI Shell environment	3
<b>attrib</b>	Displays or changes the attributes of files or directories.	2
<b>mkdir</b>	Creates one or more new directories.	2
<b>mm</b>	Displays or modifies MEM/MMIO/IO/PCI/PCIE address space.	Debug1
<b>mode</b>	Displays or changes the console output device mode.	Debug1
<b>mv</b>	Moves one or more files to a destination within a file system.	2
<b>openinfo</b>	Displays the protocols and agents associated with a handle.	Driver1
<b>parse</b>	Parse data returned from standard formatted output	2
<b>pause</b>	Pause script execution and wait for a keypress	3
<b>pci</b>	Displays PCI device list or PCI function configuration space.	Debug1
<b>ping</b>	Check response of an ip address.	Network1
<b>reconnect</b>	Reconnects drivers to the specific device.	Driver1
<b>reset</b>	Resets the system.	2
<b>rm</b>	Deletes one or more files or directories.	2
<b>sermode</b>	Sets serial port attributes.	Debug1
<b>set</b>	Used to maintain the environment variables that are available from the EFI environment.	2
<b>setsize</b>	Set the size of a file	Debug1
<b>setvar</b>	Change value of UEFI variable	Debug1
<b>shift</b>	Shift to the 2 <sup>nd</sup> set of positional parameters	1
<b>smbiosview</b>	Displays SMBIOS information.	Debug1
<b>time</b>	Displays or sets the current time for the system.	2/3
<b>timezone</b>	Displays or sets time zone information.	2/3
<b>touch</b>	Updates the time and date on a file to the current time and date.	3
<b>type</b>	Sends the contents of a file to the standard output device.	3

Command	Description	Required at Shell Level or Profile
<b>alias</b>	Displays, creates, or deletes aliases in the UEFI Shell environment	3
<b>attrib</b>	Displays or changes the attributes of files or directories.	2
<b>unload</b>	Unloads a driver image that was already loaded.	Driver1
<b>ver</b>	Displays the version information for this EFI firmware.	3

### 5.1.1 Explanation of Command Description Layout

The description of each command is composed of four sections: **Summary**, **Usage**, **Options**, **Description**, **Notes**, **Status Codes Returned** and **Examples**.

**Summary** is a brief explanation of the function of the command. **Usage** describes how the command is used. **Options** gives a complete description of each of the command-line options. **Description** describes the details of the command.

Examples give sample usage of the command. The output may differ from the output listed in this section.

### 5.1.2 Shell Command-Line Options

The following table describes the standard command-line options. No command supports all options, but when needed, the following option parameters are used:

**Table 11 Standard Command Line Options**

Option	Description
<b>-b, -break</b>	Pause after each page.
<b>-q, -quiet</b>	The command will suppress all output.
<b>-sfo</b>	Standard Format Output. Instead of normal output, the shell command will output using the standard format described Appendix D.
<b>-t, -terse</b>	Terse Output. The shell command will restrict additional informative content.
<b>-v, -verbose</b>	Verbose Output. The shell command will output additional informative content.
<b>-?</b>	Help. For more information on how command-line help is supported, see section 3.11.2.

Command-line options that begin with the '\_' character are implementation-specific.

## 5.2 Shell Command Profiles

Shell command profiles are groups of shell commands that are identified by a profile name. Profile names that begin with the '\_' character are reserved for individual implementations. For information on how profiles are identified, see section 3.3.2 (Supported Profiles).

For more information on how profiles are updated when new commands are installed, see section 3.11

There are four standard profiles:

**Table 12**Standard Profiles

Profile Name	Description
<b>Driver1</b>	Standard set of driver-related commands.
<b>Debug1</b>	Standard set of debug commands.
<b>Network1</b>	Standard set of networking-related commands.
<b>Install1</b>	Standard set of commands to aid installation.

## 5.3 Shell Commands

### alias

#### Summary

Displays, creates, or deletes aliases in the UEFI Shell environment.

#### Usage

```
alias [-d|-v] [alias-name] [command-name]
```

#### Options

*alias-name*

Alias name

*command-name*

Original command's name or original command's file name/directory.

**-d**

Delete an alias. **command-name** should not be present.

**-v**

Make the alias volatile.



## Description

This command displays, creates, or deletes aliases in the UEFI Shell environment. An alias provides a new name for an existing UEFI Shell command or UEFI application. Once the alias is created, it can be used to run the command or launch the UEFI application.

There are some aliases that are predefined in the UEFI Shell environment. These aliases provide the MS-DOS and UNIX equivalent names for the file manipulation commands. See Built-In Aliases (section 3.6.4) for more details.

Aliases will be retained even after exiting the shell unless the `-v` option is specified. If `-v` is specified then the alias will not be valid after leaving the shell.

## Status Codes Returned

<code>SHELL_SUCCESS</code>	The action was completed as requested.
<code>SHELL_INVALID_PARAMETER</code>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<code>SHELL_OUT_OF_RESOURCES</code>	A request to set a variable in a non-volatile fashion could not be completed. The resulting non-volatile request has been converted into a volatile request.
<code>SHELL_SECURITY_VIOLATION</code>	This function was not performed due to a security violation

## Examples

To display all aliases in the UEFI Shell environment:

```
Shell> alias
md  : mkdir
rd  : rm
```

To create an alias in the UEFI Shell environment:

```
Shell> alias myguid guid
Shell> alias
md    : mkdir
rd    : rm
myguid : guid
```

To delete an alias in the UEFI Shell environment:

```
Shell> alias -d myguid
Shell> alias
md    : mkdir
rd    : rm
```

To add a volatile alias in the current EFI environment, which has a star \* at the line head. This volatile alias will disappear at next boot.

```
Shell> alias -v fs0 floppy
Shell> alias
md    : mkdir
rd    : rm
* fs0 : floppy
```

## attrib

### Summary

Displays or changes the attributes of files or directories.

### Usage

```
attrib [+a|-a] [+s|-s] [+h|-h] [+r|-r] [file...] [directory...]
```

### Options

**+a|-a**

Set or clear the 'archive' attribute

**+s|-s**

Set or clear the 'system' attribute

**+h|-h**

Set or clear the 'hidden' attribute

**+r|-r**

Set or clear the 'read-only' attribute

**file**

File name (wild cards are permitted)

**directory**

Directory name (wildcards are permitted)

### Description

This command displays and sets the attributes of files or directories. The following four attribute types are supported in the UEFI file system:

- Archive [A]
- System [S]
- Hidden [H]
- Read only [R]

If a file (in general meaning) is a directory, then it is also shown to have the attribute [D].

If any file in the file list that is specified in the command line does not exist, **attrib** will continue processing the remaining files while reporting the error.

If no file or directory is specified, then all of the files in the current directory are displayed.

If no attribute is specified, then the attributes of the files will be displayed.

## Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_NOT_FOUND</b>	The requested file was not found.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_WRITE_PROTECTED</b>	The media that the action was to take place on is write-protected.

## Examples

- To display the attributes of a directory:  

```
fs0:\> attrib fs0:\
attrib:D      fs0:\
```
- To display the attributes of all files and sub-directories in the current directory:  

```
fs0:\> attrib *
attrib: AS   fs0:\serial.efi
attrib:DA   fs0:\test1
attrib: A HR fs0:\bios.inf
attrib: A    fs0:\VerboseHelp.txt
attrib: AS   fs0:\IsaBus.efi
```
- To add the system attribute to all files with extension '.efi':  

```
fs0:\> attrib +s *.efi
```
- To remove the read only attribute from all files with extension '.inf':  

```
fs0:\> attrib -r *.inf
attrib: A H  fs0:\bios.inf
```

## bcfg

### Summary

Manages the boot and driver options that are stored in NVRAM.

### Usage

```
bcfg driver|boot [dump [-v]] [add # file "desc"] [addp # file "desc"]  
[addh # handle "desc"] [rm #] [mv # #] [-opt # [[filename]|"data"]] |  
[KeyData <ScanCode UnicodeChar>*]
```

### Options

#### driver

Display/modify the driver option list

#### boot

Display/modify the boot option list

#### dump

Display the option list

#### -v

Display the option list with extra info including the optional data.

#### add

Add an option. The # is the number of options to add in hexadecimal. The file name of the UEFI application/driver for the option. The quoted parameter is the description of the option being added.

#### addh

Add an option that refers to the driver specified by *handle*. The # is the number of options to add, in hexadecimal. The *handle* is the driver handle, in hexadecimal. The device path for the option is retrieved from the handle. The quoted parameter is the description of the option being added.

#### addp

Add an option that refers to a specific file. Only the portion of the device path starting with the hard drive partition is placed in the option. The # is the number of options to add, in hexadecimal. The quoted parameter is the description of the option being added.

#### rm

Remove an option. The parameter lists the number of the options to remove in hexadecimal.

#### mv

Move an option. The first numeric parameter is the number of the option to move in hexadecimal. The second numeric parameter is the new number of the option being moved.

#### -opt

Display/modify the optional data associated with a driver or boot option. Followed either by the file name of the file which contains the binary data to be

associated with the driver or boot option optional data or else the quote-delimited data which will be associated with the driver or boot option optional data.

**KeyData**

The packed value associated with a hot-key. This is the equivalent of the EFI\_KEY\_DATA value in the UEFI specification.

**ScanCode**

This is the UEFI-defined Scan code portion of the EFI\_INPUT\_KEY structure. This value is directly associated with the preceding KeyData value and there may be 1 to 4 entries per the UEFI specification. When one instance of this parameter has a non-zero value, the paired UnicodeChar value will have a zero-based value.

**UnicodeChar**

This is the Unicode value for the character associated with the preceding KeyData value. There may be 1 to 4 entries per the UEFI specification. When one instance of this parameter has a non-zero value, the paired ScanCode value will have a zero-based value.

## Description

Manages the boot and driver options stored in NVRAM. This command can display the **Boot####** or **Driver####** environment variables by using the dump option. The add option can be used to add a new **Boot####** or **Driver####** environment variable. The rm option can be used to delete a **Boot####** or **Driver####** environment variable, and finally, then mv option can be used to reorder the **Boot####** and **Driver####** environment variables. The add, rm, and mv options also update the **BootOrder** or **DriverOrder** environment variables as appropriate.

## Examples

To display driver options:

```
Shell> bcfg driver dump
```

To display boot options:

```
Shell> bcfg boot dump
```

To display verbosely of boot options:

```
Shell> bcfg boot dump -v
```

To add a driver option #5

```
Shell> bcfg driver add 5 mydriver.efi "My Driver"
```

To add a boot option #3

```
Shell> bcfg boot add 3 osloader.efi "My OS"
```

To remove boot option #3

```
Shell> bcfg boot rm 3
```

To move boot option #3 to boot option #7

```
Shell> bcfg boot mv 3 7
```

To assign a CTRL-B hot-key to boot option #3.

```
Shell> bcfg boot -opt 3 0x40000200 0 0x42
```

## Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_NOT_FOUND</b>	The requested option was not found.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<b>SHELL_UNSUPPORTED</b>	The action as requested was unsupported.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_OUT_OF_RESOURCES</b>	There was insufficient free space for the request to be completed.

## cd

### Summary

Displays or changes the current directory.

### Usage

```
cd [path]
```

### Options

*path*

The relative or absolute directory path.

### Description

This command changes the current working directory that is used by the UEFI Shell environment. If a file system mapping is specified, then the current working directory is changed for that device. Otherwise, the current working directory is changed for the current device.

If *path* is not present, then the current working directory (including file system mapping) is displayed to standard out.

The table below describes the conventions that are used to refer to the directory, its parent, and the root directory in the UEFI Shell environment.

**Table 13 Conventions for Directory Names**

Convention	Description
.	Refers to the current directory.
..	Refers to the directory's parent.
\	Refers to the root of the current file system.

The current working directory is maintained in the environment variable `%cwd%`. See "Current Directory" (section 3.5) for more information.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.



## Examples

To change the current filesystem to the mapped fs0 filesystem:

```
Shell> fs0:
```

To change the current directory to subdirectory 'efi':

```
fs0:\> cd efi
```

To change the current directory to the parent directory (fs0:\):

```
fs0:\efi> cd ..
```

To change the current directory to 'fs0:\efi\tools':

```
fs0:\> cd efi\tools
```

To change the current directory to the root of the current fs (fs0):

```
fs0:\efi\tools> cd \  
fs0:\>
```

To change volumes with cd will not work! For example:

```
fs0:\efi\tools> cd fs1:\ !!!! will not work !!!!  
must first type fs1: then cd to desired directory
```

To move between volumes and maintain the current path.

```
fs0:\> cd \efi\tools  
fs0:\efi\tools> fs1:  
fs1:\> cd tmp  
fs1:\tmp> cp fs0:.* .  
copies all of files in fs0:\efi\tools into fs1:\tmp directory
```

## cls

### Summary

Clears the standard output and optionally changes the background color.

### Usage

```
cls [color]
```

### Options

```
color    - New background color
0        - Black
1        - Blue
2        - Green
3        - Cyan
4        - Red
5        - Magenta
6        - Yellow
7        - Light gray
```

### Description

This command clears the standard output device with an optional background color attribute. If `color` is not specified, then the background color does not change.

### Status Codes Returned

<code>SHELL_SUCCESS</code>	The action was completed as requested.
<code>SHELL_INVALID_PARAMETER</code>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<code>SHELL_SECURITY_VIOLATION</code>	This function was not performed due to a security violation
<code>SHELL_NOT_FOUND</code>	The requested file was not found.

### Examples

To clear standard output without changing the background color:

```
fs0:\> cls
```

To clear standard output and change the background color to cyan:

```
fs0:\> cls 3
```

To clear standard output and change the background to the default color:

```
fs0:\> cls 0
```

## comp

### Summary

Compares the contents of two files on a byte for byte basis.

### Usage

```
comp [-b] file1 file2
```

### Options

```
-b      - Display one screen at a time  
file1  - First file name (directory name or wildcards not  
         permitted)  
file2  - Second file name (directory name or wildcards not  
         permitted)
```

### Description

This command compares the contents of two files in binary mode. It displays up to 10 differences between the two files. For each difference, up to 32 bytes from the location where the difference starts is dumped. It will exit immediately if the lengths of the compared files are different.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The function operated as expected.
<b>SHELL_NOT_EQUAL</b>	The files were not identical.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_NOT_FOUND</b>	The requested file was not found.

### Examples

To compare two files with different lengths:

```
fs0:\> comp bios.inf legacy.inf  
Compare fs0:\bios.inf to fs0:\legacy.inf  
Difference #1: File sizes mismatch  
[difference(s) encountered]
```

To compare two files with the same contents:

```
fs0:\> comp bios.inf rafter.inf  
Compare fs0:\bios.inf to fs0:\rafter.inf  
[no difference encountered]
```

To compare two files with the same length but different contents:

```
fs0:\> comp bios.inf bios2.inf
Compare fs0:\bios.inf to fs0:\bios2.inf
Difference #1:
File1: fs0:\bios.inf
00000000: 5F          * *
File2: fs0:\bios2.inf
00000000: 33          *3*
Difference #2:
File1: fs0:\bios.inf
0000000C: 00 00 00 00    *...*
File2: fs0:\bios2.inf
0000000C: 25 32 03 03    *%2..*
[difference(s) encountered]
```

## connect

### Summary

Binds a driver to a specific device and starts the driver.

### Usage

```
connect [[DeviceHandle] [DriverHandle] | [-c] | [-r]]
```

### Options

**-r**

Recursively scan all handles and check to see if any loaded or embedded driver can match the specified device. If so, the driver will be bound to the device. Additionally, if more device handles are created during the binding, these handles will also be checked to see if a matching driver can bind to these devices as well. The process is repeated until no more drivers are able to connect to any devices. However, without the option, the newly created device handles will not be further bound to any drivers.

**-c**

Connect console devices found in the EFI global variables (see UEFI specification, chapter 3)

*DeviceHandle*

Device handle (a hexadecimal number). If not specified, then all device handles will be connected.

*DriverHandle*

Driver handle (a hexadecimal number). If not specified, then all matching drivers will be bound to the specified device. If specified, then this driver will have the highest priority.

### Description

This command binds a driver to a specific device and starts the driver. If the **-r** flag is used, then the connection is done recursively until no further connections between devices and drivers are made. If the **-c** flag is used, then the **connect** command will bind the proper drivers to the console devices that are described in the EFI environment variables. The example below shows the typical output from the verbose help for this command.

If only a single handle is specified and the handle has an **EFI\_DRIVER\_BINDING\_PROTOCOL** on it, then the handle is assumed to be a driver handle. Otherwise, it is assumed to be a device handle.

If no parameters are specified, then the command will attempt to bind all proper drivers to all devices without recursion. Each connection status will be displayed.

## Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

## Examples

- To connect all drivers to all devices recursively:  
`Shell> connect -r`
- To display all connections:  
`Shell> connect`  
`ConnectController(1) : Status = Success`  
`ConnectController(2) : Status = Success`  
`ConnectController(3) : Status = Success`  
`...`  
`ConnectController(3D) : Status = Success`
- To connect drivers with 0x17 as highest priority to all the devices they can manage:  
`Shell> connect 17`
- To connect all possible drivers to device 0x19:  
`Shell> connect 19`
- To connect drivers with 0x17 as highest priority to device 0x19 they can manage:  
`Shell> connect 19 17`
- To connect console devices described in the UEFI Shell environment variables:  
`Shell> connect -c`

## cp

### Summary

Copies one or more source files or directories to a destination.

### Usage

```
cp [-r] [-q] src [src...] [dst]
```

### Options

**src**

Source file/directory name (wildcards are permitted)

**dst**

Destination file/directory name (wildcards are not permitted). If not specified, then the current working directory is assumed to be the destination. If there are more than one directory specified, then the last is always assumed to be the destination.

**-r**

Recursive copy.

**-q**

Quiet copy (no prompt)

### Description

This command copies one or more source files or directories to a destination. If the source is a directory, the **-r** flag must be specified. If **-r** is specified, then the source directory will be recursively copied to the destination (which means that all subdirectories will be copied). If a destination is not specified, then the current working directory is assumed to be the destination.

If any target file (not directory) already exists, there will be a prompt asking the user to confirm replacing the file. The following four choices are available:

- Yes: Replace the file.
- No: Do not replace the file.
- All: Replace the existing files in all subsequent cases.
- Cancel: Do not replace any existing files in all subsequent cases.

If there are multiple source files/directories, the destination must be a directory.

If an error occurs, then the copying process will stop immediately.

When executing in a script, the default is **-q**.

When copying to another directory, the directory must already exist.

## Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<b>SHELL_OUT_OF_RESOURCES</b>	There was insufficient space to save the requested file at the destination.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_WRITE_PROTECTED</b>	An attempt was made to create a file on media that was write-protected.

## Examples

Note:

1. '-r' must be specified if src is a directory. If '-r' is specified, then the source directory will be recursively copied to the destination. Src itself will be copied. ????

- To display the contents of current directory first of all:

```
fs0:\> ls
Directory of: fs0:\

    06/18/01  01:02p <DIR>           512  efi
    06/18/01  01:02p <DIR>           512  test1
    06/18/01  01:02p <DIR>           512  test2
    06/13/01  10:00a                28,739  IsaBus.efi
    06/13/01  10:00a                32,838  IsaSerial.efi
    06/18/01  08:04p                 29  temp.txt
    06/18/01  08:05p <DIR>           512  test
          3 File(s)        61,606 bytes
          4 Dir(s)
```

- To copy a file in the same directory, but change the file name:

```
fs0:\> cp temp.txt readme.txt
copying fs0:\temp.txt -> fs0:\readme.txt
- [ok]
```

- To copy multiple files to another directory:

```
fs0:\> cp temp.txt isaBus.efi \test
copying fs0:\temp.txt -> fs0:\test\temp.txt
- [ok]
copying fs0:\isaBus.efi -> fs0:\test\IsaBus.efi
- [ok]
```

- To copy multiple directories recursively to another directory:



```
fs0:\> cp -r test1 test2 boot \test
copying fs0:\test1 -> fs0:\test\test1
copying fs0:\test1\test1.txt -> fs0:\test\test1\test1.txt
- [ok]
copying fs0:\test2 -> fs0:\test\test2
copying fs0:\test2\test2.txt -> fs0:\test\test2\test2.txt
- [ok]
copying fs0:\boot -> fs0:\test\boot
copying fs0:\boot\shell.efi -> fs0:\test\boot\shell.efi
- [ok]
```

- To see the results of the above operations:

```
fs0:\> ls \test
Directory of: fs0:\test

06/18/01  01:01p <DIR>          512  .
06/18/01  01:01p <DIR>           0  ..
01/28/01  08:21p <DIR>          512  test1
01/28/01  08:21p <DIR>          512  test2
01/28/01  08:21p <DIR>          512  boot
01/28/01  08:23p                29  temp.txt
01/28/01  08:23p                28,739 IsaBus.efi
          2 File(s)          28,828 bytes
          5 Dir(s)
```

Shell>

## date

### Summary

Displays and sets the current date for the system.

### Usage

```
date [mm/dd/[yy]yy] [-sfo]
```

### Options

*mm*

Month of the date to be set (1-12)

*dd*

Day of the date to be set (1-31)

*yy/yyyy*

Year of the date to be set. If only two digits, then 9x = 199x, otherwise 20xx.

**-sfo**

Standard-format output. See "Related Definitions" below.

### Description

This command displays and/or sets the current date for the system. If no parameters are used, it shows the current date. If a valid month, day, and year are provided, then the system's date will be updated. Detailed rules are listed below:

Except for numeric characters and /, all other characters in the argument are invalid. The Shell will report an error if the number is in the wrong month/date/year range.

Space before or after the numeric character is not allowed. Inserting a space into the number is invalid.

Repeated zeros are allowed before the number. For example:

```
Shell > date 000008/000004/000097
Shell > date
08/04/2097
Shell >
```

The year range is greater than or equal to 1998. Two numeric characters indicate the year. Numbers below 98 are regarded as 20xx, and numbers equal to or above 98 are regarded as 19xx. 00 means 2000. For example:

```

Shell > date 8/4/97
Shell > date
08/04/2097
Shell >

Shell > date 8/4/98
Shell > date
08/04/1998
Shell >

```

The range of valid years is from 1998–2099.

## Standard-Format Output

The standard-format from the date command has a single table: Date, with the following columns:

**Table 14** Date Command Table

Column	Description
2	Day (from 1-31)
3	Month (from 1-12)
4	Year (from 1998-2099)

## Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_DEVICE_ERROR</b>	There was a hardware error preventing the completion of this command
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

## Examples

- To display the current date in the system:

```

fs0:\> date
06/18/2001

```
- To set the date with long year format:

```

fs0:\> date 01/01/2050
fs0:\> date
01/01/2050

```
- To set the date with short year format:

```

fs0:\> date 06/18/01
fs0:\> date
06/18/2001

```

## dblk

### Summary

Displays the contents of one or more blocks from a block device.

### Usage

```
dblk device [lba] [blocks] [-b]
```

### Options

*device*

Block device name

*lba*

Index of the first block to be displayed (a hexadecimal number). The default is 0.

*blocks*

Number of blocks to be displayed (a hexadecimal number). The default is 0. If larger than 0x10, then only 0x10 are displayed.

*-b*

Display one screen at a time.

### Description

This command displays the contents of one or more blocks from a block device. **lba** and **blocks** should be typed in hex value. If **lba** is not specified, block #0 is assumed. If **blocks** is not specified, then only 1 block will be displayed. The maximum number of blocks that can be displayed at one time is 0x10.

If an MBR is found on the block, the partition information will be printed after all the block contents have been displayed.

If the block is a FAT partition, some FAT parameters will be displayed (label, systemid, oemid, sectorsize, clustersize, media etc) after all the blocks have been displayed.

### Examples

- To display one block of blk0, beginning from block 0:  
`Shell>dblk blk0`
- To display one block of fs0, beginning from block 0x2:  
`Shell>dblk fs0 2`
- To display 0x5 blocks of fs0, beginning from block 0x12:  
`Shell>dblk fs0 12 5`
- To display 0x10 blocks of fs0, beginning from block 0x12:  
`Shell>dblk fs0 12 10`
- The attempt to display more than 0x10 blocks will display only 0x10 blocks:

```
Shell>dbl k fs0 12 20
```

- To display one block of blk2, beginning from the first block (blk0):

fs1:\tmps1> dblk blk2 0 1

```
LBA 0000000000000000 Size 00000200 bytes BlkIo 3F0CEE78
00000000: EB 3C 90 4D 53 44 4F 53-35 2E 30 00 02 04 08 00
*.<.MSDOS5.0.....*
00000010: 02 00 02 00 00 F8 CC 00-3F 00 FF 00 3F 00 00 00
*.....?..?..*
00000020: 8E 2F 03 00 80 01 29 2C-09 1B D0 4E 4F 20 4E 41
*/.....),...NO NA*
00000030: 4D 45 20 20 20 20 46 41-54 31 36 20 20 20 33 C9 *ME FAT16
3.*
00000040: 8E D1 BC F0 7B 8E D9 B8-00 20 8E C0 FC BD 00 7C *.....
.....*
00000050: 38 4E 24 7D 24 8B C1 99-E8 3C 01 72 1C 83 EB 3A
*8N$.$.<.r....*
00000060: 66 A1 1C 7C 26 66 3B 07-26 8A 57 FC 75 06 80 CA
*f...&f;.&.W.u...*
00000070: 02 88 56 02 80 C3 10 73-EB 33 C9 8A 46 10 98 F7
*..V.....s..3..F...*
00000080: 66 16 03 46 1C 13 56 1E-03 46 0E 13 D1 8B 76 11
*f..F..V..F.....v.*
00000090: 60 89 46 FC 89 56 FE B8-20 00 F7 E6 8B 5E 0B 03 *`.F..V..
.....^*
000000A0: C3 48 F7 F3 01 46 FC 11-4E FE 61 BF 00 00 E8 E6
*.H...F..N.a.....*
000000B0: 00 72 39 26 38 2D 74 17-60 B1 0B BE A1 7D F3 A6 *.r9&8-
t.....*
000000C0: 61 74 32 4E 74 09 83 C7-20 3B FB 72 E6 EB DC A0 *at2Nt...
;.r....*
000000D0: FB 7D B4 7D 8B F0 AC 98-40 74 0C 48 74 13 B4 0E
*.....@t.Ht...*
000000E0: BB 07 00 CD 10 EB EF A0-FD 7D EB E6 A0 FC 7D EB
*.....*
000000F0: E1 CD 16 CD 19 26 8B 55-1A 52 B0 01 BB 00 00 E8
*.....&.U.R.....*
00000100: 3B 00 72 E8 5B 8A 56 24-BE 0B 7C 8B FC C7 46 F0
*;r.[.V$.<.....F.*
00000110: 3D 7D C7 46 F4 29 7D 8C-D9 89 4E F2 89 4E F6 C6
*-.F.)....N..N..*
00000120: 06 96 7D CB EA 03 00 00-20 0F B6 C8 66 8B 46 F8 *.....
...f.F.*
00000130: 66 03 46 1C 66 8B D0 66-C1 EA 10 EB 5E 0F B6 C8
*f.F.f..f.....^*
00000140: 4A 4A 8A 46 0D 32 E4 F7-E2 03 46 FC 13 56 FE EB
*JJ.F.2....F..V...*
00000150: 4A 52 50 06 53 6A 01 6A-10 91 8B 46 18 96 92 33
*JRP.Sj.j...F...3*
00000160: D2 F7 F6 91 F7 F6 42 87-CA F7 76 1A 8A F2 8A E8
*.....B....v.....*
00000170: C0 CC 02 0A CC B8 01 02-80 7E 02 0E 75 04 B4 42
*.....u..B*
00000180: 8B F4 8A 56 24 CD 13 61-61 72 0B 40 75 01 42 03
*...V$.aar.@u.B.*
00000190: 5E 0B 49 75 06 F8 C3 41-BB 00 00 60 66 6A 00 EB
*^..Iu...A...`fj..*
000001A0: B0 4E 54 4C 44 52 20 20-20 20 20 20 0D 0A 52 65 *.NTLDR
..Re*
000001B0: 6D 6F 76 65 20 64 69 73-6B 73 20 6F 72 20 6F 74 *move disks
or ot*
000001C0: 68 65 72 20 6D 65 64 69-61 2E FF 0D 0A 44 69 73 *her
media....Dis*
000001D0: 6B 20 65 72 72 6F 72 FF-0D 0A 50 72 65 73 73 20 *k
error...Press *
000001E0: 61 6E 79 20 6B 65 79 20-74 6F 20 72 65 73 74 61 *any key to
resta*
000001F0: 72 74 0D 0A 00 00 00 00-00 00 00 AC CB D8 55 AA
*rt.....U.*
```

```
Fat 16 BPB  FatLabel: 'NO NAME      '  SystemId: 'FAT16   '  OemId:
'MSDOS5.0'
SectorSize 200  SectorsPerCluster 4  ReservedSectors 8  # Fats 2
Root Entries 200  Media F8  Sectors 32F8E  SectorsPerFat CC
SectorsPerTrack 3F  Heads 255
```

## del

### Summary

Internal alias for the `rm` command.



## devices

### Summary

Displays the list of devices managed by UEFI drivers.

### Usage

```
devices [-b] [-l XXX]
```

### Options

```
-b          - Display one screen at a time  
-l XXX     - Display devices using the specified ISO 639-2 language
```

### Description

The command prints a list of devices that are being managed by drivers that follow the UEFI Driver Model.

### Examples

- To display all devices compliant with the EFI Driver Model

```
Shell> devices  
C T D  
T Y C I  
R P F A  
L E G G #P #D #C Device Name  
== = = = == == ==  
=====
```

20	R	-	-	-	1	13	VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)
3D	D	-	-	3	-	-	Primary Console Input Device
3E	D	-	-	3	-	-	Primary Console Output Device
64	B	-	-	1	6	2	"UGA Window 1
65	B	-	-	1	6	2	UGA Window 2"
66	B	-	-	1	1	1	EFI_WIN_NT_SERIAL_PORT=COM1
67	B	-	-	1	1	1	COM1
68	B	-	-	1	4	2	PC-ANSI Serial Console
69	D	-	-	1	-	-	EFI_WIN_NT_SERIAL_PORT=COM2
6E	D	-	-	1	-	-	EFI_WIN_NT_PHYSICAL_DISKS=e;RW;262144;512
6F	D	-	-	1	-	-	EFI_WIN_NT_CPU_MODEL=Intel(R) Processor Model
70	D	-	-	1	-	-	EFI_WIN_NT_CPU_SPEED=3000
71	D	-	-	1	-	-	EFI_MEMORY_SIZE=64
72	D	-	-	1	-	-	EFI_MEMORY_SIZE=64

### Standard-Format Output

Table 15 Standard-Format Output for devices

Column Number	Description
2	The handle number of the EFI device

3	The device type: R – Root Controller B – Bus Controller D – Device Controller
4	A managing driver supports the Driver Configuration Protocol: Yes (Y) or No (N)
5	A managing driver supports the Driver Diagnostics Protocol: Yes (Y) or No (N)
6	The number of parent controllers for this device
7	The number of this type of devices.
8	The number of child controllers produced by this device
9	The name of the device from the Component Name Protocol

## devtree

### Summary

Displays the tree of devices compliant with the UEFI Driver Model.

### Usage

```
devtree [-b] [-d] [-l XXX] [DeviceHandle]
```

### Options

*DeviceHandle*

Display device tree below a certain handle

*-b*

Display one screen at a time

*-d*

Display device tree using device paths

*-l*

Display device tree using the specified language

### Description

This command prints a tree of devices that are being managed by drivers that follow the UEFI Driver Model. By default, the devices are printed in device names that are retrieved from the Component Name Protocol. If the option *-d* is specified, the device paths will be printed instead.

### Examples

- To display the tree of all devices compliant with the UEFI Driver Model:  

```
Shell> devtree
```
- To display the tree of all devices below device 28 compliant with the UEFI Driver Model:  

```
Shell> devtree 28
```
- To display the tree of all devices compliant with the UEFI Driver Model one screen at a time:  

```
Shell> devtree -b
```

## dh

### Summary

Displays the device handles in the UEFI environment.

### Usage

```
dh [-l <lang>] [handle | -p <prot_id>] [-d] [-v]
```

### Options

*handle*

Specific handle to dump information about (a hexadecimal number). If not present, then all information will be dumped.

**-p**

Dumps all handles of a protocol specified by the GUID.

**-d**

Dumps UEFI Driver Model-related information.

**-l**

Dumps information using the language codes, as described in Appendix M of the UEFI specification.

**-sfo**

Displays information as described in "Standard-Format Output" below.

**-v, -verbose**

Dumps verbose information about a specific handle.

### Description

This command displays the device handles in the EFI environment. If this command is used with a specific handle number, the details of all the protocols that are associated with that device handle are displayed. Otherwise, the **-p** option can be used to list the device handles that contain a specific protocol.

If neither **-p** or *handle* is specified, then all handles will be displayed.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

## Examples

To display all handles and display one screen at a time:

```
Shell> dh -b
Handle dump
1: Image(DXE Core)
2: FwVol FwFileSys FwVolBlk DevPath(MemMap(11:1B50000-
1D4FFC8))
3: Image(Ebc)
4: DevPath(MemMap(11:1CA0000-1CB0000))
5: Image(WinNtThunk)
6: WinNtThunk DevPath(..76F3-11D4-BCEA-0080C73C8881)
7: Image(WinNtBusDriver) DriverBinding
...
```

To display the detailed information on handle 0x30:

```
Shell> dh 30
Handle 30 (01AF5308)
IsaIo
ROM Size.....: 00000000
ROM Location..: 00000000
ISA Resource List :
IO  : 000003F8-000003FF  Attr : 00000000
INT : 00000004-00000000  Attr : 00000000

dpath
PNP Device Path for PnP
HID A0341D0, UID 0x0
Hardware Device Path for PCI
PNP Device Path for PnP
HID 50141D0, UID 0
AsStr: 'Acpi(PNP0A03,0)/Pci(1F|0)/Acpi(PNP0501,0)'
```

To display all handles with 'diskio' protocol:

```
Shell> dh -p diskio
Handle dump by protocol 'Diskio'
15: DiskIo BlkIo DevPath(..i(3|1)/Ata(Secondary,Master))
16: DiskIo BlkIo DevPath(..,1)/PCI(0|0)/Scsi(Pun0,Lun0))
44: DiskIo BlkIo Fs DevPath(..ABD0-01C0-507B-9E5F8078F531))
ESP
45: DiskIo BlkIo Fs DevPath(..i(Pun0,Lun0)/HD(Part4,SigG0))
ESP
17: DiskIo BlkIo DevPath(..PCI(3|1)/Ata(Primary,Master))
```

To display all handles with 'Image' protocol and break when the screen is full:

```

Shell> dh -p Image -b
Handle dump by protocol 'image'
1: Image(DXE Core)
5: Image(WinNtThunk)
7: Image(WinNtBusDriver) DriverBinding
8: Image(Metronome)
A: Image(IsaBus) DriverBinding
B: Image(WinNtConsole) DriverBinding
...

```

## Standard-Format Output

When using the `-sfo` command-line option, the `dh` shell command will produce one of two tables: `HandlesInfo` or `HandleInfo`. The table columns are described in the following table:

**Table 16 dh Standard Formatted Output (HandlesInfo)**

Column Number	Description
2	Driver Name. Name of driver producing the handle.
3	Controller Name. Name of controller producing the handle.
4	Handle Number. Integer handle number.
5	Device Path. Device path associated with the handle.
6	Protocol Identifiers. Semicolon-delimited list of protocol identifiers or GUIDs.

## dir

### Summary

An internal alias for the `ls` command.

## disconnect

### Summary

Disconnects one or more drivers from the specified devices.

### Usage

```
disconnect DeviceHandle [DriverHandle [ChildHandle]]
```

### Options

#### *DeviceHandle*

Device handle (a hexadecimal number). If not specified, then disconnect *DriverHandle*.

#### *DriverHandle*

Driver handle (a hexadecimal number)

#### *ChildHandle*

Child handle of a device (a hexadecimal number). If not specified, then all child handles of *DeviceHandle* will be disconnected.

#### **-r**

Disconnect all drivers from all devices.

### Description

This command disconnects one or more drivers from the specified devices. If the **-r** option is used, all drivers are disconnected from all devices in the system. The following example is the typical output from the help for this command.

### Status Codes Returned

SHELL_SUCCESS	The action was completed as requested.
SHELL_SECURITY_VIOLATION	This function was not performed due to a security violation
SHELL_INVALID_PARAMETER	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

### Examples

- To disconnect all drivers from all devices:  
**Shell> disconnect -r**
- To disconnect all drivers from device 0x28:  
**fs0:\> disconnect 28**
- To disconnect driver 0x17 from device 0x28:



```
fs0:\> disconnect 28 17
```

- To disconnect driver 0x17 from controlling the child 0x32 of device 0x28

```
fs0:\> disconnect 28 17 32
```

## dmem

### Summary

Displays the contents of system or device memory.

### Usage

```
dmem [-b] [address] [size] [-MMIO]
```

### Options

*address*

Starting address in hexadecimal format

*size*

Number of bytes to display in hexadecimal format

**-b**

Display one screen at a time

**-MMIO**

Forces address cycles to the PCI bus

### Description

This command displays the contents of system memory or device memory. *address* and *size* should be typed in hex value. If Address is not specified, then the contents of the EFI System Table are displayed. Otherwise, memory starting at Address is displayed. *size* specifies the number of bytes to display. If *size* is not specified, then it defaults to 512 bytes. If **MMIO** is not specified, then main system memory is displayed. Otherwise, device memory is displayed through the use of the **EFI\_PCI\_ROOT\_BRIDGE\_IO\_PROTOCOL**.

## Examples

```
* To display the EFI system table pointer entries:
fs0:\> dmem

Memory Address 000000003FF7D808 200 Bytes
3FF7D808: 49 42 49 20 53 59 53 54-02 00 01 00 78 00 00 00 *IBI
SYST....x...*
3FF7D818: 5C 3E 6A FE 00 00 00 00-88 2E 1B 3F 00 00 00 00
*\>j.....?....*
3FF7D828: 26 00 0C 00 00 00 00 00-88 D3 1A 3F 00 00 00 00
*&.....?....*
3FF7D838: A8 CE 1A 3F 00 00 00 00-88 F2 1A 3F 00 00 00 00
*..?.....?....*
3FF7D848: 28 EE 1A 3F 00 00 00 00-08 DD 1A 3F 00 00 00 00
*(..?.....?....*
3FF7D858: A8 EB 1A 3F 00 00 00 00-18 C3 3F 3F 00 00 00 00
*..?.....?....*
3FF7D868: 00 4B 3F 3F 00 00 00 00-06 00 00 00 00 00 00 00
*.K.....*
3FF7D878: 08 DA F7 3F 00 00 00 00-70 74 61 6C 88 00 00 00
*...?....ptal....*
3FF7D888: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D898: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D8A8: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D8B8: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D8C8: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D8D8: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D8E8: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D8F8: 00 00 00 00 00 00 00 00-70 68 06 30 88 00 00 00
*.....ph.0....*
3FF7D908: 65 76 6E 74 00 00 00 00-02 02 00 60 00 00 00 00
*evnt.....~.....*
3FF7D918: 18 6F 1A 3F 00 00 00 00-10 E0 3F 3F 00 00 00 00
*.o.?.....*
3FF7D928: 10 00 00 00 00 00 00 00-40 C0 12 3F 00 00 00 00
*.....@..?....*
3FF7D938: 10 80 13 3F 00 00 00 00-00 00 00 00 00 00 00 00
*..?.....*
3FF7D948: 00 00 00 00 00 00 00 00-40 7D 3F 3F 00 00 00 00
*.....@.....*
3FF7D958: 50 6F 1A 3F 00 00 00 00-00 00 00 00 00 00 00 00
*Po.?.....*
3FF7D968: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D978: 00 00 00 00 00 00 00 00-70 74 61 6C 88 00 00 00
*.....ptal....*
3FF7D988: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D998: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D9A8: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D9B8: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D9C8: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D9D8: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
```

```
3FF7D9E8: 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
3FF7D9F8: 00 00 00 00 00 00 00 00 00-70 68 06 30 A0 00 00 00
*.....ph.0.....*
```

Valid EFI Header at Address 000000003FF7D808

```
-----
System: Table Structure size 00000078 revision 00010002
ConIn (3F1AD388) ConOut (3F1AF288) StdErr (3F1ADD08)
Runtime Services 000000003F3FC318
Boot Services    000000003F3F4B00
SAL System Table 000000003FF22760
ACPI Table       000000003FFD9FC0
ACPI 2.0 Table   00000000000E2000
MPS Table        000000003FFD0000
SMBIOS Table     00000000000F0020
```

\* To display memory contents from 1af3088 with size of 16 bytes:

```
Shell> dmem 1af3088 16
Memory Address 000000001AF3088 16 Bytes
01AF3088: 49 42 49 20 53 59 53 54-00 00 02 00 18 00 00 00 *IBI
SYST.....*
01AF3098: FF 9E D7 9B 00 00 *.....*
```

\* To display memory mapped IO contents from 1af3088 with size of 16 bytes:

```
Shell> dmem 1af3088 16 -MMIO
```

## dmpstore

### Summary

Manages all UEFI NVRAM variables.

### Usage

```
dmpstore [-b] [-d] [-all | ([variable] [-guid guid])]
dmpstore [-all | ([variable] [-guid guid])] -s file
dmpstore [-all | ([variable] [-guid guid])] -l file
```

### Options

**-b**

Display one screen at a time

**variable**

Specifies the name of the variable name. May be a literal name or a pattern as specified in the `MetaMatch()` function of the `EFI_UNICODE_COLLATION2_PROCOOL`.

**-guid**

Specifies the GUID of the variables to be displayed. The GUID has the standard text format. If `guid` is not specified and `-all` is not specified, then the `EFI_GLOBAL_VARIABLE` GUID is assumed.

**-all**

Indicates that all variables should be dumped, including those with a different GUID that `EFI_GLOBAL_VARIABLE`.

**-d**

Delete variables

**-s**

Save variables to file

**-l**

Load and set variables from file

### Description

This command is used to manage the UEFI NVRAM variables. The variables to display or delete depend on the command line options, as specified in the following table:

**Table 17 Variable command line options**

Variable	GUID	-all	Description
---	---	---	All variables with the GUID <code>EFI_GLOBAL_VARIABLE</code> will be operated on.
---	---	---	All variables (regardless of GUID or name) will be operated on.

---	X	---	All variables with the specified GUID will be operated on.
X	---	---	The variable with the GUID EFI_GLOBAL_VARIABLE and the name(s) Variable will be operated on.
X	X	---	The variable with the specified GUID and name Variable will be operated on.

The variable value is printed as hexadecimal dump.

Option **-d** is used to delete variables. Option **-s** and **-l** are used to save and load variables to and from file. The variable name can be specified when using these variables so that the operation only takes effect on that variable.

## drivers

### Summary

Displays a list of information for drivers that follow the UEFI Driver Model in the UEFI environment.

### Usage

```
drivers [-l XXX]
```

### Options

**-l**

Displays drivers using the language code XXX, which has the format specified by Appendix M of the UEFI specification.

**-sfo**

Displays information as described in "Standard-Format Output" below.

### Description

This command displays a list of information for drivers that follow the UEFI Driver Model in the UEFI environment. The list includes:

- The handle number of the EFI driver.
- The version number of the EFI driver.
- The driver type. A **B** in this column indicates a bus driver, and **D** indicates a device driver.
- Indicates that the driver supports the Driver Configuration Protocol.
- Indicates that the driver supports the Driver Diagnostics Protocol.
- The number of devices that this driver is managing.
- The number of child devices that this driver has produced.
- The name of the driver from the Component Name Protocol.
- The file path from which the driver was loaded.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

## Examples

- To display the list:

```
Shell> drivers
          T D
          Y C I
          P F A
V VERSION E G G #D #C DRIVER NAME          IMAGE
NAME
=====
39 00000010 D - - 1 - Platform Console Management
Driver      ConPlatform
3A 00000010 D - - 1 - Platform Console Management
Driver      ConPlatform
3B 00000010 B - - 1 1 Console Splitter
Driver      ConSplitter
3C 00000010 ? - - - - Console Splitter
Driver      ConSplitter
3D 00000010 B - - 1 1 Console Splitter
Driver      ConSplitter
3E 00000010 ? - - - - Console Splitter
Driver      ConSplitter
42 00000010 D - - 1 - UGA Console
Driver      GraphicsConsole
43 00000010 ? - - - - Serial Terminal
Driver      Terminal
44 00000010 D - - 1 - Generic Disk I/O Driver      DiskIo
45 00000010 D - - 1 - FAT File System Driver      Fat
48 00000010 ? - - - - ISA Bus Driver              IsaBus
49 00000010 ? - - - - ISA Serial
Driver      IsaSerial
4C 00000010 B - - 1 1 PCI Bus Driver              PciBus
55 00000010 D X X 1 - Windows Block I/O
Driver      WinNtBlockIo
56 00000010 ? - - - - Windows Text Console
Driver      WinNtConsole
57 00000010 ? - - - - Windows Serial I/O
Driver      WinNtSerialIo
58 00000010 D - - 1 - Windows Simple File System Driver
WinNtSimpleFileSystem
59 00000010 B - - 1 3 Windows Bus
Driver      WinNtBusDriver
5F 00000010 D - - 1 - Windows Universal Graphics
Adapter     WinNtUga
```

## Standard-Format Output

The standard-format output for the drivers command produces a single table: Drivers. The following columns are described:

**Table 18 Drivers command table**

Column Number	Description
2	Handle Number. The handle number of the UEFI driver.
3	Version Number. The version number of the UEFI Driver
4	Driver Type. Either 'B' for bus driver or 'D' for device driver.
5	Configuration Protocol Support. Either "Y" (Yes) or "N" (No)
6	Driver Protocol Support. Either 'Y' (Yes) or 'N' (No)



7	Devices Managed. The number of devices that this driver is managing.
8	Child Devices. The number of child devices that this driver has produced.
9	Driver Name. The name of the driver from the Component Name Protocol.
10	Driver Image Path. The device path from which the driver was loaded.

## drvcfg

### Summary

Configures the driver using the platform's underlying configuration infrastructure.

### Usage

```
drvcfg [-l XXX] [-c] [-f <Type>|-v|-s] [DriverHandle [DeviceHandle  
[ChildHandle]]] [-i filename] [-o filename]
```

### Options

#### *Type*

The type of default configuration options to force on the controller.

0 - Standard Defaults.

1 - Manufacturing Defaults.

2 - Safe Defaults.

4000-FFFF - Custom Defaults.

#### *DriverHandle*

The handle of the driver to configure

#### *DeviceHandle*

The handle of a device that DriverHandle is managing

#### *ChildHandle*

The handle of a device that is a child of DeviceHandle

#### *-c*

Configure all child devices

#### *-l*

Configure using the ISO 3066 language specified by XXX

#### *-f*

Force defaults

#### *-v*

Validate options

#### *-s*

Set options

#### *-i*

Receive configuration updates from an input file

-o

Export the settings of the specified driver instance to a file

## Description

This command invokes the platform's Configuration infrastructure. The table below describes the values for the *Type* parameter. Other values depend on the driver's implementation.

**Table Default Values for the "Type" Parameter**

Value	Type of Default	Description
0x0000	Standard Defaults	Places a controller in a state that is prepared for normal operation in a platform.
0x0001	Manufacturing Defaults	Optional type that places the controller in a configuration that is suitable for a manufacturing and test environment.
0x0002	Safe Defaults	Optional type that places a controller in a safe configuration that has the greatest probability of functioning correctly in a platform.
0x0003 – 0x3FFF	Reserved	Specification reserved range of default values
0x4000 – 0xFFFF	Custom Defaults	Optional type that places the controller in a configuration that has custom characteristics.

## Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_UNSUPPORTED</b>	The action as requested was unsupported.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

## Examples

To display the list of devices that are available for configuration:

```
Shell> drvcfg
```

To display the list of devices and child devices that are available for configuration:

```
Shell> drvcfg -c
```

To force defaults on all devices:

```
Shell> drvcfg -f 0
```

To force defaults on all devices that are managed by driver 0x17:

```
Shell> drvcfg -f 0 17
```

To force defaults on device 0x28 that is managed by driver 0x17:

```
Shell> drvcfg -f 0 17 28
```

To force defaults on all child devices of device 0x28 that is managed by driver 0x17:

```
Shell> drvcfg -f 0 17 28 -c
```

To force defaults on child device 0x30 of device 0x28 that is managed by driver 0x17:

```
Shell> drvcfg -f 0 17 28 30
```

To validate options on all devices:

```
Shell> drvcfg -v
```

To validate options on all devices that are managed by driver 0x17:

```
Shell> drvcfg -v 17
```

To validate options on device 0x28 that is managed by driver 0x17:

```
Shell> drvcfg -v 17 28
```

To validate options on all child devices of device 0x28 that is managed by driver 0x17:

```
Shell> drvcfg -v 17 28 -c
```

To validate options on child device 0x30 of device 0x28 that is managed by driver 0x17:

```
Shell> drvcfg -v 17 28 30
```

To set options on device 0x28 that is managed by driver 0x17:

```
Shell> drvcfg -s 17 28
```

To set options on child device 0x30 of device 0x28 that is managed by driver 0x17:

```
Shell> drvcfg -s 17 28 30
```

To set options on device 0x28 that is managed by driver 0x17 in English:

```
Shell> drvcfg -s 17 28 -l eng
```

To set options on device 0x28 that is managed by driver 0x17 in Spanish:

```
Shell> drvcfg -s 17 28 -l spa
```

## drvdiag

### Summary

Invokes the Driver Diagnostics Protocol.

### Usage

```
drvdiag [-c] [-l XXX] [-s|-e|-m] [DriverHandle [DeviceHandle  
[ChildHandle]]]
```

### Options

*DriverHandle*

The handle of the driver to diagnose

*DeviceHandle*

The handle of a device that DriverHandle is managing

*ChildHandle*

The handle of a device that is a child of DeviceHandle

*-c*

Diagnose all child devices

*-l*

Diagnose using the ISO 639-2 language specified by XXX

*-s*

Run diagnostics in standard mode

*-e*

Run diagnostics in extended mode

*-m*

Run diagnostics in manufacturing mode

### Description

This command invokes the Driver Diagnostics Protocol.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

## Examples

To display the list of devices that are available for diagnostics:

```
Shell> drvdiag
```

To display the list of devices and child devices that are available for diagnostics:

```
Shell> drvdiag -c
```

To run diagnostics in standard mode on all devices:

```
Shell> drvdiag -s
```

To run diagnostics in standard mode on all devices in English:

```
Shell> drvdiag -s -l eng
```

To run diagnostics in standard mode on all devices in Spanish:

```
Shell> drvdiag -s -l spa
```

To run diagnostics in standard mode on all devices and child devices:

```
Shell> drvdiag -s -c
```

To run diagnostics in extended mode on all devices:

```
Shell> drvdiag -e
```

To run diagnostics in manufacturing mode on all devices:

```
Shell> drvdiag -m
```

To run diagnostics in standard mode on all devices managed by driver 0x17:

```
Shell> drvdiag -s 17
```

To run diagnostics in standard mode on device 0x28 managed by driver 0x17:

```
Shell> drvdiag -s 17 28
```

To run diagnostics in standard mode on all child devices of device 0x28 managed by driver 0x17:

```
Shell> drvdiag -s 17 28 -c
```

To run diagnostics in standard mode on child device 0x30 of device 0x28 managed by driver 0x17:

```
Shell> drvdiag -s 17 28 30
```

## echo

### Summary

Controls whether or not script commands are displayed as they are read from the script file and prints the given message to the display.

### Usage

```
echo [-on|-off]
echo [message]
```

### Options

*message*

Message to display

**-on**

Enables display when reading commands from script files.

**-off**

Disables display when reading commands from script files.

### Description

The first form of this command controls whether or not script commands are displayed as they are read from the script file. If no argument is given, the current "on" or "off" status is displayed. The second form prints the given message to the display.

### Note

This command does not change the value of the environment variable **lasterror**.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation

### Examples

- To display a message string of 'Hello World':

```
fs0:\> echo Hello World
Hello World
```
- To turn command echoing on:

```
fs0:\> echo -on
```
- To execute HelloWorld.nsh, and display when reading lines from the script file:



```
fs0:\> HelloWorld.nsh
+HelloWorld.nsh> echo Hello World
Hello World
```

- To turn command echoing off:  

```
fs0:\> echo -off
```
- To display the current echo setting:  

```
fs0:\> echo
Echo is off
```

## edit

### Summary

Full screen editor for ASCII or UCS-2 files.

### Usage

```
edit [file]
```

### Options

*file*

Name of file to be edited. If none is specified, then an empty file will be created with a default file name.

### Description

This command allows a file to be edited using a full screen editor. The editor supports both UCS-2 and ASCII file types. The following example shows typical output for help on this command.

### Examples

- To edit the 'shell.log' file:  

```
fs0:\> edit shell.log
```

## eficompress

### Summary

Compress a file using EFI Compression Algorithm.

### Usage

```
eficompress infile outfile
```

### Options

```
infile - Filename for uncompressed input file  
outfile - Filename for compressed output file
```

### Description

This command is used to compress a file using EFI Compression Algorithm and write the compressed form out to a new file.

### Examples

- To compress a file named 'uncompressed' to file 'compressed':  

```
fs0:\> efigompress uncompressed compressed
```

## efidecompress

### Summary

Decompress a file using EFI Decompression Algorithm.

### Usage

```
efidecompress infile outfile
```

### Options

*infile*

Filename of compressed input file

*outfile*

Filename of decompressed output file

### Description

This command is used to decompress a file using EFI Decompression Algorithm and write the decompressed form out to a new file.

### Examples

- To decompress a file named 'compressed' to file 'uncompressed':  

```
fs0:\> efidecompress compressed uncompressed
```

## exit

### Summary

Exits the UEFI Shell or the current script.

### Usage

```
exit [/b] [exit-code]
```

### Options

*/b*

Indicates that only the current UEFI shell script should be terminated. Ignored if not used within a script.

*exit-code*

If exiting a UEFI shell script, the value that will be placed into the environment variable **lasterror**. If exiting an instance of the UEFI shell, the value that will be returned to the caller. If not specified, then 0 will be returned.

### Description

This command exits the UEFI Shell or, if */b* is specified, the current script.

### Status Codes Returned

0	Exited normally
exit-code	The value specified as an option.

## for

### Usage

```
for %indexvar in set
  command [arguments]
  [command [arguments]]
  ...
endfor

for %indexvar run (start end [step])
  command [arguments]
  [command [arguments]]
  ...
endfor
```

### Description

The **for** command executes one or more *commands* for each item in a *set* of items. The *set* may be text strings or filenames or a mixture of both, separated by spaces (if not in a quotation). If the length of an element in the *set* is between 0 and 256, and if the string contains wildcards, the string will be treated as a file name containing wildcards, and be expanded before *command* is executed.

If after expansion no such files are found, the literal string itself is kept. *Indexvar* is any alphabet character from 'a' to 'z' or 'A' to 'Z', and they are case sensitive. It should not be a digit (0-9) because %digit will be interpreted as a positional argument on the command line that launches the script. The namespace for index variables is separate from that for environment variables, so if *indexvar* has the same name as an existing environment variable, the environment variable will remain unchanged by the **for** loop.

Each *command* is executed once for each item in the *set*, with any occurrence of *%indexvar* in the command replacing with the current item. In the second format of for ... endfor statement, *indexvar* will be assigned a value from *start* to *end* with an interval of *step*. *start* and *end* can be any integer whose length is less than 7 digits excluding sign, and it can also applied to *step* with one exception of zero. *step* is optional, if *step* is not specified it will be automatically determined by following rule, if  $start \leq end$  then  $step = 1$ , otherwise  $step = -1$ . *start*, *end* and *step* are divided by space.

### Note

This command may only be used in scripts.

This command does not change the value of the environment variable **lasterror**.

## Examples

```
#  
# Sample for loop - listing all .txt files  
#  
echo -off  
for %a in *.txt  
    echo %a exists  
endfor
```

If in current directory, there are 2 files named file1.txt and file2.txt, the output of the sample script will be:

```
Sample1> echo -off  
file1.txt exists  
file2.txt exists
```

Theoretically it is legal for 2 nested `for` commands to use the same alphabet letter as their index variable, for instance, `a`.

```
#  
# Sample for loop from 1 to 3 with step 1  
#  
echo -off  
for %a run (1 3)  
    echo %a  
endfor  
  
#  
# Sample for loop from 3 down to 1 with step -1  
#  
echo -off  
for %a run (3 1 -1)  
    echo %a  
endfor  
  
#  
# Sample for loop - 2 nested for using same index variable  
#  
echo -off  
for %a in value1 value2  
    for %a in value3 value4  
        echo %a  
    endfor  
endfor
```

When processing first `for` and before seeing the `endfor`, the index variable `%a` has the value "value1", so in second `for`, the `%a` has been already defined and it will be replaced with the current value of `%a`. The string after substitution becomes `for value1 in value3 value4`, which is not a legal `for` command. Thus only when the value of `%a` is also a single alphabet letter, the script will be executed without error. If 2 independent `for` commands use the same index variable, when the second `for` is encountered, the first `for` has already freed the variable so there'll be no problem in this case.

## getmtc

### Usage

```
getmtc
```

### Description

This command displays the current monotonic counter value. The lower 32 bits increment every time this command is executed. Every time the system is reset, the upper 32 bits will be incremented, and the lower 32 bits will be reset to 0. The following example is typical output from help for this command.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_DEVICE_ERROR</b>	The underlying device was not working correctly.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation

### Examples

```
fs0:\> getmtc
100000000
fs0:\> getmtc
100000001
```



## goto

### Usage

```
goto label
```

### Description

The `goto` command directs script file execution to the line in the script file after the given *label*. The command is not supported from the interactive shell. A *label* is a line beginning with a colon (:). It can appear either after the `goto` command, or before the `goto` command. The search for *label* is done forward in the script file, from the current file position. If the end of the file is reached, the search resumes at the top of the file and continues until *label* is found or the starting point is reached. If *label* is not found, the script process terminates and an error message is displayed. If a *label* is encountered but there is no `goto` command executed, the *label* lines are ignored. Using `goto` command to jump into another `for` loop is not allowed, but jumping into an `if` statement is legal.

### Note

The `goto` command is only valid in script files.

### Examples

```
# This is a script
goto Done
...
:Done
cleanup.nsh
```

## guid

### Summary

Displays all registered EFI GUIDs.

### Usage

```
guid [-b]
```

### Options

```
-b
```

Display one screen at a time

### Description

This command displays a list of all the GUIDs that have been registered with the UEFI environment.

### Examples

Note:

1. Only displays the GUIDs included in the core EFI build at the time the core was built. Additional GUIDs may have been added by the BIOS integrator. Any GUIDs that are not in the original core build or were added by a new protocol loaded by the user will show up as an Unknown Device.
2. The GUID with a '\*' at end means that there will probably be dump information or token available for the protocol to this GUID. The 'dh' command can be used to dump this information.

```
fs0:\> guid -b
DevIo           : AF6AC311-84C3-11D2-8E3C-00A0C969723B
diskio          : CE345171-BA0B-11D2-8E4F-00A0C969723B
blkio           : 964E5B21-6459-11D2-8E39-00A0C969723B *
txtin           : 387477C1-69C7-11D2-8E39-00A0C969723B
txtout          : 387477C2-69C7-11D2-8E39-00A0C969723B *
fs              : 964E5B22-6459-11D2-8E39-00A0C969723B
load            : 56EC3091-954C-11D2-8E3F-00A0C969723B
image           : 5B1B31A1-9562-11D2-8E3F-00A0C969723B *
...
```

```
Examples
Shell> exit
```

## help

### Summary

Displays the list of commands that are built into the UEFI Shell.

### Usage

```
help [cmd | pattern / special] [-usage] [-verbose] [-section  
sectionname] [-b]
```

### Options

*cmd*

Command to display help about.

*pattern*

Pattern which describes the commands to be displayed.

**special**

Displays a list of the special characters used in the shell command line.

**-usage**

Display the usage information for the command. The same as specifying **-section:NAME** and **-section:SYNOPSIS**

**-section** *sectionname*

Display the specified section of the help information. Standard section names can be found in Appendix B.

### Description

The help command displays information about one or more shell commands.

If no other options are specified, each command will be displayed along with a brief description of its function. If **-verbose** is specified, then all help information for the specified commands. If **-section** is specified, only the help section specified will be displayed (see below). If **-usage** is specified, then the command, a brief description and the usage will be displayed.

The help text is gathered from UCS-2 text files found in the directory where the shell or shell command executable was located. The files have the name *command-name.man*, where *command-name* is the name of the shell command. The files follow a sub-set of the MAN page format, as described below.

If no option is specified, then only the NAME section of the page is reproduced.

### Status Codes Returned

0	The help was displayed
1	No command help was displayed

## Examples

To display the list of commands in the UEFI Shell and break after one screen:

```
Shell> help -b
?          - Displays commands list or verbose help of a
           command
alias     - Displays, creates, or deletes aliases in the
           UEFI Shell
attrib   - Displays or changes the attributes of files
           or directories
cd        - Displays or changes the current directory
cls       - Clears the standard output with an optional
           background color
connect  - Binds an EFI driver to a device and starts
           the driver
copy     - Copies one or more files/directories to
           another location
...
```

To display help information of a Shell command - ls:

```
Shell> help ls
Shell> ? ls
Shell> ls -?
```

To display the list of commands that start with character 'p':

```
Shell> help p*
pause - Prints a message and suspends for keyboard input
```

## hexedit

### Summary

Full screen hex editor for files, block devices, or memory.

### Usage

```
hexedit [[-f] filename| [-d diskname offset size] | [-m address size]]
```

### Options

**-f**

Name of file to edit

**-d**

Disk block to edit:

*DiskName* - Name of disk to edit (for example fs0)

*Offset* - Starting block number (beginning from 0)

*Size* - Number of blocks to be edited

**-m**

Memory region to edit:

*Address* - Starting 32-bit memory address (beginning from 0)

*Size* - Size of memory region to be edited in bytes

### Description

This command allows a file, block device, or memory region to be edited. The region being edited is displayed as hexadecimal bytes, and the contents can be modified and saved. The following example shows typical output for help on this command.

### Examples

- To edit a file in hex mode:  

```
fs0:\> hexedit test.bin
```
- To edit block device fs0 starting at block 0 with size of 2 blocks:  

```
fs0:\> hexedit -d fs0 0 2
```
- To edit memory region starting at address 0x00000000 with size of 2 bytes:  

```
fs0:\> hexedit -m 0 2
```

## if

Controls which script commands will be executed based on provided conditional expressions.

### Usage

```
if [not] exist filename then
  command [arguments]
  [command [arguments]]
  ...
  [else
  command [arguments]
  [command [arguments]]
  ...
  ]
endif

if [/i] [not] string1 == string2 then
  command [arguments]
  [command [arguments]]
  ...
  [else
  command [arguments]
  [command [arguments]]
  ...
  ]
endif

if [/i][/s] ConditionalExpression then
  command [arguments]
  [command [arguments]]
  ...
  [else
  command [arguments]
  [command [arguments]]
  ...
  ]
Endif
```

### Options

#### *ConditionalExpression*

Conditional expression, as described in “Expressions”, below.

### Description

The **if** command executes one or more *commands* before the **else** or **endif** commands, if the specified condition is true; otherwise *commands* between **else** (if present) and **endif** are executed.

In the first usage of **if**, the **exist** condition is true when the file specified by *filename* exists. The *filename* argument may include device and path information. Also wildcard expansion is supported by this form. If more than one file matches the wildcard pattern, the condition evaluates to TRUE.

In the second usage, the `string1 == string2` condition is true if the two strings are identical. Here the comparison can be case sensitive or insensitive, it depends on the optional switch `/i`. If `/i` is specified, it will compare strings in the case insensitive manner; otherwise, it compares strings in the case sensitive manner.

In the third usage, general purpose comparison is supported using expressions optionally separated by `and` or `or`. Since `<` and `>` are used for redirection, the expressions use common two character (FORTRAN) abbreviations for the operators (augmented with unsigned equivalents):

## Expressions

Conditional expressions are evaluated strictly from left to right. Complex conditionals requiring precedence may be implemented as nested `ifs`.

The expressions used in the third usage have the following syntax:

```
conditional-expression := expression |
                           expression and expression
                           expression or expression
expression              := expr |
                           not expr
expr                    := item binop item |
                           boolfunc(string)
item                    := mapfunc(string) |
                           string
mapfunc                 := efierror | pierror | oemerror
boolfunc                := isint | exists | available | profile
binop                   := gt | lt | eq | ne | ge | le | == | ugt | ult | uge | ule
```

## Comparisons

By default, comparisons are done numerically if the strings on both sides of the operator are numbers (as defined below) and in case sensitive character sort order otherwise. Spaces separate the operators from operands.

The `/s` option forces string comparisons and the `/i` option forces case-insensitive string comparisons. If either of these is used, the signed or unsigned versions of the operators have the same results. The `/s` and `/i` apply to the entire line and must

appear at the start of the line (just after the `if` itself). The two may appear in either order.

When performing comparisons, the Unicode Byte Ordering Character is ignored at the beginning of any argument.

**Table 19 Comparison Operators**

Operator	Definition
<code>gt</code>	Greater than
<code>ugt</code>	Unsigned Greater than
<code>lt</code>	Less than
<code>ult</code>	Unsigned Less than
<code>ge</code>	Greater than or equal
<code>uge</code>	Unsigned greater than or equal
<code>le</code>	Less than or equal
<code>ule</code>	Unsigned less than or equal
<code>ne</code>	Not equal
<code>eq</code>	Equals (semantically equivalent to <code>==</code> )
<code>==</code>	Equals (semantically equivalent to <code>eq</code> )

## Error Mapping Functions

These functions are used to convert integers into UEFI, PI or OEM error codes, as defined by Appendix D of the UEFI specification.

**Table 20 Functions used to convert integers into UEFI, PI or OEM error codes**

Function	Definition
<code>UefiError</code>	Sets top nibble of parameter to 0100 binary (0x8)
<code>PiError</code>	Sets top nibble of parameter to 1010 binary (0xA)
<code>OemError</code>	Sets top nibble of parameter to 1100 binary (0xC)

Each function maps the small positive parameter into its equivalent error classification as described in Appendix D of the UEFI Specification. For example,

```
...
if %lasterror% == EfiError(8) then # Check for write protect.
...
```

These functions may only be used to modify operators in comparisons.

## Boolean Functions

The following built-in Boolean functions are also available:



**Table 21 Boolean Functions**

Function	Definition
<b>IsInt</b>	Evaluates to true if the parameter string that follows is a number (as defined below) and false otherwise.
<b>Exists</b>	Evaluates to true if the file specified by <i>string</i> exists in the current working directory or false if not.
<b>Available</b>	Evaluates to true if the file specified by <i>string</i> is in the current working directory or current path.
<b>Profile</b>	Determines whether the parameter string matches one of the profile names in the <b>profiles</b> environment variable.

No spaces are allowed between function names and the open parenthesis, between the open parenthesis and the string or between the string and the closed parenthesis. Constant strings containing spaces must be quoted.

**Note:** *To avoid ambiguity and current or future incompatibility, users are strongly encouraged to surround constant strings that contain parenthesis with quotes in **if** statements.*

## Conditional Expressions

**Not** inverts the sense of only the following expression.

## Numbers

Allowable number formats are decimal numbers and C-style case insensitive hexadecimal numbers. Numbers may be preceded by a "-" indicating a negative number. Examples:

- 13
- 46
- -0x3FFF
- 0x3fff
- 0x1234

Unsigned values must be less than  $2^{64}$ . Signed integer values are bounded by  $\pm 2^{63}$ . Numbers are internally represented in two's complement form. The representation of the number in the string has no bearing on the way that number is treated in a numeric expression – type is assigned by the operator. So, for example, **-1 lt 2** is true but **-1 ult 2** is false.

## Examples

```
#
# Example script for "if" command usages 1 and 2
#
if exist fs0:\myscript.nsh then
  myscript myarg1 myarg2
endif
if %myvar% == runboth then
  myscript1
  myscript2
else
  echo ^%myvar^ != runboth
endif
```

In this example, if the script file `myscript.nsh` exists in `fs0:\`, this script will be launched with 2 arguments, `myarg1` and `myarg2`. After that, environment variable `%myvar%` is checked to see if its value is `runboth`, if so, script `myscript1` and `myscript2` will be executed one after the other, otherwise a message `%myvar% != runboth` is printed.

```
#
# Example script for "if" command usage 3
#
:Redo
echo Enter 0-6 or q to quit
# assumes "input y" stores a character of user input into variable y
InputCh MyVar
if x%MyVar% eq x then
  echo Empty line. Try again
  goto Redo
endif

if IsInt(%MyVar%) and %MyVar% le 6 then
  myscript1 %MyVar%
  goto Redo
endif
if /i %MyVar% ne q then
  echo Invalid input
  goto Redo
endif
```

In this example, the script requests user input and uses the `if` command for input validation. It checks for empty line first and then range checks the input. Note also the use of the `/i` in the last comparison so "Q" and "q" are both supported.

**Note:** This command does not change the value of the environment variable `!lasterror`.

**Note:** The `if` command is only available in scripts.

**Note:** The `else` command is optional in an `if/else` statement.

## ifconfig

### Summary

Modify the default IP address of the UEFI IP4 Network Stack.

### Usage

```
ifConfig [-?] [-c [Name]] [-l [Name]] [-s <Name> dhcp | <static <IP> <Mask> <Gateway>> [permanent]]
```

### Options

#### *Name*

Adapter name, i.e., eth0

#### **-c** [Name]

Clear the configuration for all or specified interface, and the network stack for related interface will fall back to the DHCP as default.

#### **-l** [Name]

List the configuration for all or the specified interface.

#### **-s** <Name> **static** <IP> <SubnetMask> <GatewayMask> [permanent]

Use static IP4 address configuration for all or specified interface. If **permanent** is not present, the configuration is one-time only, otherwise this configuration request will survive a network stack reload.

#### **-s** <Name> **dhcp** [permanent]

Use DHCP4 to request the IP4 address configuration dynamically for all interface or specified interface. If **permanent** is not present, the configuration is one-time only, otherwise this configuration request will survive a network stack reload.

#### *IP*

IP4 address in four integer values (each between 0-255). i.e., 192.168.0.10

#### *SubnetMask*

Subnet mask in four integer values (each between 0-255), i.e., 255.255.255.0

#### *GatewayMask*

Default gateway in four integer values (each between 0-255), i.e., 192.168.0.1

#### **-?**

Display the help message

### Description

This command is used to modify the default IP address for the UEFI IP4 Network Stack.

## Examples

To list the configuration for the interface eth0:

```
Shell:\> IfConfig -l eth0
```

To use DHCP4 to request the IP4 address configuration dynamically for the interface eth0:

```
Shell:\> IfConfig -s eth0 dhcp
```

To use the static IP4 address configuration for the interface eth0, and this configuration survives the network reload:

```
Shell:\> IfConfig -s eth0 static 192.168.0.5 255.255.255.0  
192.168.0.1 permanent
```

## ipconfig

### Summary

Displays or modifies the current IP configuration.

### Usage

```
ipconfig [-r] | [-b] [-c Instance] [IpAddress [-m NetMask]]
```

### Options

**-r**  
Restart the PXE base code and DHCP settings

**-b**  
Display one screen at a time

**-c, -controller**  
Specify the 0-based handle of the `EFI_SIMPLE_NETWORK_PROTOCOL` instance to use.

*IpAddress*  
IP address in a.b.c.d format

**-m, -mask**  
Network mask in 255.255.255.0 format

### Description

This command displays or modifies the IP configuration of `EFI_PXE_BASE_CODE_PROTOCOL`. If no parameter is specified in the command line, it just displays the IP configuration of `EFI_PXE_BASE_CODE_PROTOCOL`. If *IpAddress* or *NetMask* is specified, the according configuration of `EFI_PXE_BASE_CODE_PROTOCOL` will be changed.

### Examples

- To restart the PXE base code and refresh the DHCP settings:  
`shell:\> IpConfig -r`
- To display the current IP configuration:  
`Shell:\> IpConfig`
- To modify the IP address without changing the network mask:  
`shell:\> IpConfig 192.168.10.30`
- To modify the IP address and network mask:  
`shell:\> IpConfig 192.168.10.30 -m 255.255.255.0`

## load

### Summary

Loads a UEFI driver into memory.

### Usage

```
load [-nc] file [file...]
```

### Options

**-nc**

Load the driver, but do not connect the driver.

*File*

File that contains the image of the UEFI driver (wildcards are permitted)

### Description

This command loads an driver into memory. It can load multiple files at one time, and the file name supports wildcards.

If the **-nc** flag is not specified, this command will try to connect the driver to a proper device; it may also cause already loaded drivers be connected to their corresponding devices.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_NOT_FOUND</b>	The requested file was not found

## Examples

```
fs0:\> load Isabus.efi
load: Image 'fs0:\Isabus.efi' loaded at 18FE000 - Success

fs0:\> load Isabus.efi IsaSerial.efi
load: Image 'fs0:\Isabus.efi' loaded at 18E5000 - Success
load: Image 'fs0:\IsaSerial.efi' loaded at 18DC000 - Success

fs0:\> load Isa*.efi
load: Image 'fs0:\IsaBus.efi' loaded at 18D4000 - Success
load: Image 'fs0:\IsaSerial.efi' loaded at 18CB000 - Success

fs0:\> load -nc IsaBus.efi
load: Image 'fs0:\Isabus.efi' loaded at 18FE000 - Success
```

## loadpcirom

### Summary

Loads a UEFI driver from a file in the format of a PCI Option ROM.

### Usage

```
loadpcirom [-nc] romfile [romfile...]
```

### Options

```
-nc      - Load the ROM image but do not connect the driver  
romfile - PCI option ROM image file (wildcards are permitted)
```

### Description

This command is used to load PCI option ROM images into memory for execution. The file can contain legacy images and multiple PE32 images, in which case all PE32 images will be loaded. The example below shows typical output from help for this command.

### Examples

```
To load a rom file 'rom.bin':  
fs0:\> LoadPciRom rom.bin  
To load '*.bin' files but do not connect the driver  
fs0:\> LoadPciRom -nc *.bin
```



## ls

### Summary

Lists a directory's contents or file information.

### Usage

```
ls [-r] [-a[attrib]] [-sfo] [file]
```

### Options

**-r**

Displays recursively (including subdirectories)

**-a**

Display only those files with the attributes which follow. If no attributes are listed, then all files will be listed. If `-a` is not specified, then all non-system and non-hidden files will be listed. The attributes (*attrib*) may be one or more of the following:

1. a - Archive
2. s - System
3. h - Hidden
4. r - Read-only
5. d - Directory

**-sfo**

Display information as described in "Standard-Format Output" below.

*file*

Name of file/directory (wildcards are permitted)

### Description

This command lists directory contents or file information. If no file name or directory name is specified, then the current working directory is assumed. The contents of a directory are listed if all of the following are true:

- If option `-r` is not specified
- If no wildcard characters are specified in the *file* parameter
- If *file* represents an existing directory

In all other cases, the command functions as follows:

- All files/directories that match the specified name are displayed.
- The `-r` flag determines whether a recursive search is performed.
- The option flag `-a[attrib]` tells the command to display only those files with the attributes that are specified by `[attrib]`. If more than one attribute is

specified, only the files that have all those attributes will be listed. If **-a** is followed by nothing, then all files/directories are displayed, regardless of their attributes. If **-a** itself is not specified, then all files except system and hidden files are displayed.

## Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_NOT_FOUND</b>	The requested file or directory was not found.

## Examples

To hide files by adding the hidden or system attribute to them:

```
fs0:\> attrib +s +h *.efi
ASH fs0:\IsaBus.efi
ASH fs0:\IsaSerial.efi
```

To display all, except the files/directories with 'h' or 's' attribute:

```
fs0:\> ls
Directory of: fs0:\

06/18/01 09:32p                153 for.nsh
06/18/01 01:02p <DIR>          512 efi
06/18/01 01:02p <DIR>          512 test1
06/18/01 01:02p <DIR>          512 test2
06/18/01 08:04p                29 temp.txt
06/18/01 08:05p <DIR>          512 test
01/28/01 08:24p                r    29 readme.txt
3 File(s)                211 bytes
4 Dir(s)
```

To display files with all attributes in the current directory:

```
fs0:\> ls -a
Directory of: fs0:\

06/18/01 09:32p                153 for.nsh
06/18/01 01:02p <DIR>          512 efi
06/18/01 01:02p <DIR>          512 test1
06/18/01 01:02p <DIR>          512 test2
06/18/01 10:59p            28,739 IsaBus.efi
06/18/01 10:59p            32,838 IsaSerial.efi
06/18/01 08:04p                29 temp.txt
06/18/01 08:05p <DIR>          512 test
01/28/01 08:24p                r    29 readme.txt
5 File(s)            61,788 bytes
4 Dir(s)
```

To display files with read-only attributes in the current directory:

```

fs0:\> ls -ar
Directory of: fs0:\

    06/18/01  11:14p          r             29  readme.txt
           1 File(s)          29 bytes
           0 Dir(s)

```

To display the files with attribute of 's':

```

fs0:\> ls -as isabus.efi
Directory of: fs0:\

    06/18/01  10:59p          28,739  IsaBus.efi
           1 File(s)          28,739 bytes
           0 Dir(s)

```

To display all in fs0:\efi directory recursively:

```

fs0:\> ls -r -a efi

```

To search for files with the specified type in the current directory recursively:

```

fs0:\> ls -r -a *.efi -b

```

## Standard-Format Output

The ls command will produce at least two tables: VolumeInfo and FileInfo. The VolumeInfo table reports one row for each file system volume reported. The FileInfo table reports one row for each file, including directories. The following tables describe the standard table column headings and their description. For more information on "Standard-Format Output", see Appendix D.

**Table 22 Is Standard Formatted Output (VolumeInfo)**

Column Number	Description
2	Name. Standard volume label
3	Total Size. Total number of bytes in the volume.
4	Read Only. "True" if the volume is read-only, otherwise "False".
5	Free Space. Total number of free bytes in the volume.
6	Block Size. Nominal block size by which files are typically grown, in bytes.

**Table 23 Is Standard Formatted Output (FileInfo)**

Column Number	Description
2	Name. Complete file name & directory, including the file system's mapped name.
3	Logical Size. Size of the file, in bytes.
4	Physical Size. Size of the file in the volume, including any padding, in bytes.

5	<p>Attributes. List of file attributes. The string can contain zero or more of the following (but no repeats):</p> <ul style="list-style-type: none"> <li>a – Archive</li> <li>d – Directory</li> <li>h – Hidden</li> <li>r – Read-Only</li> <li>s – System</li> </ul>
6	File Creation Time. Time when the file was created, in the format: hh:mm:ss.
7	File Creation Date. Date when the file was created, in the format: dd.mm.yyyy.
8	File Access Time. Time when the file was accessed, in the format: hh:mm:ss
9	File Access Date. Date when the file was accessed, in the format: dd.mm.yyyy
10	File Modification Time. Time when the file was modified, in the format: hh:mm:ss
11	File Modification Date. Date when the file was modified, in the format: dd.mm.yyyy.

## map

### Summary

Defines a mapping between a user-defined name and a device handle.

### Usage

```
map [-d <sname>]
map [-r] [-v] [-c] [-f] [-u] [-t <type[, type...]>] [sname]
map [sname handle | mapping]
```

### Options

*sname*

Mapping Mapped name

*handle*

The number of handle, which is same as dumped from 'dh'

*mapping*

The device's mapped name. Use this parameter to assign a new mapping to a device. The mapping must end with a ':'.

**-sfo**

Output will be formatted according to "Standard-Format Output" below.

**-t**

Shows the device mappings, filtered according to the device type. The supported types are **fp** (floppy), **hd** (hard disk) and **cd** (CD-ROM). Types can be combined by putting a comma between two types. Spaces are not allowed between types.

**-d**

Deletes a mapping

**-r**

Resets to default mappings

**-v**

Lists verbose information about all mappings.

**-c**

Shows the consistent mapping.

**-f**

Shows the normal mapping (not the consistent mapping).

**-u**

This option will add mappings for newly installed devices and remove mappings for uninstalled devices but will not change the mappings of existing devices. The user-defined mappings are also preserved.

## Description

This command creates a mapping between a user-defined name and a device. The most common use of this command is to create a mapped name for devices that support a file system protocol. Once these mappings are created, the names can be used with all the file manipulation commands.

The UEFI Shell environment creates default mappings for all of the devices that support a recognized file system.

This command can be used to create additional mappings, or it can be used to delete an existing mapping with the `-d` option. If the `map` command is used without any parameters, all of the current mappings will be listed. If the `-v` option is used, the mappings will be shown with additional information about each device.

The `-r` option is used to reset all the default mappings in a system; this option is useful if the system configuration has changed since the last boot.

The `-u` option will add mappings for newly installed devices and remove mappings for uninstalled devices but will not change the mappings of existing devices. The user-defined mappings are also preserved. A mapping history will be saved so that the original mapping name is used for a device with a specific device path if that mapping name was used for that device path last time. The current directory is also preserved if the current device is not changed.

Each device in the system has a consistent mapping. If the hardware configuration has not changed, the device's consistent mappings do not change. If two or more machines have the same hardware configurations, the device's consistent mapping will be the same. Use the `-c` option to list all the consistent mappings in the system.

The mapping consist of digits and characters. Other characters are illegal.

This command support wildcards. You can use the wildcards to delete or show the mapping . However, when you assign the mapping, wildcards are forbidden.

## Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

## Standard-Format Output

If `-sfo` is specified, then the `map` command will output a single table: Mappings. The following table describes the table columns for this table:

**Table 24 Standard Formatted Output (Mappings)**

Column Number	Description
2	Mapped Name. The mapped device name.
3	Device Path. The device path which corresponds to the mapped device name.
4	Consistent Name. The consistent mapped name (if any) which is equivalent to <i>MappedName</i> . If <i>MappedName</i> is already a consistent mapped name, then this column is empty.

## md

### Summary

An internal alias for the `mkdir` command.



## mem

### Summary

This is a built-in alias for `dmem`.

## memmap

### Summary

Displays the memory map maintained by the EFI environment.

### Usage

```
memmap [-b]
```

### Options

*-b*

Display one screen at a time

### Description

This command displays the memory map that is maintained by the EFI environment. The EFI environment keeps track all the physical memory in the system and how it is currently being used. The EFI Specification defines a set of Memory Type Descriptors. Please see the EFI Specification for a description of how each of these memory types is used. The following example shows typical output for help on this command.

### Examples

To display the system memory map:

```

fs0:\> memmap

Type      Start      End          # Pages      Attributes
available 000000000750000-0000000001841FFF 0000000000010F2
000000000000009
LoaderCode 0000000001842000-00000000018A3FFF 000000000000062
000000000000009
available 00000000018A4000-00000000018C1FFF 00000000000001E
000000000000009
LoaderData 00000000018C2000-00000000018CAFFF 000000000000009
000000000000009
BS_code    00000000018CB000-0000000001905FFF 00000000000003B
000000000000009
BS_data    0000000001906000-00000000019C9FFF 0000000000000C4
000000000000009
...
RT_data    0000000001B2B000-0000000001B2BFFF 000000000000001
800000000000009
BS_data    0000000001B2C000-0000000001B4FFFF 000000000000024
000000000000009
reserved   0000000001B50000-0000000001D4FFFF 000000000000200
000000000000009

reserved :      512 Pages (2,097,152)
LoaderCode:     98 Pages (401,408)
LoaderData:     32 Pages (131,072)
BS_code :       335 Pages (1,372,160)
BS_data :       267 Pages (1,093,632)
RT_data :       19 Pages (77,824)
available :     4,369 Pages (17,895,424)
Total Memory: 20 MB (20,971,520) Bytes

```

## Standard-Format Output

The standard-format output produced with the `-sfo` option produces two tables: MemoryMap and Summary.

**Table 25 Standard-Format Output for memmap (MemoryMap)**

Column Number	Description
2	Type. <b>Available</b> <b>LoaderCode</b> <b>LoaderData</b> <b>BootServiceCode</b> <b>BootServiceData</b> <b>RuntimeCode</b> <b>RuntimeData</b> <b>Reserved</b>
3	Starting Address
4	Ending Address

5	Number Of 4KB Pages
6	Attributes

**Table 26 Standard-Format Output for memmap (Summary)**

Column Number	Description
2	Total Memory Size (bytes)
3	Reserved Memory Total Size (bytes)
4	Boot Service Code Total Size (bytes)
5	Boot Service Data Total Size (bytes)
6	Runtime Code Total Size (bytes)
7	Runtime Data Total Size (bytes)
8	Loader Code Total Size (bytes)
9	Loader Data Total Size (bytes)
10	Available Total Size (bytes)

## mkdir

### Summary

Creates one or more new directories.

### Usage

```
mkdir dir [dir...]
```

### Options

*dir*

Name of directory or directories to be created. Wildcards are not allowed.

### Description

This command creates one or more new directories. If *dir* includes nested directories, then parent directories will be created before child directories. If the directory already exists, then the command will exit with an error.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<b>SHELL_OUT_OF_RESOURCES</b>	There was insufficient space on the destination to create the requested directory.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_WRITE_PROTECTED</b>	An attempt was made to create a directory when the target media was write-protected.

### Examples

To create a new directory:

```
fs0:\> mkdir rafter
fs0:\> ls
Directory of: fs0:\

    06/18/01  08:05p <DIR>          512  test
    06/18/01  11:14p          r           29  readme.txt
    06/18/01  11:50p <DIR>          512  rafter
           1 File(s)          211 bytes
           2 Dir(s)
```

- To create multiple directories:

```
fs0:\> mkdir temp1 temp2
fs0:\> ls
Directory of: fs0:\

06/18/01  08:05p <DIR>          512  test
06/18/01  11:14p          r          29  readme.txt
06/18/01  11:50p <DIR>          512  rafter
06/18/01  11:52p <DIR>          512  temp1
06/18/01  11:52p <DIR>          512  temp2
          1 File(s)          211 bytes
          4 Dir(s)
```

## mm

### Summary

Displays or modifies MEM/MMIO/IO/PCI/PCIE address space.

### Usage

```
mm address [value] [-w 1|2|4|8] [-MEM | -MMIO | -IO | -PCI | -PCIE] [-n]
```

### Options

*address*

Starting address

*value*

The value to write. If not specified, then the current value will be displayed.

**-MEM**

Memory Address type.

**-IO**

IO Address type

**-PCI**

PCI Configuration Space. The address will have the format 0x000000ssbbddffrr, where ss = Segment, bb = Bus, dd = Device, ff = Function and rr = Register. This is the same format used in the **PCI** command.

**-PCIE**

PCI Express Configuration Space. The address will have the format 0x00000000ssbbddffrr, where ss = Segment, bb = Bus, dd = Device, ff = Function and rrr = Register.

**-w**

Access Width, in bytes. 1 = byte, 2 = 2 bytes, 4 = 4 bytes, 8 = 8 bytes. If not specified, then 1 is assumed.

**-n**

Non-interactive mode.

### Description

This command allows the user to display or modify I/O register, memory contents, or PCI configuration space. The user can specify the start address and the access size they wish to perform using the **Address** parameter and **-w** option. **Address** should be typed in hex value. **-MEM** accesses system memory, **-MMIO** accesses device memory, **-IO** accesses device I/O ports, **-PCI** accesses PCI Configuration Space, and **-PCIE** accesses PCIE Configuration Space. If **-MEM**, **-MMIO**, **-IO**, **-PCI** and **-PCIE** are not specified, then **-MEM** is assumed.

If *Value* is specified which should be typed in hex value, this command will write this value to specified address. Otherwise when this command is executed, the current contents of *Address* are displayed. If *Value* is specified, then `-n` is assumed.

If `-n` is not specified, the command will run in interactive mode and the user has the option of modifying the contents by typing in a hex value. When the user pressed 'ENTER', then next address is displayed. This is continued until the user enters 'q'.

## Examples

To display or modify memory:

```
Address 0x1b07288, default width=1 byte:
```

```
fs0:\> mm 1b07288
MEM 0x0000000001B07288 : 0x6D >
MEM 0x0000000001B07289 : 0x6D >
MEM 0x0000000001B0728A : 0x61 > 80
MEM 0x0000000001B0728B : 0x70 > q
```

```
fs0:\> mm 1b07288
MEM 0x0000000001B07288 : 0x6D >
MEM 0x0000000001B07289 : 0x6D >
MEM 0x0000000001B0728A : 0x80 > *Modified
MEM 0x0000000001B0728B : 0x70 > q
```

To modify memory: Address 0x1b07288, width = 2 bytes:

```
Shell> mm 1b07288 -w 2
MEM 0x0000000001B07288 : 0x6D6D >
MEM 0x0000000001B0728A : 0x7061 > 55aa
MEM 0x0000000001B0728C : 0x358C > q
```

```
Shell> mm 1b07288 -w 2
MEM 0x0000000001B07288 : 0x6D6D >
MEM 0x0000000001B0728A : 0x55AA > *Modified
MEM 0x0000000001B0728C : 0x358C > q
```

To display IO space: Address 80h, width = 4 bytes:

```
Shell> mm 80 -w 4 -IO
IO 0x0000000000000080 : 0x000000FE >
IO 0x0000000000000084 : 0x00FF5E6D > q
```

To modify IO space using non-interactive mode:

```
Shell> mm 80 52 -w 1 -IO
Shell> mm 80 -w 1 -IO
IO 0x0000000000000080 : 0x52 > FE *Modified
IO 0x0000000000000081 : 0xFF >
IO 0x0000000000000082 : 0x00 >
IO 0x0000000000000083 : 0x00 >
IO 0x0000000000000084 : 0x6D >
IO 0x0000000000000085 : 0x5E >
IO 0x0000000000000086 : 0xFF >
IO 0x0000000000000087 : 0x00 > q
```

- To display PCI configuration space, `ss=00, bb=00, dd=00, ff=00, rr=00`:



```
Shell> mm 0000000000 -PCI
PCI 0x0000000000000000 : 0x86 >
PCI 0x0000000000000001 : 0x80 >
PCI 0x0000000000000002 : 0x30 >
PCI 0x0000000000000003 : 0x11 >
PCI 0x0000000000000004 : 0x06 >
PCI 0x0000000000000005 : 0x00 > q
```

These contents can also be displayed by 'PCI 00 00 00'.

To display PCIE configuration space, ss=00, bb=06, dd=00, ff=00, rrr=000:

```
Shell> mm 000600000000 -PCIE
PCIE 0x0000000006000000 : 0xAB >
PCIE 0x0000000006000001 : 0x11 >
PCIE 0x0000000006000002 : 0x61 >
PCIE 0x0000000006000003 : 0x43 >
PCIE 0x0000000006000004 : 0x00 > q
```

## mode

### Summary

Displays or changes the console output device mode.

### Usage

```
mode [col row]
```

### Options

*row*

Number of rows

*col*

Number of columns

### Description

This command is used to change the display mode for the console output device. When this command is used without any parameters, it shows the list of modes that the standard output device currently supports. The mode command can then be used with the **row** and **col** parameter to change the number of rows and columns on the standard output device. The following examples show how the mode command can be used. The first example lists all modes that are currently available, and the current selected mode is indicated by an '\*'. The second example changes the mode to an 80 X 50 text mode display. The display is cleared every time the mode command is used to change the currently selected display mode.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

### Examples

To display all available modes on standard output:

```
Shell> mode
Available modes on standard output
col 80 row 25 *
col 80 row 50
col 80 row 43
col 100 row 100
```

To change the current mode setting:

```
Shell> mode 80 50
Available modes on standard output
col 80 row 25
col 80 row 50 *
col 80 row 43
col 100 row 100
```

## mv

### Summary

Moves one or more files to a destination within a file system.

### Usage

```
mv src [src...] [dst]
```

### Options

*src*

Source file/directory name (wildcards are permitted)

*dst*

Destination file/directory name (wildcards are permitted). If not specified, then the current working directory is assumed to be the destination. If there is more than one argument on the command line, the last one will always be considered the destination.

### Description

This command moves one or more files to a destination within a file system. Moving between filesystem volumes is not supported. If the destination is an existing directory, then the sources are moved into that directory. Otherwise, the sources are moved to the destination, as if the directory has been renamed. If a destination is not specified, the current directory is assumed to be the destination.

Attempting to move a read-only file/directory will result in an error. Moving a directory that contains read-only files is allowed. You cannot move a directory into itself or its subdirectories. You cannot move a directory if the current working directory is itself or its subdirectories.

If an error occurs, the remaining files or directories will still be moved.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<b>SHELL_NOT_FOUND</b>	The source file was not able to be found
<b>SHELL_OUT_OF_RESOURCES</b>	There was insufficient free space to move the requested file to its destination.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_WRITE_PROTECTED</b>	An attempt was made to create a file on media that was write-protected.

## Examples

To rename a file:

```
fs0:\> mv IsaBus.efi Bus.efi
moving fs0:\IsaBus.efi -> \Bus.efi
- [ok]
```

## openinfo

### Summary

Displays the protocols and agents associated with a handle.

### Usage

```
openinfo Handle [-b]
```

### Options

*Handle*

Display open protocol information for specified handle

*-b*

Display one screen at a time

### Description

This command is used to display the open protocols on a given handle. The example below is typical output from help for this command.

**Table 27 Open Protocol Information Layout**

Column Index	Description
1	Agent handle that opens the protocol
2	Controller handle that requires the protocol interface
3	Open count
4	Open type: <b>HandProt</b> , <b>GetProt</b> , <b>TestProt</b> , <b>Child</b> , <b>Driver</b> , <b>Exclusive</b> , <b>DriverEx</b> or <b>Unknown</b>
5	Name of image of the agent if available

### Examples

To show open protocols on handle 0x23:

```
Shell> openinfo 23
Handle 23 (07DEE108)
PciRootBridgeIo
  Drv[1D] Ctrl[23] Cnt(01) Driver Image(PciBus)
  Drv[1D] Ctrl[28] Cnt(01) Child Image(PciBus)
  Drv[1D] Ctrl[29] Cnt(01) Child Image(PciBus)
  Drv[1D] Ctrl[2A] Cnt(01) Child Image(PciBus)
  Drv[1D] Ctrl[2B] Cnt(01) Child Image(PciBus)
  Drv[1D] Ctrl[2C] Cnt(01) Child Image(PciBus)
  Drv[1D] Ctrl[2D] Cnt(01) Child Image(PciBus)
  Drv[1D] Ctrl[2E] Cnt(01) Child Image(PciBus)
  Drv[00] Ctrl[ ] Cnt(01) HandProt
dpath
  Drv[1D] Ctrl[23] Cnt(01) Driver Image(PciBus)
  Drv[00] Ctrl[ ] Cnt(0D) HandProt
```

## parse

### Summary

Command used to retrieve a value from a particular record which was output in a standard formatted output.

### Usage

```
parse filename tablename column [-i <Instance>] [-s <Instance>]
```

### Options

*filename*

Source file name

*tablename*

The name of the table being parsed.

*column*

The one-based column index to use to determine which value from a particular record to parse.

**-i** <Instance>

Start parsing with the nth instance of specified tablename, after the specified instance of ShellCommand. If not present, then all instances will be returned.

**-s** <Instance>

Start parsing with the nth instance of the ShellCommand table. If not present, then 1 is assumed.

### Description

This command will enable the parsing of data from a file which contains data which has been output from a command having used the `-sfo` parameter. Since the standard formatted output has a well known means of parsing, this command is intended to be used as a simplified means of having scripts consume such constructed output files and use this retrieved data in logic of the scripts being written for the UEFI shell.

### Examples

The following data is contained in a temporary file (temp.txt):

```
VolumeInfo, "MikesVolume", "400000000", "32000000", "16000000"  
FileInfo, "fs0:/efi/boot/winloader.efi", "45670", "arsh"  
FileInfo, "fs0:/efi/boot/mikesfile.txt", "1250", "a"  
FileInfo, "fs0:/efi/boot/readme.txt", "795", "a"
```

The following shows the parse command being used:

```
fs0:\> parse temp.txt VolumeInfo 2
MikesVolume
```

Below is an example using the Index parameter:

```
fs0:\> parse temp.txt FileInfo 3 -i 3
795
```

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<b>SHELL_NOT_FOUND</b>	The source file was not able to be found
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation



## pause

### Usage

```
pause [-q]
```

### Description

The `pause` command prints a message to the display and then suspends script file execution and waits for keyboard input. Pressing any key resumes execution, except for `q` or `Q`. If `q` or `Q` is pressed, script processing terminates; otherwise execution continues with the next line after the `pause` command.

The `pause` command is available only in scripts. Switch `-q` can hide the message and it's optional.

### Examples

Following script is a sample of 'pause' command:

```
fs0:\> type pause.nsh
#
# Example script for 'pause' command
#
echo pause.nsh begin..
date
time
pause
echo pause.nsh done.
```

To execute the script with echo on:

```
fs0:\> pause.nsh
+pause.nsh> echo pause.nsh begin..
pause.nsh begin..
+pause.nsh> date
06/19/2001
+pause.nsh> time
00:51:45
+pause.nsh> pause
Enter 'q' to quit, any other key to continue:
+pause.nsh> echo pause.nsh done.
pause.nsh done.
```

To execute the script with echo off:

```
fs0:\> echo -off
fs0:\> pause.nsh
pause.nsh begin..
06/19/2001
00:52:50
Enter 'q' to quit, any other key to continue: q
fs0:\>
```

## pci

### Summary

Displays PCI device list or PCI function configuration space.

### Usage

```
pci [Bus Dev [Func] [-s Seg] [-i]]
```

### Options

*Bus*

Bus number

*Dev*

Device number

*Func*

Function number

-s

Optional segment number *Seg* specified

-i

Information interpreted

### Description

This command will display all the PCI devices found in the system. And it can also display the configuration space of a PCI device according to the specified bus (**Bus**), device (**Dev**), and function (**Func**) addresses. If the function address is not specified, it will default to 0. The **-i** option is used to display verbose information for the specified PCI device. The PCI configuration space for the device will be dumped with a detailed interpretation.

If no parameters are specified all PCI devices will be listed. If the Bus and Device number parameters are specified while the Func or Seg parameters are not, Function or Seg will be set as default value 0.

The '-i' option can be used to display verbose information for the specified PCI device. The PCI configuration space for the specified device will be dumped with a detailed interpretation.

### Examples

To display all PCI devices in the system:

Shell> PCI

Seg	Bus	Dev	Func	
00	00	00	00	==> Bridge Device - Host/PCI bridge
				Vendor 8086 Device 1130 Prog Interface 0
00	00	01	00	==> Bridge Device - PCI/PCI bridge
				Vendor 8086 Device 1131 Prog Interface 0
00	00	1E	00	==> Bridge Device - PCI/PCI bridge
				Vendor 8086 Device 244E Prog Interface 0
00	00	1F	00	==> Bridge Device - PCI/ISA bridge
				Vendor 8086 Device 2440 Prog Interface 0
00	00	1F	01	==> Mass Storage Controller - IDE controller
				Vendor 8086 Device 244B Prog Interface 80
00	00	1F	02	==> Serial Bus Controllers - USB
				Vendor 8086 Device 2442 Prog Interface 0
00	00	1F	03	==> Serial Bus Controllers - System Management Bus
				Vendor 8086 Device 2443 Prog Interface 0
00	00	1F	04	==> Serial Bus Controllers - USB
				Vendor 8086 Device 2444 Prog Interface 0
00	00	1F	05	==> Multimedia Device - Audio device
				Vendor 8086 Device 2445 Prog Interface 0
00	00	1F	06	==> Simple Communications Controllers - Modem
				Vendor 8086 Device 2446 Prog Interface 0
00	01	00	00	==> Display Controller - VGA/8514 controller
				Vendor 1002 Device 5246 Prog Interface 0
00	02	07	00	==> Multimedia Device - Audio device
				Vendor 1274 Device 1371 Prog Interface 0
00	02	0A	00	==> Bridge Device - CardBus bridge
				Vendor 1180 Device 0476 Prog Interface 0
00	02	0A	01	==> Bridge Device - CardBus bridge
				Vendor 1180 Device 0476 Prog Interface 0

To display the configuration space of Bus 0, Device 0, Function 0:

Shell> PCI 00 00 00 -i

```
PCI Segment 00 Bus 00 Device 00 Func 00
00000000: 86 80 30 11 06 00 90 20-02 00 00 06 00 00 00 00 *..0....
.....*
00000010: 08 00 00 20 00 00 00 00-00 00 00 00 00 00 00 00 *...
.....*
00000020: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
00000030: 00 00 00 00 88 00 00 00-00 00 00 00 00 00 00 00
*.....*
00000040: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
00000050: 50 00 09 38 00 00 00 00-00 00 00 00 00 00 00 00
*P..8.....*
00000060: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
00000070: 00 00 18 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
00000080: DE 2C CF 00 00 00 00 00-09 A0 04 F1 00 00 00 00 00
*.....*
00000090: 00 00 D6 FF FE FF 00 00-33 80 33 80 85 84 C4 00
*.....3.3.....*
000000A0: 02 00 20 00 07 02 00 1F-00 00 00 00 00 00 00 00 *..
.....*
000000B0: 00 00 00 00 30 00 00 00-00 00 00 00 00 00 08 00
*.....0.....*
000000C0: 00 00 00 00 00 00 00 00-00 08 00 00 00 00 00 00
*.....*
000000D0: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
000000E0: 00 00 00 00 00 00 00 00-00 00 90 14 00 00 00 00
*.....*
000000F0: 00 00 00 00 74 F8 00 00-00 00 00 00 08 00 00 00
*.....t.....*
```

Vendor ID(0): 8086

Device ID(2): 1130

Command(4): 0006

```
(00)I/O space access enabled: 0 (01)Memory space access enabled:
1
(02)Behave as bus master: 1 (03)Monitor special cycle
enabled: 0
(04)Mem Write & Invalidate enabled: 0 (05)Palette snooping is enabled:
0
(06)Assert PERR# when parity error: 0 (07)Do address/data stepping:
0
(08)SERR# driver enabled: 0 (09)Fast back-to-back
transact....: 0
```

Status(6): 2090

```
(04)New Capabilities linked list: 1 (05)66MHz Capable:
0
(07)Fast Back-to-Back Capable: 1 (08)Master Data Parity Error:
0
(09)DEVSEL timing: Fast (11)Signaled Target Abort:
0
(12)Received Target Abort: 0 (13)Received Master Abort:
1
(14)Signaled System Error: 0 (15)Detected Parity Error:
0
```

Revision ID(8): 02

BIST(0F): Incapable

Cache Line Size(C): 00

Latency Timer(D): 00

Header Type(0E): 0, Single function, PCI device

Class: Bridge Device - Host/PCI bridge -

Base Address Registers(10):

Start	Type	Space	Prefetchable?	Size	Limit
20000000	Mem	32 bits	YES	04000000	24000000

No Expansion ROM(30)

```

Cardbus CIS ptr(28): 00000000
Sub VendorID(2C): 0000 Subsystem ID(2E): 0000
Capabilities Ptr(34): 88
Interrupt Line(3C): 00 Interrupt Pin(3D): 00
Min_Gnt(3E): 00 Max_Lat(3F): 00

```

To display configuration space of Segment 0, Bus 0, Device 0, Function 0:

Shell> PCI 00 00 00 -s 0

```

PCI Segment 00 Bus 00 Device 00 Func 00
00000000: 86 80 30 11 06 00 90 20-02 00 00 06 00 00 00 00 *..0....
.....*
00000010: 08 00 00 20 00 00 00 00-00 00 00 00 00 00 00 00 *...
.....*
00000020: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
00000030: 00 00 00 00 88 00 00 00-00 00 00 00 00 00 00 00
*.....*
00000040: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
00000050: 50 00 09 38 00 00 00 00-00 00 00 00 00 00 00 00
*P..8.....*
00000060: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
00000070: 00 00 18 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
00000080: DE A8 CE 00 00 00 00 00-09 A0 04 F1 00 00 00 00
*.....*
00000090: 00 00 D6 FF FE FF 00 00-33 80 33 80 85 84 C4 00
*.....3.3.....*
000000A0: 02 00 20 00 07 02 00 1F-00 00 00 00 00 00 00 00 *..
.....*
000000B0: 00 00 00 00 30 00 00 00-00 00 00 00 00 00 08 00
*.....0.....*
000000C0: 00 00 00 00 00 00 00 00-00 08 00 00 00 00 00 00
*.....*
000000D0: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
*.....*
000000E0: 00 00 00 00 00 00 00 00-00 00 A0 18 00 00 00 00
*.....*
000000F0: 00 00 00 00 74 F8 00 00-00 00 00 00 08 00 00 00
*.....t.....*

```

## ping

### Summary

Ping the target host with IPv4 stack.

### Usage

```
Ping [-n count] [-l size] TargetIp
```

### Options

**-n**  
Number of echo request datagram to be sent.

**-l**  
Size of data buffer in echo request datagram.

**TargetIp**  
IPv4 address of the target machine.

### Description

This command uses the ICMPv4 ECHO\_REQUEST datagram to elicit ECHO\_REPLY from a host.

### Examples

To ping the target host with 64 bytes data:

```
Shell:\> ping -l 64 192.168.0.1
```

To ping the target host by sending 20 echo request datagram:

```
Shell:\> ping -n 20 202.120.100.1
```

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

## reconnect

### Summary

Reconnects drivers to the specific device.

### Usage

```
reconnect DeviceHandle [DriverHandle [ChildHandle]]
reconnect -r
```

### Options

#### *DeviceHandle*

Device handle (a hexadecimal number)

#### *DriverHandle*

Driver handle (a hexadecimal number). If not specified, all drivers on the specified device will be reconnected.

#### *ChildHandle*

Child handle of device (a hexadecimal number). If not specified, then all child handles of the specified device will be reconnected.

#### **-r**

Reconnect drivers to all devices.

### Description

This command reconnects drivers to the specific device. It will first disconnect the specified driver from the specified device and then connect the driver to the device recursively.

If the **-r** option is used, then all drivers will be reconnected to all devices. Any drivers that are bound to any devices will be disconnected first and then connected recursively. See the **connect** and **disconnect** commands for more details.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

### Examples

To reconnect all drivers to all devices:

```
Shell> reconnect -r
```

To reconnect all drivers to device 0x28:

```
fs0:\> reconnect 28
```

To disconnect 0x17 from 0x28 then reconnect drivers with 0x17 as highest priority to device 0x28:

```
fs0:\> reconnect 28 17
```

To disconnect 0x17 from 0x28 destroying child 0x32 then reconnect drivers with 0x17 as highest priority to device 0x28

```
fs0:\> reconnect 28 17 32
```



## reset

### Summary

Resets the system.

### Usage

```
reset [-w [string]]
reset [-s [string]]
reset [-c [string]]
```

### Options

```
-s      - Performs a shutdown
-w      - Performs a warm boot
-c      - Performs a cold boot
string  - String to be passed to reset service
```

### Description

This command resets the system. The default is to perform a cold reset unless the `-w` parameter is specified. If the reset `string` is specified, then it is passed into the `Reset()` function, so the system can know the reason for the system reset.

### Status Codes Returned

<code>SHELL_INVALID_PARAMETER</code>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
--------------------------------------	---

# rm

## Summary

Deletes one or more files or directories.

## Usage

```
rm [-q] file/directory [file/directory ...]
```

## Options

**-q** Quiet mode; does not prompt user for a confirmation

*file* File name (wildcards are permitted)

*directory* Directory name (wildcards are permitted)

## Description

This command deletes one or more files or directories. If the target is a directory, it will delete the directory, including all its subdirectories. It is not allowed to redirect a file whose parent directory (or the file itself) is being deleted.

Removing a read-only file/directory will result in a failure. Removing a directory containing read-only file(s) will result in a failure. If an error occurs, rm will exit immediately and later files/directories will not be removed.

You cannot remove a directory when the current directory is itself or its subdirectory. If file contains wildcards, it will not ask user for confirmation.

You cannot remove the root directory. You cannot remove the current directory or its ancestor.

## Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_NOT_FOUND</b>	The target file or directory was not able to be found
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_WRITE_PROTECTED</b>	The target media was write-protected.

## Examples

To remove multiple directories at a time:

```
fs0:\> ls test
Directory of: fs0:\test

06/18/01  01:01p <DIR>          512  .
06/18/01  01:01p <DIR>           0  ..
06/19/01  12:59a <DIR>          512  temp1
06/19/01  12:59a <DIR>          512  temp2
          0 File(s)              0 bytes
          4 Dir(s)
```

Error occurs and RM will exit:

```
fs0:\> rm test\temp1 temp2
rm/del: Cannot find 'fs0:\test\temp11' - Not Found
```

To remove multiple directories with wildcards:

```
fs0:\> rm test\temp*
rm/del: Remove subtree 'fs0:\test\temp1' [y/n]? y
removing fs0:\test\temp1\temp1.txt
- [ok]
removing fs0:\test\temp1\boot\nshell.efi
- [ok]
removing fs0:\test\temp1\boot
- [ok]
removing fs0:\test\temp1
- [ok]
rm/del: Remove subtree 'fs0:\test\temp2' [y/n]? y
removing fs0:\test\temp2\temp2.txt
- [ok]
removing fs0:\test\temp2
- [ok]
```

Removing a directory that contains a read-only file will fail:

```
fs0:\> attrib +r test\temp1\readme.txt
A R fs0:\test\temp1\readme.txt

fs0:\> rm test\temp1
rm/del: Cannot open 'readme.txt' under 'fs0:\test\temp1' in
writable mode
- [error] - Access Denied
Exit status code: Access Denied
```

## sermode

### Summary

Sets serial port attributes.

### Usage

```
sermode [handle [baudrate parity databits stopbits]]
```

### Options

#### *handle*

Device handle for a serial port in hexadecimal. The **dh** command can be used to retrieve the right handle.

#### *baudrate*

Baud rate for specified serial port. The following values are supported: 50, 75, 110, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600(default), 19200, 38400, 57600, 115200, 230400, and 460800. All other values will be converted to the next highest setting.

#### *parity*

Parity bit settings for specified serial port. Any one of the following:

- d** - Default parity
- n** - No parity
- e** - Even parity
- o** - Odd parity
- m** - Mark parity
- s** - Space parity

#### *databits*

Data bits for the specified serial port. The following settings are supported: 4, 7, 8 (default). All other settings are invalid.

#### *stopbits*

Stop bits for the specified serial port. The following settings are supported:

- 0** (0 stop bits - default setting)
- 1** (1 stop bit)
- 2** (2 stop bits)
- 15** (1.5 stop bits)

**Note:** *All other settings are invalid.*

## Description

This command displays or sets baud rate, parity attribute, data bits and stop bits of serial ports. If no attributes are specified, then the current settings are displayed. If no handle is specified, then all serial ports are displayed.

## Examples

To display the settings for all serial port devices:

```
Shell> sermode
4F06B08 - (115200, N, 8, 1)
4F05F88 - (115200, N, 8, 1)
```

To display the settings for the serial port device whose handle is 0x6B:

```
Shell> sermode 6B
4F06B08 - (115200, N, 8, 1)
```

To configure the serial port settings for handle 0x6B to 9600bps, even parity, 8 data bits, and 1 stop bit:

```
Shell> sermode 6B 9600 e 8 1
sermode: Mode set on handle 04F06B08
```

## set

### Summary

Displays, changes or deletes a UEFI Shell environment variables.

### Usage

```
set [-v] [sname [value]]
set [-d <sname>]
```

### Options

**-d**  
Deletes the environment variable

**-v**  
Volatile variable

**sname**  
Environment variable name

**value**  
Environment variable value

### Description

This command is used to maintain the UEFI Shell environment variables. This command can do the following:

- Display the environment variables.
- Create new environment variables.
- Change the value of existing environment variables.
- Delete environment variables.

The **set** command will set the environment variable that is specified by **sname** to **value**. This command can be used to create a new environment variable or to modify an existing environment variable.

If the **set** command is used without any parameters, then all the environment variables are displayed. If the **set** command is used with the **-d** option, then the environment variable that is specified by **sname** will be deleted.

**Note:** This command does not change the value of the environment variable **lasterror**.

### Status Codes Returned

SHELL_SUCCESS	The action was completed as requested.
SHELL_SECURITY_VIOLATION	This function was not performed due to a security violation

SHELL_OUT_OF_RESOURCES	A request to set a variable in a non-volatile fashion could not be completed. The resulting non-volatile request has been converted into a volatile request.
------------------------	--

## Examples

To add an environment variable:

```
Shell> set DiagnosticPath fs0:\efi\diag;fs1:\efi\diag
```

To display environment variables:

```
Shell> set
* path : .
  diagnosticPath : fs0:\efi1.1\diag;fs1:\efi1.1\diag
```

To delete an environment variable:

```
Shell> set -d diagnosticpath
Shell> set
* path : .
```

To change an environment variable:

```
fs0:\> set src efi
fs0:\> set
* path : .;fs0:\efi\tools;fs0:\efi\boot;fs0:\
  src : efi
fs0:\> set src efi1.1
fs0:\> set
* path : .;fs0:\efi\tools;fs0:\efi\boot;fs0:\
  src : efi1.1
```

To append an environment variable:

```
Shell> set
* path : .
Shell> set path %path%;fs0:\efi\tools;fs0:\efi\boot;fs0:\
Shell> set
* path : .;fs0:\efi\tools;fs0:\efi\boot;fs0:\
```

To set a volatile variable that will disappear at the next

```
boot:
Shell> set -v EFI_SOURCE c:\project\EFI1.1
Shell> set
* path : .;fs0:\efi\tools;fs0:\efi\boot;fs0:\
* EFI_SOURCE : c:\project\EFI1.1
```

## setsize

### Summary

Adjusts the size of a file.

### Usage

```
setsize size file [file...]
```

### Options

**file**

The file or files which will have its size adjusted.

**size**

The desired size of the file once it is adjusted. Setting the size smaller than the actual data contained in this file will truncate this data.

### Description

This command adjusts the size of a particular target file. When adjusting the size of a file, it should be noted that it will automatically truncate or extend the size of a file based on the passed in parameters. If the file does not exist, it will be created.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_VOLUME_FULL</b>	The media has insufficient space to complete the request.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.



## setvar

### Summary

Changes the value of a UEFI variable.

### Usage

```
setvar variable-name [-guid guid] [-bs] [-rs] [-nv] [=data]
```

### Options

*variable-name*

The name of the UEFI variable to modify or display.

*-guid*

Specifies the GUID of the UEFI variable to modify or display. If not present, then the GUID `EFI_GLOBAL_VARIABLE`, as defined in chapter 3.2 of the UEFI specification.

*-bs*

Indicates that the variable is a boot variable. Should only be present for new variables, otherwise it is ignored.

*-rt*

Indicates that the variable is a runtime variable. Should only be present for new variables, otherwise it is ignored.

*-nv*

Indicates that the variable is non-volatile. If not present, then the variable is assumed to be volatile. Should only be present for new variables, otherwise it is ignored.

*=data*

New data for the variable. If there is nothing after the '=' then the variable is deleted. If = is not present, then the current value of the variable is dumped as hex bytes. The *data* may consist of zero or more of the following:

*xx[xx]:*

Hexadecimal bytes

*"ascii-string":*

ASCII-string with no null-terminator

*L"UCS2-string":*

UCS-2 encoded string with no null-terminator

*--device*

Device path text format, as specified by the EFI Device Path Display Format Overview section of the UEFI 2.1 specification.

## Description

This command changes the UEFI variable specified by name and guid. If = is specified, but *data* is not, the variable is deleted, if it exists. If = is not specified, then the current variable contents are displayed. If =data is specified, then the variable's value is changed to the value specified by *data*.

**-bs**, **-rt** and **-nv** are only useful if the variable already exists. If the variable already exists and the attributes cannot be changed, then -1 is returned.

## shift

### Usage

`shift`

### Description

The `shift` command shifts the contents of a UEFI Shell script's positional parameters so that `%1` is discarded, `%2` is copied to `%1`, `%3` is copied to `%2`, `%4` is copied to `%3` and so on. This allows UEFI Shell scripts to process script parameters from left to right.

**Note:** This command does not change the UEFI shell environment variable `lasterror`.

**Note:** The `shift` command is available only in UEFI Shell scripts.

### Examples

Following script is a sample of 'shift' command:

```
fs0:\> type shift.nsh
#
# Example script for 'shift' command
#
echo %1 %2 %3
shift
echo %1 %2
```

To execute the script with echo on:

```
fs0:\> shift.nsh welcome EFI world
shift.nsh> echo welcome EFI world
welcome EFI world
shift
echo EFI world
EFI world
```

To execute the script with echo off:

```
fs0:\> echo -off
fs0:\> shift.nsh welcome EFI world
welcome EFI world
EFI world
```

## smbiosview

### Summary

Displays SMBIOS information.

### Usage

```
smbiosview [-t SmbiosType] | [-h SmbiosHandle] | [-s] | [-a]
```

### Options

**-t**

Display all structures of *SmbiosType*. The following values are supported:

- 0 - BIOS Information
- 1 - System Information
- 3 - System Enclosure
- 4 - Processor Information
- 5 - Memory Controller Information
- 6 - Memory Module Information
- 7 - Cache Information
- 8 - Port Connector Information
- 9 - System Slots
- 10 - On Board Devices Information
- 15 - System Event Log
- 16 - Physical Memory Array
- 17 - Memory Device
- 18 - 32-bit Memory Error Information
- 19 - Memory Array Mapped Address
- 20 - Memory Device Mapped Address
- 21 - Built-in Pointing Device
- 22 - Portable Battery
- 34 - Management Device
- 37 - Memory Channel
- 38 - IPMI Device Information
- 39 - System Power Supply

**-h**

Display the structure of *SmbiosHandle*, the unique 16-bit value assigned to each SMBIOS structure. *SmbiosHandle* can be specified in either decimal or hexadecimal format. Use the 0x prefix for hexadecimal values.

**-s**

Display statistics table.

**-a**

Display all information.

### Description

This command displays the SMBIOS information. Users can display the information of SMBIOS structures specified by type or handle.



## stall

### Summary

Stalls the operation for a specified number of microseconds.

### Usage

```
stall time
```

### Options

```
time
```

The number of microseconds for the processor to stall.

### Description

This command would be used to establish a timed stall of operations during a script.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_DEVICE_ERROR</b>	There was a hardware error associated with this request.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

## time

### Summary

Displays or sets the current time for the system.

### Usage

```
time [hh:mm[:ss]] [-tz tz] [-d dl]
```

### Options

**hh**

New hour (0-23) (required)

**mm**

New minute (0-59) (required)

**ss**

New second (0-59) If not specified, then zero will be used.

**-tz**

Timezone adjustment, measured in minutes offset from GMT. Valid values can be between -1440 and 1440 or 2047. If not present or set to 2047, time is interpreted as local time.

**-d**

Indicates that time is not affected by daylight savings time (0), time is affected by daylight savings time but time has not been adjusted (1), or time is affected by daylight savings time and has been adjusted (3).. All other values are invalid. If no value follows -d, then the current daylight savings time will be displayed.

### Description

This command displays or sets the current time for the system. If no parameters are used, it shows the current time. If valid hours, minutes, and seconds are provided, then the system's time will be updated.

Note the following rules:

Except for numeric characters and the : character, all other characters in the argument are invalid. The Shell will report an error if the number is in the wrong hour/minute/second range.

Spaces before or after the numeric character are not allowed. Spaces inserted into the number are not allowed either.

Repeated zeros are allowed before the number. For example:

```
Shell > time 00000017:000004:0000
Shell > time
17:04:00 (GMT+08:00)
Shell >
```

The seconds parameter is optional. If there is no seconds number, it will set to zero by default.

## Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_DEVICE_ERROR</b>	There was a hardware error preventing the completion of this command
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

## Examples

```
Shell > time 17:23
Shell > time
17:23:00 (GMT+08:00)
Shell > time 17:23:
Shell > time
17:23:00 (GMT+08:00)
```

To display current time:

```
fs0:\> time
16:51:03 (GMT+08:00)
```

To set the system time:

```
fs0:\> time 9:51:30
fs0:\> time
09:51:31 (GMT+08:00)
```

To get the time, including daylight savings time:

```
fs0:\> time 9:51:30
fs0:\> time -d
09:51:31 (GMT+08:00) DST: Not Affected
```



## timezone

### Summary

Displays or sets time zone information.

### Usage

```
timezone [-s hh:mm | -l] [-b] [-f]
```

### Options

- `-s`  
Set time zone associated with `hh:mm` offset from GMT
- `-l`  
Display list of all time zones
- `-b`  
Display one screen at a time
- `-f`  
Display full information for specified timezone

### Description

This command displays and sets the current time zone for the system. If no parameters are used, it shows the current time zone. If a valid `hh:mm` parameter is provided, then the system's time zone information will be updated.

### Examples

To display all available time zones:

```
Shell> timezone -l
GMT-12:00, International Date Line West
GMT-11:00, Midway Island, Samoa
GMT-10:00, Hawaii
GMT-09:00, Alaska
GMT-08:00, Pacific Time(US & Canada); Tijuana
GMT-07:00, Arizona, Chihuahua, La Paz, Mazatlan
GMT-06:00, Central America, Central Time(US & Canada)
GMT-05:00, Bogota, Lima, Quito, Eastern Time(US & Canada)
GMT-04:00, Atlantic Time(Canada), Caracas, Santiago
GMT-03:30, Newfoundland
GMT-03:00, Brasilia, Buenos Aires, Georgetown, Greenland
GMT-02:00, Mid-Atlantic
GMT-01:00, Azores, Cape Verde Is.
GMT, Greenwich Mean Time, Casablanca, Monrovia, Dublin, London
GMT+01:00, Amsterdam, Berlin, Bern, Rome, Paris, West Central Africa
GMT+02:00, Athens, Istanbul, Bucharest, Cairo, Jerusalem
GMT+03:00, Baghdad, Kuwait, Riyadh, Moscow, Nairobi
GMT+03:30, Tehran
GMT+04:00, Abu Dhabi, Muscat, Baku, Tbilisi, Yerevan
GMT+04:30, Kabul
GMT+05:00, Ekaterinburg, Islamabad, Karachi, Tashkent
GMT+05:30, Chennai, Kolkata, Mumbai, New Delhi
GMT+05:45, Kathmandu
GMT+06:00, Almaty, Novosibirsk, Astana, Dhaka, Sri Jayawardenepura
GMT+06:30, Rangoon
GMT+07:00, Bangkok, Hanio, Jakarta, Krasnoyarsk
GMT+08:00, Beijing, Chongqing, Hong Kong, Urumqi, Taipei, Perth
GMT+09:00, Osaka, Sapporo, Tokyo, Seoul, Yakutsk
GMT+09:30, Adelaide, Darwin
GMT+10:00, Canberra, Melbourne, Sydney, Guam, Hobart, Vladivostok
GMT+11:00, Magadan, Solomon Is., New Caledonia
GMT+12:00, Auckland, Wellington, Fiji, Kamchatka, Marshall Is.
GMT+13:00, Nuku'alofa
```

To set the time zone:

```
Shell> timezone -s -7:00
Shell> timezone
GMT-07:00

Shell> timezone -s 5:00
Shell> timezone
GMT+05:00
```

To display detailed information for the current time zone:

```
Shell> timezone -f
GMT+05:00, Ekaterinburg, Islamabad, Karachi, Tashkent
Shell> timezone
GMT+05:00
```

## touch

### Summary

Updates the time and date on a file to the current time and date.

### Usage

```
touch [-r] file [file ...]
```

### Options

*file*

The name or pattern of the file or directory. There can be multiple files on the command-line.

**-r**

Recurse into subdirectories

### Description

This command updates the time and date on the file that is specified by the **file** parameter to the current time and date.

If multiple files are specified on the command line, it will continue processing. It will touch the files one by one and errors will be ignored.

Touch cannot change the time and date of read-only files and directories.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_NOT_FOUND</b>	The target file or set of files were not found.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_WRITE_PROTECTED</b>	The media was write-protected or the file had a read-only attribute associated with it.

## type

### Summary

Sends the contents of a file to the standard output device.

### Usage

```
type [-a|-u] file [file...]
```

### Options

*file*

Name of the file to display.

**-a**

Display the file as if it is encoded as 8-bit ASCII

**-u**

Displays the file as if it were encoded as UCS-2 Unicode.

### Description

This command sends the contents of a file to the standard output device. If no options are used, then the command attempts to automatically detect the file type. If it fails, then UCS-2 is presumed.

If the **-a** option is specified, the file is sent to the standard output device as a stream of ASCII characters.

If the **-u** option is specified, the file is sent to the standard output device as a stream of Unicode (UCS-2) characters.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_NOT_FOUND</b>	The target file or set of files were not found.

### Examples

To display a file in UCS-2 format:

```
fs0:\> type -u pause.nsh
#
# Example script for 'pause' command
#
echo pause.nsh begin..
date
time
pause
echo pause.nsh done.
```

```
To display multiple files:
fs0:\> type test.*
How to Install?
time
stall 3000000
time
```

## unload

### Summary

Unloads a driver image that was already loaded.

### Usage

```
unload [-n] [-v|-verbose] Handle
```

### Options

**-n**

Skips all prompts during unloading, so that it can be used in a script file.

**-v, -verbose**

Dump verbose status information before the image is unloaded.

*Handle*

Handle of driver to unload, always taken as hexadecimal number

### Description

This command unloads a driver image that was already loaded and which supports the unloading option (see [EFI\\_LOADED\\_IMAGE\\_PROTOCOL's Unload\(\)](#) member.)

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.

## ver

### Summary

Displays the version information for the UEFI Shell and the underlying UEFI firmware.

### Usage

```
ver [-s|-terse]
```

### Options

```
-s  
    Displays only the UEFI Shell version  
-terse
```

### Description

This command displays the version information for this EFI Firmware or the version information for the UEFI Shell itself. The information is retrieved through the EFI System Table or the Shell image.

```
UEFI <support-level> Shell v<uefi-shell-version>  
shell-supplier-specific-data  
UEFI v<uefi-firmware-version> (<firmware vendor name>, 0x<firmware vendor  
version as 32-bit hex value> <optional additional vendor version>)
```

```
UEFI Basic Shell v2.0  
Build 8192. Copyright 2008 by Intel(R) Corporation.  
UEFI v2.10 Firmware (Phoenix Technologies Ltd., 0x01014318)
```

```
<support-level>  
    0 = Minimal, 1 = Scripting, 2 = Basic, 3 = Interactive
```

```
<uefi-shell-version>  
    comes from the shell spec upon which the shell implementation is based.
```

```
<shell-supplier-specific-data>  
    Build, copyright, etc.
```

```
<uefi-firmware-version>  
    From the EFI System Table's Hdr.Revision field, formatted as two period  
    delimited fields. The first field is the upper 16-bits of this field, represented as a  
    decimal unsigned integer. The second field is the lower 16-bites of this field,  
    represented as a two-digit, zero-filled decimal unsigned integer.
```

```
<firmware vendor name>  
    From the EFI System Table's FirmwareVendor field
```

*<firmware vendor version>*

From the EFI System Table's FirmwareRevision field

### Status Codes Returned

SHELL_SUCCESS	The action was completed as requested.
---------------	--



## vol

### Summary

Displays the volume information for the file system that is specified by **fs**.

### Usage

```
vol [fs] [-n <VolumeLabel>]
vol [fs] [-d]
```

### Options

*fs*

The name of the file system.

*VolumeLabel*

The name of the file system. The following characters cannot be used: % ^ \* + = [ ] | : ; " < > ? / . No spaces are allowed in the volume label.

*-d*

Empty volume label.

### Description

This command displays the volume information for the file system that is specified by *fs*. If *fs* is not specified, the current file system will be used. If *-n* is specified, then the volume label for *fs* will be set to *VolumeLabel*. The maximum length for *VolumeLabel* is 11 characters.

### Status Codes Returned

<b>SHELL_SUCCESS</b>	The action was completed as requested.
<b>SHELL_INVALID_PARAMETER</b>	One of the passed in parameters was incorrectly formatted or its value was out of bounds.
<b>SHELL_SECURITY_VIOLATION</b>	This function was not performed due to a security violation
<b>SHELL_NOT_FOUND</b>	The target file-system was not found

### Examples

To display the volume of the current fs:

```
fs0:\> vol
Volume has no label (rw)
1,457,664 bytes total disk space
1,149,440 bytes available on disk
512 bytes in each allocation unit
```

To change the label of fs0:

```
shell> vol fs0 -n help_test
Volume HELP_TEST (rw)
1,457,664 bytes total disk space
1,149,440 bytes available on disk
512 bytes in each allocation unit
```

To get rid of the label of fs0:

```
fs0:\> vol fs0 -d
Volume has no label (rw)
1,457,664 bytes total disk space
220,160 bytes available on disk
512 bytes in each allocation unit
```

# Appendix A

## UEFI Shell Consistent Mapping Design

---

This appendix describes how device assignments are created.

### A.1 **Requirement:**

1. The shell shall support consistent device assignments across (through) reboots.  
e.g. same concept as how you assign the letter D: to a partition under DOS/Windows.
2. The shell commands shall support defining mappings.  
For identical machines with the same hardware configurations the mapping result should always be the same.
3. Not use the NV storage.  
In the OS, it is easy to implement the consistent mapping, because it can store the mapping info on the hard disk or other storage. The firmware has no large storage to store all of this system info. To save space, it is strongly desired that such mapping data does not use NV storage to maintain this data.

### A.2 **Design**

#### A.2.1 **What does consistent mapping mean?**

If hardware configuration is not changed, the mappings should not change. EXAMPLE: map -r, reboot, map -r will not change the mappings.

If two or more machines have the same hardware configurations, mapping result should be the same.

#### A.2.2 **Hardware configuration change:**

Generally, buses, controllers, hubs or bridges changing mean hardware configuration change.

The change of the number or physical connection of hardware which can have child hardware devices will be considered as hardware configuration change.

## Example:

### A.2.2.1 *Hardware configuration change example*

1. Change IDE disk from IDE primary master to slave
2. Change USB device from port0 to port1
3. Add or remove a SCSI controller adapt card

Formatted: Bullets and Numbering

### A.2.2.2 *Hardware configuration not change example*

1. Remove floppy/cdrom disk in drive will not affect mapped names of other existing names
2. Remove floppy/cdrom, then insert back, the newly mapped name will be the same as the last time it was mapped.
3. Unplugging a usb device will not affect mapped names of other existing names
4. Unplug usb device, then plug back to the same port, the newly mapped will be the same as the last time it was mapped.

Formatted: Bullets and Numbering

### A.2.3 *Mapping generated from device path*

The device path is used to generate the mapping, because in a platform, the device path is unique and if the hardware configuration doesn't change, the device's device path doesn't change.

### A.2.4 *Consistent Mapping*

A consistent mapping consists of 3 parts:

<MTD><HI><CSD>

MTD(Media Type Descriptor): A string carries device's media info (harddisk, CD-ROM, ...etc.)

1. Auxiliary name of media type, determined by device path
2. Matches with EFI device path specification
3. Proposed name (hd for harddisk, cd for CD-ROM, fp for floppy, etc.)

Formatted: Bullets and Numbering

HI(Hardware Index): The index of the hardware device path node described in current device path. The index is determined by the position of the whole sorted hardware device path node in system.

1. Extract the hardware device path node and ACPI device path node from each device path in system, make a condensed device path.

2. Use certain algorithm to sort all the condensed device paths

3. Adding/Removing controller(s) would change the index (hardware configuration change)

Formatted: Bullets and Numbering

CSD(Connection Specific Descriptor): A string of numbers and characters, which identifies how the device connects to parent controller.

1. Specify the connection of device

2. use one or several numbers or characters to describe each media and messaging device path node in device path to specify the connection

Formatted: Bullets and Numbering

## A.2.5

### *Example (USB Devices)*

Hardware Configuration (USB part)

4 UHC

7 USB devices: 4 Hard disks, 3 hub

Other hardware are ignored when mapping usb subsystem

Device Path (controllers are underlined)

```
acpi(pnp0a03,0)/pci(1d,0)/usb(0,1)/usb(1,1)/HD(Part4, sigxxx)
acpi(pnp0a03,0)/pci(1d,3)/usb(0,1)/usb(5,1)/HD(Part2, sigxxx)
acpi(pnp0a03,0)/pci(1d,1)/usb(1,0)/HD(Part3, sigxxx)
acpi(pnp0a03,0)/pci(1d,2)/usb(1,0)/usb(2,0)/HD(Part1, sigxxx)
```

Steps to determine the consistent mapping for devices:

1. Determine the MTD

2. Determine the HI

3. Determine the CSD

4. Make the final mapping

Formatted: Bullets and Numbering

### *A.2.5.1*

#### *Step 1: Determine MTD*

MTD for all devices:

- hd is defined for harddisk.
- cd is defined for CD-ROM.
- fp is defined for floppy.
- f is defined for unknown device.

...

More names will be added according to the evolution of UEFI specification

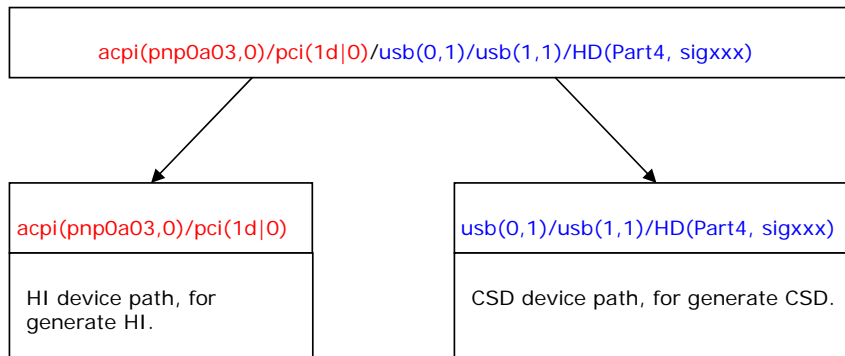
The 4 USB devices are the hard disks, so their MTDs are "hd".

### A.2.5.2 Step 2: Determine HI

HI is determined by ACPI device path node and hardware device path node

1. Algorithm to calculate the HI
2. Enumerate all device path exist in system, whether there is a file system on it or not.
3. Split the device path: the hardware and ACPI device path node part consists of the HI device path that generates the HI and the remain device path consist the CSD device path which generate the CSD.

Formatted: Bullets and Numbering



Each kind of device path node has its own compare algorithm.

For acpi device path node, the compare algorithm is:

`acpi(h1, u1) > acpi(h2, u2) only if h1 > h2 or (h1 = h2 and u1 > u2)`

For pci device path node, the compare algorithm is:

`pci(d1, f1) > pci(d2, f2) only if d1 > d2 or (d1 = d2 and f1 > f2)`

For example:

`acpi(pnp0a03,0)/pci(1d|0) <...< acpi(pnp0a03,0)/pci(1d|1)<...<  
acpi(pnp0a03,0)/pci(1d|2) <...< acpi(pnp0a03,0)/pci(1d|3)`

Use decimal number for HI (0, 1, 2, 3, ...)

After sorting we can assign HI for each hardware device path node part:

```

UHC1 acpi(pnp0a03,0)/pci(1d|0) - 5
UHC2 acpi(pnp0a03,0)/pci(1d|1) - 8
UHC3 acpi(pnp0a03,0)/pci(1d|2) - 12
UHC4 acpi(pnp0a03,0)/pci(1d|3) - 20

```

### A.2.5.3 Step 3: Determine CSD

For each kind device path node, there are rules to translate the device path node to the CSD.

#### A.2.5.3.1 Rules for USB device path node

Use interface number and port number for usb node

Numbers in device path will be mapped at intervals to characters or numbers

When mapping from numbers to characters: use a – 0 , b – 1, ..., z – 25

Sample:

```
usb(0,1) → a1 or 0b.
```

#### A.2.5.3.2 Rules for hard drive device path node

Use partition number for hard drive device node

Sample:

```
HD(Part4, sigxxx) → e or 4.
```

The CSD device paths of the 4 harddisk in our sample are:

```

usb(0,1)/usb(1,1)/HD(Part4, sigxxx)
usb(0,1)/usb(5,1)/HD(Part2, sigxxx)
usb(1,0)/HD(Part3, sigxxx)
usb(1,0)/usb(2,0)/HD(Part1, sigxxx)

```

#### A.2.5.3.3 Corresponding CSDs

```

usb(0,1)/usb(1,1)/HD(Part4, sigxxx) → a1b1e
usb(0,1)/usb(5,1)/HD(Part2, sigxxx) → a5e1c
usb(1,0)/HD(Part3, sigxxx) → b0d
usb(1,0)/usb(2,0)/HD(Part1, sigxxx) → b0c0b

```

#### A.2.5.3.4 Step 4: Make the whole mapping

Whole mapping rule:

```

<USB mapping > ::= [<MTD>] <HI> <CSD>
<MTD> ::= {hd, fp, cd..}
<HI> ::= {0, 1, 2, ...}
<CSD> ::= <node sequence>

```

Put the three parts (MTD, HI and CSD) together to get final mapping for the four hard disks

```

acpi(pnp0a03,0)/pci(1d,0)/usb(0,1)/usb(1,1)/HD(Part4, sigxxx) → hd5a1b1e
acpi(pnp0a03,0)/pci(1d,3)/usb(0,1)/usb(5,1)/HD(Part2, sigxxx) → hd8a1d1c
acpi(pnp0a03,0)/pci(1d,1)/usb(1,0)/HD(Part3, sigxxx) → hd12b0d
acpi(pnp0a03,0)/pci(1d,2)/usb(1,0)/usb(2,0)/HD(Part1, sigxxx) →
hd20b0c0b

```

A.3

## Implementation

Consistent Mapping = <MTD><HI><CSD>

### MTD

<b>fp</b>	floppy
<b>hd</b>	hard disk
<b>cd</b>	cd rom
<b>f</b>	unknown media

### HI

1. Extract the hardware device path node and ACPI device path node from each device path in system,
2. Extract the condensed HI device path
3. Sort the extracted HI device paths.

Formatted: Bullets and Numbering

HI is index of the sorted HI device paths.

### CSD

Use one or several numbers or characters to describe each media and messaging device path node in device path to specify the connection.

All of the reference values in the table below are references to the device path type and sub-type values in the UEFI specification. For actual values of each of the corresponding Type and Sub-type values, please refer to the UEFI spec.

**Table 28 How to process each type the device path node:**

Type	Sub Type	Node Type	Note
HARDWARE_DEVICE	HW_PCI	HI Node	Used to get HI
HARDWARE_DEVICE	HW_PCCARD	HI Node	Used to get HI
HARDWARE_DEVICE	HW_MEMMAP	HI Node	Used to get HI
HARDWARE_DEVICE	HW_VENDOR	HI Node	Used to get HI
HARDWARE_DEVICE	HW_CONTROLLER	HI Node	Used to get HI
ACPI_DEVICE	ACPI	HI/CSD Node	Used ACPI(0604) to get HI and other to get CSD.



MESSAGING_DEVICE	MSG_ATAPI	CSD Node	IDE channel index (0 for primary, master, 3 for secondary slave)
MESSAGING_DEVICE	MSG_SCSI	CSD Node	Use LUN number and PUN number for SCSI node
MESSAGING_DEVICE	MSG_FIBRECHANNEL	CSD Node	Use WWW number and LUN number for fibre channel device node
MESSAGING_DEVICE	MSG_1394	CSD Node	Use GUID as CSD for 1394 device path node
MESSAGING_DEVICE	MSG_USB	CSD Node	Use interface number and port number for USB node
MESSAGING_DEVICE	MSG_USB_CLASS	NONE	Not process.
MESSAGING_DEVICE	MSG_I2O	CSD Node	Use Target ID as I2o Device Path
MESSAGING_DEVICE	MSG_MAC_ADDR	CSD Node	Use MAC address as CSD for MAC node
MESSAGING_DEVICE	MSG_IPv4	CSD Node	Use local IP address, local port, Remote IP address, remote port for IPv4 node
MESSAGING_DEVICE	MSG_IPv6	CSD Node	Use local IP address, local port, Remote IP address, remote port for IPv6 node
MESSAGING_DEVICE	MSG_INFINIBAND	CSD Node	Use PORT GUID, IOC GUID, Target Port ID, Device ID
MESSAGING_DEVICE	MSG_UART	CSD Node	Use Baud Rate, Data Bits, Parity, Stop Bits as CSD
MESSAGING_DEVICE	MSG_VENDOR	CSD Node	Use GUID as CSD
MESSAGING_DEVICE	MSG_LUN	CSD Node	Use the Logical Unit Number
MESSAGING_DEVICE	MSG_SATA	CSD Node	Use the HBA Port Number, Port Multiplier, and LUN value.
MESSAGING_DEVICE	MSG_SAS	CSD Node	Use the SAS Address, LUN, Device Topology, and Relative Target Port
MEDIA_DEVICE	MEDIA_HARDDRIVE	CSD Node	Partition Number
MEDIA_DEVICE	MEDIA_CDROM	CSD Node	Boot Entry
MEDIA_DEVICE	MEDIA_VENDOR	CSD Node	Use Vendor_GUID as CSD
MEDIA_DEVICE	MEDIA_FILEPATH	NONE	Don't process.
MEDIA_DEVICE	MEDIA_PROTOCOL	NONE	Don't process.
MEDIA_DEVICE	MEDIA_FV_FILEPATH	NONE	Don't process.

BBS_DEVICE	BBS_BBS	NONE	Don't process.
------------	---------	------	----------------

### A.3.1

#### **Get the MTD**

The MTD is a string that carries device's media info. Such as floppy, hard disk or others. The MTD is a string that makes the mapping more readable.

Now, the MTD has four types: fp(floppy), hd(hard disk), cd(CD Rom) and f(unknown type).

The MTD's info come from the device path. the algorithm go through each node of the device path to find the special node that can specify this device's MTD.

**Table 29 MTD Naming**

TYPE	SUBTYPE	MTD
EDIA_DEVICE_PATH	MEDIA_HARDDRIVE	hd
EDIA_DEVICE_PATH	MEDIA_CDROM	cd
ACPI_DEVICE_PATH	ACPI &HID=0x0604	fp

**Note:** If a device has the MEDIA\_HARDDRIVE and MEDIA\_CDROM device path node, then its MTD is cd.

If a device path has no any node list on the table, then, its MTD is f(unknown).

### A.3.2

#### **Get the HI**

Each device can be separated into two part: the HI part and CSD part.

The HI part is used to get the HI section in the consistent mapping.

The CSD part is used to get the CSD section in the consistent mapping.

The algorithm of get HI goes through each device path in the system and extracts each HI node in the device path according Table 28 to create a HI device path.

### A.3.3

#### **Get the CSD**

A device path remove the HI device path part, the remain part is the CSD part.

The algorithm is go through the CSD device part, to each node, according Table 28 get the data that will be add to the CSD, translate the data to a string of digital or character(according the position in the CSD).

#### **A.3.3.1**

##### **USB Devices**

CSD of hard drive device path node

Use partition number and interface for CSD

#### A.3.3.1.1 *Example*

`usb(0,1) → a1`  
`usb(1,0) → b0`

#### A.3.3.2 *Floppy Devices*

General consistent mapping rule  
Use `_UID` for CSD

#### A.3.3.2.1 *Examples:*

1. `acpi(pnp0604,0) → a`
2. `acpi(pnp0604,1) → b`
3. \_\_\_\_\_ IDE Devices

← - - - Formatted: Bullets and Numbering

#### A.3.3.3 *CSD of ATAPI device path node*

ATAPI node → ide channel index  
0 for primary, master, 3 for secondary slave

#### A.3.3.4 *CSD of LUN Device path node*

Use the LUN number

#### A.3.3.4.1 *Examples:*

`USB(0x2,0x1)/Unit(0x0) → c10`  
`USB(0x2,0x1)/Unit(0x2) → c12`

#### A.3.3.5 *CSD of SAS Device path node*

Use the SAS Address, followed by the LUN number, Device Topology Value, and Relative Target Port

#### A.3.3.6 *CSD of SATA Device path node*

Use the HBA Port Number, followed by the Port Multiplier Number, and the LUN value.

#### A.3.3.7 *CSD of hard drive device path node*

Hard disk node - > partition number

#### A.3.3.7.1 *Examples*

```
Ata(Primary,Master) → a
Ata(Primary, Slave) → b
Ata(Secondary,Master) → c
Ata(Secondary, Slave) → d
HD(p2, sig*) → c
CD(Entry0) → a
```

SCSI device

#### A.3.3.8 *CSD of SCSI device path node*

Use LUN number and PUN number for scsi node

#### A.3.3.8.1 *Examples:*

```
scsi(1,0) → b0
scsi(2,0) → c0
scsi(2,0)/scsi(1,0) → c0b0
```

#### A.3.3.9 *Fibre Channel Device Path*

Use WWW number and LUN number for fibre channel device node:  
Acpi(0a03)/pci(0,0)/FC(0x1b833212, 0x34a65)/HD(Part4, sig\*\*4)  
hd45bmwccxe4654145e  
(bmwccxe)26 = (1b833212)16, (4654145)10 = (34a65)16

#### A.3.3.10 *1394 Device Path*

Use GUID as CSD for 1394 device path node

Treat the guid as a string, for CSD, translate character by character  
0001,db08,5001,0a5f → aaabnlaifaabakfp

#### A.3.3.11 *I2o Device Path*

Use Target ID as I2o Device Path

#### A.3.3.12 *MAC Address Device Path*

Use MAC address as CSD for MAC node

IPv4 Device Path & IPv6 Device Path

Use local Ip address, local port, Remote Ip address, remote port for IPv4 node

#### A.3.3.13 *InfiniBand Device Path*

Use PORT GUID, IOC GUID, Target Port ID, Device ID

### A.3.3.14 *UART Device Path*

Use Baud Rate, Data Bits, Parity, Stop Bits as CSD

### A.3.3.15 *Vendor-Defined Device Path*

Use guid as CSD  
{18ABEA39-F472-4278-BD55-E8C81C7030E1} →  
biklokdjpehcechiInffoimibmhadaob

### A.3.3.16 *Vendor-defined Media Device Path*

Use Vendor\_GUID as CSD

### A.3.3.17 *File Path Media Device Path*

Put file path in MTD

## A.4 *Function & Structure.*

```
EFI_STATUS  
ConsistMappingCreateHIDevicePathTable (  
    OUT EFI_DEVICE_PATH_PROTOCOL          ***HIDevicePathTable  
) ;
```

### Parameters

*HIDevicePathTable*  
A pointer to the Table of HI Device Path.

### Description

This function will go through all the device paths in the system, extract the HI device path from each device path and add the HI device path to the HI device path table then sort the HI device path table.

In this function, only use the device path's first instance, if the device path has more than one instance, the other instances are ignored.

Extract the HI device path.

To each device path, go through each node from the first device path node. According to Table 33, if the node is the HI Device path node, add it to the HI Device, path, until meet the first NuHI Device path node.

Sort the HI device path table.

According the compare rules above, sort the HI device path table.

## Pseudo Code

```
foreach handle in System
{
  Get DevicePath form handle;
  If DevicePath is null {
    Continue;
  }
  HIDevicePath = ExtractHI(DevicePath);
  Add the HIDevicePath to the HDPT
}
Sort the HDPT
```

## Status Codes Returned

SHELL_SUCCESS	Success to get the HI device path table.
SHELL_OUT_OF_RESOURCE	Can not create the HI device path table.

```
CHAR16*
ConsistMappingGenMappingName(
  IN EFI_DEVICE_PATH_PROTOCOL *DevicePath,
  IN EFI_DEVICE_PATH_PROTOCOL **HIDevicePathTable
) ;
```

## Parameters

*DevicePath*

A pointer to a device path that will be translated to the consist name.

*HIDevicePathTable*

A pointer to the Table of HI Device Path.

## Description

This function go through each node of the DevicePath, extract the info about the MTD, HI, and CSD, and then according to the extracted info, create a consistent mapping.

## Pseudo Code

```
Foreach handle in System
{
  Get DevicePath form handle;
  If DevicePath is null {
    Continue;
  }
  HIDevicePath = ExtractHI(DevicePath);
  Add the HIDevicePath to the HDPT
}
Sort the HDPT
```

## Status Codes Returned

NULL	Can't create the consistent mapping.
NULL	The consistent mapping.

Sort the HI device path table.

Consistent mapping device path compare:

```
Compare(Node1, Node2)
{
    If (DeviceType(Node1) != DeviceType(Node2)) {
        Return DeviceType(Node1) -
DeviceType(Node2);
    }
    If (DeviceSubType(Node1) != DeviceSubType(Node2)) {
        Return DeviceSubType(Node1) -
DeviceSubType(Node2);
    }
    If (DeviceSubType(Node) == PCI) {
pci(d1, f1) > pci (d2, f2) only if d1 > d2 or (d1 = d2 and f1>f2)
    }
    If (DeviceSubType(Node) == ACPI) {
pci(d1, f1) > pci (d2, f2) only if d1 > d2 or (d1 = d2 and f1>f2)
    }
    If (Length(Node) != Length(Node2)) {
        Return (Length(Node1) - Length(Node2));
    }
    Return memcmp(Node1, Node2, Length(Node1));
}
```





# Appendix B UEFI Help Manual

## Page Syntax

---

The manual page files are standard text files with title and section heading information embedded using commands which begin with a '.'. The following two macros are supported:

*.TH command-name 0 "short-description"*

Title header. When printing multi-page help, this will appear at the top of each page.

*.SH section-name*

Sub-header. Specifies one of several standard sub-headings.

**Table 30 Subheadings and descriptions**

Sub-Heading Name	Description
<b>NAME</b>	The name of the function or command, along with a one-line summary.
<b>SYNOPSIS</b>	Usage of the command
<b>DESCRIPTION</b>	General description
<b>OPTIONS</b>	Description of all options and parameters.
<b>RETURN VALUES</b>	Values returned.
<b>ENVIRONMENT VARIABLES</b>	Environment variables used
<b>FILES</b>	Files associated with the subject.
<b>EXAMPLES</b>	Examples and suggestions.
<b>ERRORS</b>	Errors reported by the command.
<b>STANDARDS</b>	Conformance to applicable standards.
<b>BUGS</b>	Errors and caveats.
<b>CATEGORY</b>	The comma-delimited list of categories to which this command belongs. Category names must follow normal file naming conventions. Category names which begin with '_' will not be used in the specification. See section TBD ("Command-Line Help") for more information on how this category is used when installing new commands.
<i>other</i>	Other sections added by the help author.



# Appendix C UEFI Shell Status Codes

Most UEFI Shell commands return SHELL\_STATUS code values. These UEFI Shell status codes are enumerated below.

**Table 31 SHELL\_STATUS return codes**

Mnemonic	Value	Description
SHELL_SUCCESS	0	The operation completed successfully
SHELL_LOAD_ERROR	1	The image failed to load.
SHELL_INVALID_PARAMETER	2	There was an error in the command-line options.
SHELL_UNSUPPORTED	3	The operation is not supported.
SHELL_BAD_BUFFER_SIZE	4	The buffer was not the proper size for the request.
SHELL_BUFFER_TOO_SMALL	5	The buffer is not large enough to hold the requested data. The required buffer size is returned in the appropriate parameter when this error occurs.
SHELL_NOT_READY	6	There is no data pending upon return.
SHELL_DEVICE_ERROR	7	The physical device reported an error while attempting the operation.
SHELL_WRITE_PROTECTED	8	The device cannot be written to.
SHELL_OUT_OF_RESOURCES	9	A resource has run out.
SHELL_VOLUME_CORRUPTED	10	An inconsistency was detected on the file system causing the operating to fail.
SHELL_VOLUME_FULL	11	There is no more space on the file system.
SHELL_NO_MEDIA	12	The device does not contain any medium to perform the operation.
SHELL_MEDIA_CHANGED	13	The medium in the device has changed since the last access.
SHELL_NOT_FOUND	14	The item was not found.
SHELL_ACCESS_DENIED	15	Access was denied.
SHELL_TIMEOUT	18	The timeout time expired.
SHELL_NOT_STARTED	19	The specified operation could not be started.
SHELL_ALREADY_STARTED	20	The specified operation had already started.
SHELL_ABORTED	21	The operation was aborted by the user
SHELL_INCOMPATIBLE_VERSION	25	The function encountered an internal version that was incompatible with a version requested by the caller.

<b>SHELL_SECURITY_VIOLATION</b>	26	The function was not performed due to a security violation.
<b>SHELL_NOT_EQUAL</b>	27	The function was performed and resulted in an unequal comparison..

# Appendix D UEFI Shell Command Standard Formatted Output

---

This section describes the general, table-based standard output format for UEFI shell commands. The format is designed so that tools can easily process output produced by shell commands.

UEFI shell commands using standard formatted output display the same information they would normally display, except using rows and columns of comma-delimited data. The first column always contains a C-style identifier which describes the type of data on the row. This identifier is known as the *table name*. Table names which begin with the '\_' character are implementation-specific.

The second and subsequent columns are quoted C-style strings containing the actual UEFI shell command data. For each UEFI shell command, the format and meaning of each column depends on the column number and the *table name*.

Shell commands which support the `-sfo` option will always produce the table name `ShellCommand`. The second column contains the name of the shell command without any extension. For example:

```
ShellCommand,"ls"
```

For example:

```
FileInfo,"fs0:/efi/boot/winloader.efi","45670","arsh"  
FileInfo,"fs0:/efi/boot/timsfile.txt","1250","a"  
FileInfo,"fs0:/efi/boot/readme.txt","795","a"
```

In the syntax below, an *identifier* is a C-style identifier, which starts with an alphabetic character or underscore. A quoted string starts with a double-quotation mark (") character, followed by zero or more characters and concluding with a double-quotation mark (") character. Quotation marks in the string must be escaped by using a ^ character (i.e. ^"). The ^ character may be inserted using ^^.

## Extended Syntax

```
sfo-format           := sfo-row  
  
                       sfo-row <EOL> <sfo-row>  
  
sfo-row              := sfo-table-name, sfo-columns  
  
sfo-table-name      := identifier  
  
sfo-columns         := sfo-column |
```

```
                                sfo-columns, | sfo-column
sfo-column := quoted-string |
                                <empty>
```

## Example

```
VolumeInfo, "TimsVolume", "400000000", "320000000", "160000000"
FileInfo, "fs0:/efi/boot/winloader.efi", "45670", "arsh"
FileInfo, "fs0:/efi/boot/timsfile.txt", "1250", "a"
FileInfo, "fs0:/efi/boot/readme.txt", "795", "a"
```