Using the EFI_SHELL_PROTOCOL To Read a File

Finnbarr P. Murphy

(fpm@fpmurphy.com)

In this post, I provide the source code for a working UEFI shell application which displays the contents of an ASCII file using functionality provided by the *EFI_SHELL_PROTOCOL* protocol.

The current version of the *EFI_SHELL_PROTOCOL* is 2.1 and here are the exposed protocol interfaces:

```
typedef struct _EFI_SHELL_PROTOCOL {
  EFI_SHELL_EXECUTE                         Execute;
  EFI_SHELL_GET_ENV                         GetEnv;
  EFI_SHELL_SET_ENV                         SetEnv;
  EFI_SHELL_GET_ALIAS                       GetAlias;
  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;
  // Added for Shell 2.1
  EFI_SHELL_REGISTER_GUID_NAME              RegisterGuidName;
  EFI_SHELL_GET_GUID_NAME                   GetGuidName;
  EFI_SHELL_GET_GUID_FROM_NAME              GetGuidFromName;
  EFI_SHELL_GET_ENV_EX                      GetEnvEx;
```



```
} EFI_SHELL_PROTOCOL;
```

See the UEFI Shell Specification (currently version 2.2, January 2016) for all the details on the protocol.

There is a page on TianoCore which attempts to explain how to create a UEFI shell application. It was written some time ago and is woefully inadequate for somebody who is trying to learn how to make use of the *EFI_SHELL_PROTOCOL*. Few of the example applications provided in UDK2015 are invoked in the described manner. The page needs to be completely rewritten and a complete working example included.

Moving on, how can an application access the functions exposed by the *EFI_SHELL_PROTOCOL*? It turns out that there is a single global instance of the *EFI_SHELL_PROTOCOL* per shell instance – not one per application. Note, however, if you have nested copies of the UEFI Shell you may have multiple copies of the *EFI_SHELL_PROTOCOL* protocol. Personally, I have never got nested copies of a UEFI Shell to reliably work for any period of time on any platform so far, including the Intel S1200V3RPS server board which is the current reference board for UEFI development.

One way of accessing the functions exposed by *EFI_SHELL_PROTOCOL* is via the global pointer *gEfiShellProtocol*:

```
EFI_SHELL_PROTOCOL            *gEfiShellProtocol;

gEfiShellProtocol->SetFilePosition(Handle, FilePosition);
gEfiShellProtocol->CloseFile(Handle);
```

If you have nested shells, you can locate the applicable *EFI_SHELL_PROTOCOL* by doing something like the following:

```
Status = gBS->LocateProtocol( &gEfiShellProtocolGuid,
                              NULL,
                              (VOID **)&);
```

Another way is via the *FILE_HANDLE_FUNCTION_MAP* structure:

```
typedef struct  {
  EFI_SHELL_GET_FILE_INFO                   GetFileInfo;
  EFI_SHELL_SET_FILE_INFO                   SetFileInfo;
  EFI_SHELL_READ_FILE                       ReadFile;
  EFI_SHELL_WRITE_FILE                      WriteFile;
  EFI_SHELL_CLOSE_FILE                      CloseFile;
  EFI_SHELL_DELETE_FILE                     DeleteFile;
  EFI_SHELL_GET_FILE_POSITION               GetFilePosition;
  EFI_SHELL_SET_FILE_POSITION               SetFilePosition;
  EFI_SHELL_FLUSH_FILE                      FlushFile;
  EFI_SHELL_GET_FILE_SIZE                   GetFileSize;
} FILE_HANDLE_FUNCTION_MAP;

FILE_HANDLE_FUNCTION_MAP      FileFunctionMap;

EFI_STATUS
EFIAPI
ShellLibConstructorWorker ( IN EFI_HANDLE        ImageHandle,
                            IN EFI_SYSTEM_TABLE  *SystemTable )
{
  if ((mEfiShellEnvironment2 != NULL && mEfiShellInterface         != NULL) ||
      (gEfiShellProtocol     != NULL && gEfiShellParametersProtocol != NULL)   ) {
```



```
    if (gEfiShellProtocol != NULL) {
      FileFunctionMap.GetFileInfo     = gEfiShellProtocol->GetFileInfo;
      FileFunctionMap.SetFileInfo     = gEfiShellProtocol->SetFileInfo;
      FileFunctionMap.ReadFile        = gEfiShellProtocol->ReadFile;
      FileFunctionMap.WriteFile       = gEfiShellProtocol->WriteFile;
      FileFunctionMap.CloseFile       = gEfiShellProtocol->CloseFile;
      FileFunctionMap.DeleteFile      = gEfiShellProtocol->DeleteFile;
      FileFunctionMap.GetFilePosition = gEfiShellProtocol->GetFilePosition;
      FileFunctionMap.SetFilePosition = gEfiShellProtocol->SetFilePosition;
      FileFunctionMap.FlushFile       = gEfiShellProtocol->FlushFile;
      FileFunctionMap.GetFileSize     = gEfiShellProtocol->GetFileSize;
    } else {
      FileFunctionMap.GetFileInfo     = (EFI_SHELL_GET_FILE_INFO)FileHandleGetInfo;
      FileFunctionMap.SetFileInfo     = (EFI_SHELL_SET_FILE_INFO)FileHandleSetInfo;
      FileFunctionMap.ReadFile        = (EFI_SHELL_READ_FILE)FileHandleRead;
      FileFunctionMap.WriteFile       = (EFI_SHELL_WRITE_FILE)FileHandleWrite;
      FileFunctionMap.CloseFile       = (EFI_SHELL_CLOSE_FILE)FileHandleClose;
      FileFunctionMap.DeleteFile      = (EFI_SHELL_DELETE_FILE)FileHandleDelete;
      FileFunctionMap.GetFilePosition = (EFI_SHELL_GET_FILE_POSITION)FileHandleGetPosition;
      FileFunctionMap.SetFilePosition = (EFI_SHELL_SET_FILE_POSITION)FileHandleSetPosition;
      FileFunctionMap.FlushFile       = (EFI_SHELL_FLUSH_FILE)FileHandleFlush;
      FileFunctionMap.GetFileSize     = (EFI_SHELL_GET_FILE_SIZE)FileHandleGetSize;
    }
    ...
  }
  ...
}

// example use of FILE_HANDLE_FUNCTION_MAP
EFI_STATUS
EFIAPI
ShellCloseFile ( IN SHELL_FILE_HANDLE *FileHandle )
{
    return (FileFunctionMap.CloseFile(*FileHandle));
}
```

The *Shell** set of functions are implemented in *.../ShellLib.c* and *ShellCommandLib.c*:

```
ShellCommandAddMapItemAndUpdatePath
ShellCommandConsistMappingGenMappingName
ShellCommandConsistMappingInitialize
ShellCommandConsistMappingUnInitialize
ShellCommandCreateInitialMappingsAndPaths
ShellCommandCreateNewMappingName
ShellCommandFindMapItem
ShellCommandGetCommandHelp
ShellCommandGetCommandList
ShellCommandGetCurrentScriptFile
ShellCommandGetEchoState
ShellCommandGetExit
ShellCommandGetExitCode
ShellCommandGetInitAliasList
ShellCommandGetManFileNameHandler
ShellCommandGetProfileList
ShellCommandGetScriptExit
ShellCommandIsCommandOnList
ShellCommandIsOnAliasList
ShellCommandLineCheckDuplicate
ShellCommandLineFreeVarList
ShellCommandLineGetCount
ShellCommandLineGetFlag
ShellCommandLineGetRawValue
ShellCommandLineGetValue
ShellCommandLineParseEx
ShellCommandRegisterAlias
```



```
ShellCommandRegisterCommandName
ShellCommandRegisterExit
ShellCommandRunCommandHandler
ShellCommandSetEchoState
ShellCommandSetNewScript
ShellCommandUpdateMapping

ShellCloseFile
ShellCloseFileMetaArg
ShellConvertStringToUint64
ShellCopySearchAndReplace
ShellCreateDirectory
ShellDeleteFile
ShellDeleteFileByName
ShellExecute
ShellFileExists
ShellFileHandleEof
ShellFileHandleGetPath
ShellFileHandleReadLine
ShellFileHandleRemove
ShellFileHandleReturnLine
ShellFindFilePath
ShellFindFilePathEx
ShellFindFirstFile
ShellFindNextFile
ShellFlushFile
ShellGetCurrentDir
ShellGetEnvironmentVariable
ShellGetExecutionBreakFlag
ShellGetFileInfo
ShellGetFilePosition
ShellGetFileSize
ShellHexStrToUintn
ShellInitialize
ShellIsDecimalDigitCharacter
ShellIsDirectory
ShellIsFile
ShellIsFileInPath
ShellIsHexaDecimalDigitCharacter
ShellIsHexOrDecimalNumber
ShellOpenFileByDevicePath
ShellOpenFileByName
ShellOpenFileMetaArg
ShellPrintEx
ShellPrintHelp
ShellPrintHiiEx
ShellPromptForResponse
ShellPromptForResponseHii
ShellReadFile
ShellSetEnvironmentVariable
ShellSetFileInfo
ShellSetFilePosition
ShellSetPageBreakMode
ShellStrToUintn
ShellWriteFile
```

I choose to use the second method in my example application as it involves less typing and appears to me to be the “cleaner” method.

Here is the source code for my example application:

```


//
//  Copyright (c) 2017  Finnbarr P. Murphy.   All rights reserved.
//
//  Print contents of ASCII file
```

```
//  Demonstrates use of EFI_SHELL_PROTOCOL functions
//
//  License: BSD License
//
#include <Uefi.h>
#include <Library/UefiLib.h>
#include <Library/ShellCEntryLib.h>
#include <Library/ShellLib.h>
#include <Library/ShellCommandLib.h>
#include <Library/BaseMemoryLib.h>
#include <Library/MemoryAllocationLib.h>
#include <Library/UefiBootServicesTableLib.h>
#include <Library/PrintLib.h>
#include <Protocol/EfiShell.h>
#include <Protocol/LoadedImage.h>
#define UTILITY_VERSION L"0.8"
#define LINE_MAX  1024
VOID
Usage(CHAR16 *Str)
{
    Print(L"Usage: %s [--version | --help ]\n", Str);
    Print(L"       %s [--number] [--nocomment] filename\n", Str);
}
INTN
EFIAPI
ShellAppMain(UINTN Argc, CHAR16 **Argv)
{
    EFI_STATUS Status = EFI_SUCCESS;
    SHELL_FILE_HANDLE FileHandle;
    CHAR16  *FileName;
    CHAR16  *FullFileName;
    CHAR16  *ReadLine;
    CHAR16  *Walker;
    BOOLEAN Ascii = TRUE;
    BOOLEAN NoComment = FALSE;
    BOOLEAN LineNumber = FALSE;
    UINTN   Size = LINE_MAX;
    UINTN   LineNo = 0;
    int     i;
    if (Argc < 2 || Argc > 4) {
        Usage(Argv[0]);
        return Status;
    }
    for (i = 1; i < Argc; i++) {
        Walker = (CHAR16 *)Argv[i];
        if (!StrCmp(Argv[i], L"--version") ||
            !StrCmp(Argv[i], L"-V")) {
            Print(L"Version: %s\n", UTILITY_VERSION);
            return Status;
        } else if (!StrCmp(Argv[i], L"--help") ||
            !StrCmp(Argv[i], L"-h") ||
            !StrCmp(Argv[i], L"-?")) {
            Usage(Argv[0]);
            return Status;
        } else if (!StrCmp(Argv[i], L"--nocomment")) {
            NoComment = TRUE;
        } else if (!StrCmp(Argv[i], L"--number")) {
            LineNumber = TRUE;
        } else if (*Walker != L'-') {
            break;
        } else {
            Print(L"ERROR: Unknown option.\n");
            Usage(Argv[0]);
            return Status;
        }
    }
    FileName = AllocateCopyPool(StrSize(Argv[i]), Argv[i]);
    if (FileName == NULL) {
```



```
        Print(L"ERROR: Could not allocate memory\n");
        return (EFI_OUT_OF_RESOURCES);
    }
    FullFileName = ShellFindFilePath(FileName);
    if (FullFileName == NULL) {
        Print(L"ERROR: Could not find %s\n", FileName);
        Status = EFI_NOT_FOUND;
        goto Error;
    }
    // open the file
    Status = ShellOpenFileByName(FullFileName, &amp;FileHandle, EFI_FILE_MODE_READ, 0);
    if (EFI_ERROR(Status)) {
        Print(L"ERROR: Could not open file\n");
        goto Error;
    }
    // allocate a buffer to read lines into
    ReadLine = AllocateZeroPool(Size);
    if (ReadLine == NULL) {
        Print(L"ERROR: Could not allocate memory\n");
        Status = EFI_OUT_OF_RESOURCES;
        goto Error;
    }
    // demo setting file position. Not actually needed here
    ShellSetFilePosition(FileHandle, 0);
    // read file line by line
    for (;!ShellFileHandleEof(FileHandle); Size = 1024) {
         Status = ShellFileHandleReadLine(FileHandle, ReadLine, &amp;Size, TRUE, &amp;Asci
i);
         if (Status == EFI_BUFFER_TOO_SMALL) {
             Status = EFI_SUCCESS;
         }
         if (EFI_ERROR(Status)) {
             break;
         }
         LineNo++;
         // Skip comment lines
         if (ReadLine[0] == L'#' &amp;&amp; NoComment) {
             continue;
         }
         if (LineNumber) {
            Print(L"%0.4d  ", LineNo);
         }
         Print(L"%s\n", ReadLine);
    }
Error:
    if (FileName != NULL) {
       FreePool(FileName);
    }
    if (FullFileName != NULL) {
       FreePool(FullFileName);
    }
    if (ReadLine != NULL) {
        FreePool(ReadLine);
    }
    if (FileHandle != NULL) {
        ShellCloseFile(&amp;FileHandle);
    }
    return Status;
}
```

As usual, it is assumed that you are familiar with the various UEFI specifications and UDK2015 if you are reading this blog post so no detailed explanation of the code or the build process is provided.

There is a “flaw” in the above code which renders this application unusable in a UEFI shell script.

My code returns *Status* as an exit code; *Status* is of type *EFI_STATUS*. If the application is to be used within a script, it needs to return one of the *SHELL_STATUS* values defined in *ShellBase.h*.

```
typedef enum {
   SHELL_SUCCESS               = 0,
   SHELL_LOAD_ERROR            = 1,
   SHELL_INVALID_PARAMETER     = 2,
   SHELL_UNSUPPORTED           = 3,
   SHELL_BAD_BUFFER_SIZE       = 4,
   SHELL_BUFFER_TOO_SMALL      = 5,
   SHELL_NOT_READY             = 6,
   SHELL_DEVICE_ERROR          = 7,
   SHELL_WRITE_PROTECTED       = 8,
   SHELL_OUT_OF_RESOURCES      = 9,
   SHELL_VOLUME_CORRUPTED      = 10,
   SHELL_VOLUME_FULL           = 11,
   SHELL_NO_MEDIA              = 12,
   SHELL_MEDIA_CHANGED         = 13,
   SHELL_NOT_FOUND             = 14,
   SHELL_ACCESS_DENIED         = 15,
   SHELL_TIMEOUT               = 18,
   SHELL_NOT_STARTED           = 19,
   SHELL_ALREADY_STARTED       = 20,
   SHELL_ABORTED               = 21,
   SHELL_INCOMPATIBLE_VERSION  = 25,
   SHELL_SECURITY_VIOLATION    = 26,
   SHELL_NOT_EQUAL             = 27
} SHELL_STATUS;
```

In order to make my example application usable within UEFI shell scripts, I would have to map the various possible *EFI_STATUS* values to *SHELL_STATUS* values.

The following code snippet comes from UDK2015 and illustrates mapping between *EFI_STATUS* and *SHELL_STATUS*:

```
{
  EFI_STATUS           Status;
  SHELL_STATUS         ShellStatus;
  UINT32               NameSize;
  UINT32               DataSize;
  UINTN                BufferSize;
  UINTN                RemainingSize;
  UINT64               Position;
  UINT64               FileSize;
  LIST_ENTRY           List;
  DMP_STORE_VARIABLE   *Variable;
  LIST_ENTRY           *Link;
  CHAR16               *Attributes;
  UINT8                *Buffer;

  Status = ShellGetFileSize (FileHandle, &FileSize);
  if (EFI_ERROR (Status)) {
    return SHELL_DEVICE_ERROR;
  }

  ShellStatus = SHELL_SUCCESS;

  InitializeListHead (&List);

  Position = 0;
  while (Position < FileSize) {
//
// NameSize
```



```
//
BufferSize = sizeof (NameSize);
Status = ShellReadFile (FileHandle, &BufferSize, &NameSize);
if (EFI_ERROR (Status) || (BufferSize != sizeof (NameSize))) {
ShellStatus = SHELL_VOLUME_CORRUPTED;
break;
}

//
// DataSize
//
BufferSize = sizeof (DataSize);
Status = ShellReadFile (FileHandle, &BufferSize, &DataSize);
if (EFI_ERROR (Status) || (BufferSize != sizeof (DataSize))) {
ShellStatus = SHELL_VOLUME_CORRUPTED;
break;
}
```

Here is a copy the *.inf* build file for this example application:

```
[Defines]
  INF_VERSION                    = 0x00010006
  BASE_NAME                      = ReadDemo1
  FILE_GUID                      = 4ea87c51-7491-4dfd-0055-747010f3ce51
  MODULE_TYPE                    = UEFI_APPLICATION
  VERSION_STRING                 = 0.1
  ENTRY_POINT                    = ShellCEntryLib
  VALID_ARCHITECTURES            = X64

[Sources]
  ReadDemo1.c

[Packages]
  MdePkg/MdePkg.dec
  ShellPkg/ShellPkg.dec

[LibraryClasses]
  ShellCEntryLib
  ShellLib
  ShellCommandLib
  BaseLib
  BaseMemoryLib
  UefiLib

[Protocols]

[BuildOptions]

[Pcd]
```

Note that you may have to add various *ShellPkg* libraries to your *.dsc* file to successfully build the example application. Here is the contents of my current *../UDK2015/MyApps/MyApps.dsc*:

```
[Defines]
  PLATFORM_NAME                  = MyApps
  PLATFORM_GUID                  = 0458dade-8b6e-4e45-b773-1b27cbda3e06
  PLATFORM_VERSION               = 0.01
  DSC_SPECIFICATION              = 0x00010006
  OUTPUT_DIRECTORY               = Build/MyApps
  SUPPORTED_ARCHITECTURES        = X64
  BUILD_TARGETS                  = DEBUG|RELEASE|NOOPT
  SKUID_IDENTIFIER               = DEFAULT
```



```
#
#  Debug output control
#
  DEFINE DEBUG_ENABLE_OUTPUT      = FALSE       # Set to TRUE to enable debug output
  DEFINE DEBUG_PRINT_ERROR_LEVEL  = 0x80000040  # Flags to control amount of debug output
  DEFINE DEBUG_PROPERTY_MASK      = 0

[PcdsFeatureFlag]

[PcdsFixedAtBuild]
  gEfiMdePkgTokenSpaceGuid.PcdDebugPropertyMask|$(DEBUG_PROPERTY_MASK)
  gEfiMdePkgTokenSpaceGuid.PcdDebugPrintErrorLevel|$(DEBUG_PRINT_ERROR_LEVEL)

[PcdsFixedAtBuild.IPF]

[LibraryClasses]
  #
  # Entry Point Libraries
  #
  UefiApplicationEntryPoint|MdePkg/Library/UefiApplicationEntryPoint/UefiApplicationEntryP
oint.inf
  ShellCEntryLib|ShellPkg/Library/UefiShellCEntryLib/UefiShellCEntryLib.inf
  UefiDriverEntryPoint|MdePkg/Library/UefiDriverEntryPoint/UefiDriverEntryPoint.inf
  #
  # Common Libraries
  #
  BaseLib|MdePkg/Library/BaseLib/BaseLib.inf
  BaseMemoryLib|MdePkg/Library/BaseMemoryLib/BaseMemoryLib.inf
  UefiLib|MdePkg/Library/UefiLib/UefiLib.inf
  PrintLib|MdePkg/Library/BasePrintLib/BasePrintLib.inf
  PcdLib|MdePkg/Library/BasePcdLibNull/BasePcdLibNull.inf
  MemoryAllocationLib|MdePkg/Library/UefiMemoryAllocationLib/UefiMemoryAllocationLib.inf
  UefiBootServicesTableLib|MdePkg/Library/UefiBootServicesTableLib/UefiBootServicesTableLi
b.inf
  UefiRuntimeServicesTableLib|MdePkg/Library/UefiRuntimeServicesTableLib/UefiRuntimeServic
esTableLib.inf
  !if $(DEBUG_ENABLE_OUTPUT)
    DebugLib|MdePkg/Library/UefiDebugLibConOut/UefiDebugLibConOut.inf
    DebugPrintErrorLevelLib|MdePkg/Library/BaseDebugPrintErrorLevelLib/BaseDebugPrintError
LevelLib.inf
  !else   ## DEBUG_ENABLE_OUTPUT
    DebugLib|MdePkg/Library/BaseDebugLibNull/BaseDebugLibNull.inf
  !endif  ## DEBUG_ENABLE_OUTPUT

  DevicePathLib|MdePkg/Library/UefiDevicePathLib/UefiDevicePathLib.inf
  PeCoffGetEntryPointLib|MdePkg/Library/BasePeCoffGetEntryPointLib/BasePeCoffGetEntryPoint
Lib.inf
  IoLib|MdePkg/Library/BaseIoLibIntrinsic/BaseIoLibIntrinsic.inf
  PciLib|MdePkg/Library/BasePciLibCf8/BasePciLibCf8.inf
  PciCf8Lib|MdePkg/Library/BasePciCf8Lib/BasePciCf8Lib.inf
  SynchronizationLib|MdePkg/Library/BaseSynchronizationLib/BaseSynchronizationLib.inf
  UefiRuntimeLib|MdePkg/Library/UefiRuntimeLib/UefiRuntimeLib.inf
  HiiLib|MdeModulePkg/Library/UefiHiiLib/UefiHiiLib.inf
  UefiHiiServicesLib|MdeModulePkg/Library/UefiHiiServicesLib/UefiHiiServicesLib.inf
  PerformanceLib|MdeModulePkg/Library/DxePerformanceLib/DxePerformanceLib.inf
  HobLib|MdePkg/Library/DxeHobLib/DxeHobLib.inf
  FileHandleLib|MdePkg/Library/UefiFileHandleLib/UefiFileHandleLib.inf
  SortLib|MdeModulePkg/Library/UefiSortLib/UefiSortLib.inf

  # FPM - ShellLib
  ShellLib|ShellPkg/Library/UefiShellLib/UefiShellLib.inf
  ShellCommandLib|ShellPkg/Library/UefiShellCommandLib/UefiShellCommandLib.inf
  HandleParsingLib|ShellPkg/Library/UefiHandleParsingLib/UefiHandleParsingLib.inf

  CacheMaintenanceLib|MdePkg/Library/BaseCacheMaintenanceLib/BaseCacheMaintenanceLib.inf

[Components]
```



```
#### Applications.
  # MyApps/HelloWorld/HelloWorld.inf
  # MyApps/Hello/Hello.inf
  # MyApps/WriteDemo/WriteDemo.inf
  # MyApps/ScreenCapture/ScreenCapture.inf
  # MyApps/ShowESRT/ShowESRT.inf
  # MyApps/ShowMSDM/ShowMSDM.inf
  # MyApps/ShowECT/ShowECT.inf
  # MyApps/ShowBGRT/ShowBGRT.inf
  # MyApps/ScreenModes/ScreenModes.inf
  # MyApps/DisplayBMP/DisplayBMP.inf
  # MyApps/ShowEDID/ShowEDID.inf
  # MyApps/ShowTrEE/ShowTrEE.inf
  # MyApps/ShowTrEELog/ShowTrEELog.inf
  # MyApps/ShowTCM20/ShowTCM20.inf
  # MyApps/ShowEK/ShowEK.inf
  # MyApps/tpm_getpermflags/tpm_getpermflags.inf
  # MyApps/tpm_getrandom/tpm_getrandom.inf
  # MyApps/ShowTPM2/ShowTPM2.inf
  # MyApps/ShowOSIndications/ShowOSIndications.inf
  # MyApps/ShowRNG/ShowRNG.inf
  # MyApps/CheckSMM/CheckSMM.inf
  # MyApps/ThinkPwn/ThinkPwn.inf
  # MyApps/ShowPCR12/ShowPCR12.inf
  # MyApps/CheckFTP4/CheckFTP4.inf
  # MyApps/ShowPCR20/ShowPCR20.inf
  # MyApps/ShowPCI/ShowPCI.inf
  MyApps/ReadDemo1/ReadDemo1.inf
```

Well, this should be sufficient information to enable you to write your own UEFI shell applications that read files using interfaces exposed by the *EFI_SHELL_PROTOCOL*.

Enjoy!

