[UEFI Practice] Redfish BIOS Implementation 1

原创 jiangwei0512 Posted on 2022-12-11 19:05:07 Read 5.5k Collection 16 Likes 3

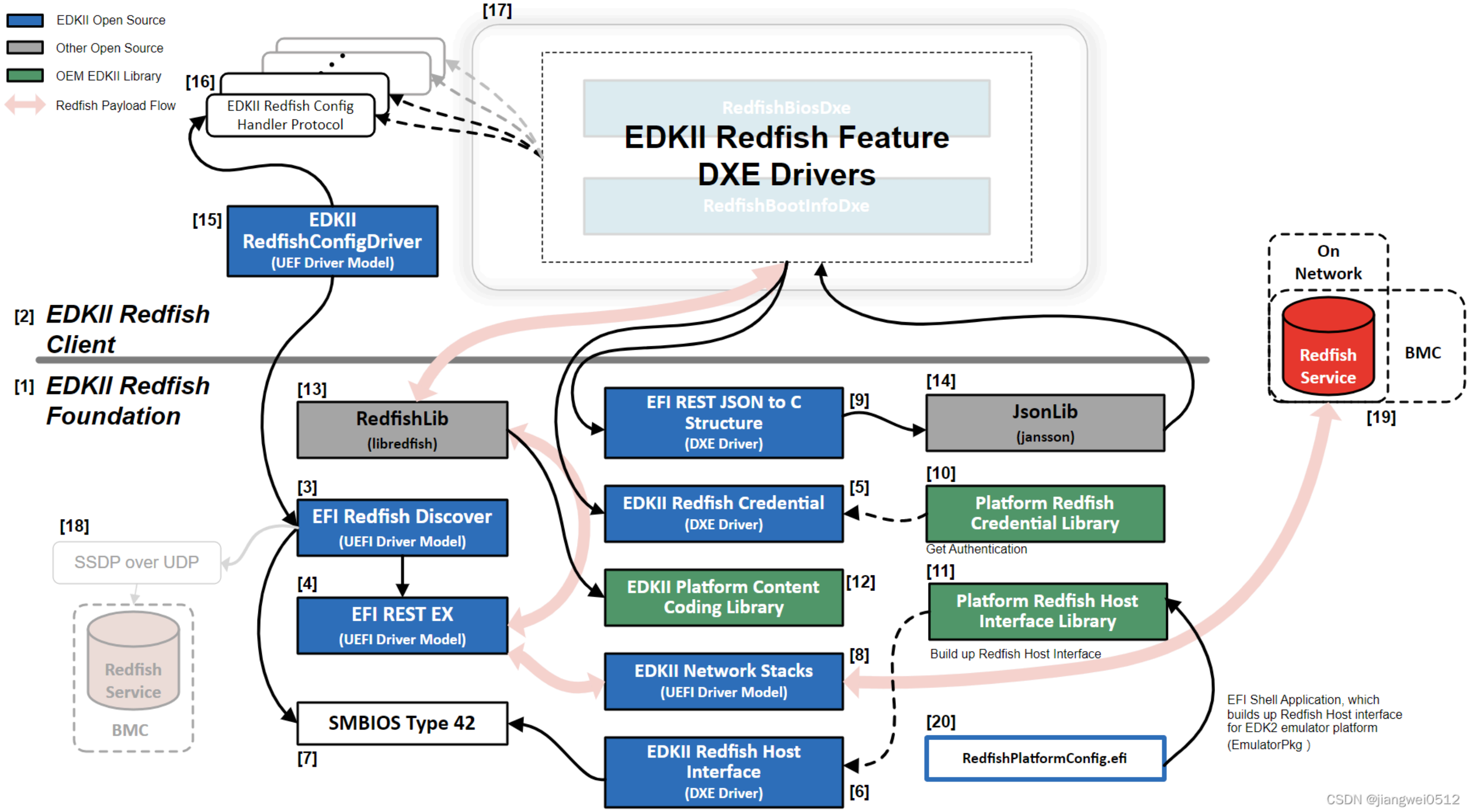
Category Column: UEFI Development Basics Article Tags: uefi redfish

AI摘要 This article describes how to use the EDK2 framework to implement Redfish BIOS configuration, covering the functions and dependencies of key modules, and providing technical support for out-of-band BIOS management.

The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

Redfish BIOS implementation

EDK2 provides the Redfish framework to implement out-of-band BIOS configuration. Its basic framework is as follows:


🟦 EDKII Open Source
▨ Other Open Source
🟩 OEM EDKII Library
⇔ Redfish Payload Flow
[17]
RedfishBiosDxe
EDKII Redfish Feature DXE Drivers
RedfishBootInfoDxe
[16]
EDKII Redfish Config Handler Protocol
[15]
EDKII RedfishConfigDriver (UEF Driver Model)
[2] EDKII Redfish Client
[1] EDKII Redfish Foundation
[13]
RedfishLib (libredfish)
[3]
EFI Redfish Discover (UEFI Driver Model)
[4]
EFI REST EX (UEFI Driver Model)
SMBIOS Type 42
[7]
[18]
SSDP over UDP
Redfish Service
BMC
EFI REST JSON to C Structure (DXE Driver)
[9]
[14]
JsonLib (jansson)
EDKII Redfish Credential (DXE Driver)
[5]
[10]
Platform Redfish Credential Library
Get Authentication
EDKII Platform Content Coding Library
[12]
[11]
Platform Redfish Host Interface Library
Build up Redfish Host Interface
EDKII Network Stacks (UEFI Driver Model)
[8]
EDKII Redfish Host Interface (DXE Driver)
[6]
[20]
RedfishPlatformConfig.efi
EFI Shell Application, which builds up Redfish Host interface for EDK2 emulator platform (EmulatorPkg)
On Network
Redfish Service
BMC
[19]
CSDN @jiangwei0512

The Driver provided in RedfishPkg enables BIOS to communicate with BMC or other software. It is mainly divided into two parts, Client and Foundation. Client corresponds to UEFI application and is used to interact with Service, while Service is the BMC or other program that implements Redfish.

How to include Redfish content in BIOS

The relevant modules are all in RedfishPkg, so you only need to add the modules to your own project. The main modules are as follows:

bash AI generated projects 登录复制

```bash
!if $(REDFISH_ENABLE) == TRUE
  EmulatorPkg/Application/RedfishPlatformConfig/RedfishPlatformConfig.inf
  RedfishPkg/RestJsonStructureDxe/RestJsonStructureDxe.inf
  RedfishPkg/RedfishHostInterfaceDxe/RedfishHostInterfaceDxe.inf
  RedfishPkg/RedfishRestExDxe/RedfishRestExDxe.inf
  RedfishPkg/RedfishCredentialDxe/RedfishCredentialDxe.inf
  RedfishPkg/RedfishDiscoverDxe/RedfishDiscoverDxe.inf
  RedfishPkg/RedfishConfigHandler/RedfishConfigHandlerDriver.inf
!endif
```

Of course, there are some additional Lib and PCD, which will be explained later if they are used. In addition, Redfish also depends on other basic modules, which also need to be added. They are mainly the contents in SecurePkg and NetworkPkg. The former is security support and the latter is the network basic module.

Module Introduction

The modules included above are described below.

RedfishHostInterfaceDxe.inf

This module is used to create SMBIOS Type42, which is described in the Redfish Host Interface Specification:

> The SMBIOS Type 42 structure is used to describe a Management Controller Host Interface. It consists of standard SMBIOS entry information, followed by interface descriptors (which detail the physical interface to the Redfish Service), and protocol descriptors (which describe the supported payload encoding between the Host and Redfish Service).

Specific data description (Table 1):

Offset	Name	Length	Description
00h	Type	1 byte	SMBIOS type value, here is 42
01h	Length	1 byte	Minimum 9 bytes
02h	Handle	2 bytes	Handle value, determined according to the actual code
04h	Interface Type	1 byte	The enumeration for the network interface is MCHostInterfaceTypeNetworkHostInterface, and the value is 40h
05h	Interface Specific Data Length	1 byte	See subsequent instructions
06h	Interface Specific Data	N bytes	See subsequent instructions
06h+N	Protocol Count	1 byte	Usually there is only one, so the value is 1
07h+N	Protocol Records	M bytes	See subsequent instructions

Implementation in code:

c AI generated projects 登录复制 run

```c
///
/// Management Controller Host Interface (Type 42).
///
/// The information in this structure defines the attributes of a Management
/// Controller Host Interface that is not discoverable by "Plug and Play" mechanisms.
///
/// Type 42 should be used for management controller host interfaces that use protocols
/// other than IPMI or that use multiple protocols on a single host interface type.
///
/// This structure should also be provided if IPMI is shared with other protocols
/// over the same interface hardware. If IPMI is not shared with other protocols,
/// either the Type 38 or Type 42 structures can be used. Providing Type 38 is
/// recommended for backward compatibility. The structures are not required to
/// be mutually exclusive. Type 38 and Type 42 structures may be implemented
/// simultaneously to provide backward compatibility with IPMI applications or drivers
/// that do not yet recognize the Type 42 structure.
///
typedef struct {
  SMBIOS_STRUCTURE                  Hdr;
  UINT8                             InterfaceType;                   ///< The enumeration value from MC_HOST_INTERFACE_TYPE
  UINT8                             InterfaceTypeSpecificDataLength;
  UINT8                             InterfaceTypeSpecificData[4];   ///< This field has a minimum of four bytes
} SMBIOS_TABLE_TYPE42;
```

收起 ^

When `InterfaceType` the value is 40h, the corresponding `InterfaceTypeSpecificData` data is as follows:

Offset	Name	Length	Description
00h	Device Type	1 byte	The underlying hardware of the network interface also has different types. The following are the specific types: 02h: USB network interface 03h: PCI/PCIe network interface 04h: USB network interface v2 05h: PCI/PCIe network interface v2 80h-FFh: OEM Others: Reserved
01h	Device Descriptor Data	N bytes	Because the types are different, the corresponding data is also different

The corresponding code:

c AI generated projects 登录复制 run

```c
///
///  Interface Specific Data starts at offset 06h of the SMBIOS Type 42 struct.
///  This table defines the Interface Specific data for Interface Type 40h. There
///  are 3 types of Device Descriptor3 defined , however only 1 may be used in
///  specific Tape 42 table.
///
typedef struct {
  UINT8             DeviceType;        ///< The Device Type of the interface.
  DEVICE_DESCRITOR DeviceDescriptor;  ///< The Device descriptor.
} REDFISH_INTERFACE_DATA;
```

收起 ^

The `DeviceType` values are defined in the code:

c AI generated projects 登录复制 run

```c
#define REDFISH_HOST_INTERFACE_DEVICE_TYPE_USB         0x02 // We don't support this type of interface.
                                                            // Use REDFISH_HOST_INTERFACE_DEVICE_TYPE_USB_V2 instead.
#define REDFISH_HOST_INTERFACE_DEVICE_TYPE_PCI_PCIE    0x03 // We don't support this type of interface.
                                                            // Use REDFISH_HOST_INTERFACE_DEVICE_TYPE_PCI_PCIE_V2 instead.
#define REDFISH_HOST_INTERFACE_DEVICE_TYPE_USB_V2      0x04
#define REDFISH_HOST_INTERFACE_DEVICE_TYPE_PCI_PCIE_V2 0x05
```

Corresponding `DeviceDescriptor` :

c AI generated projects 登录复制 run

```c
///
/// Define union for the Host Interface Device Descriptor
///
typedef union {
    USB_INTERFACE_DEVICE_DESCRIPTOR_V2          UsbDeviceV2;     ///< Device type USB V2 device discriptor.
    PCI_OR_PCIE_INTERFACE_DEVICE_DESCRIPTOR_V2  PciPcieDeviceV2; ///< Device type PCI/PCIe V2 device discriptor.
    OEM_DEVICE_DESCRIPTOR                       OemDevice;       ///< OEM type device discriptor.
} DEVICE_DESCRITOR; /// Device descriptor data formated based on Device Type.
```

Here we take the PCI network card as an example to explain its content:

c AI generated projects 登录复制 run

```c
//
// Structure definitions of Host Interface device type 05h (PCI/PCIE V2)
//
typedef struct {
  UINT8                   Length;               ///< Length of the structure, including Device Type and Length fields.
  UINT16                  VendorId;             ///< The Vendor ID of the PCI/PCIe device.
  UINT16                  DeviceId;             ///< The Device ID of the PCI/PCIe device.
  UINT16                  SubsystemVendorId;    ///< The Subsystem Vendor ID of the PCI/PCIe device.
  UINT16                  SubsystemId;          ///< The Subsystem ID of the PCI/PCIe device.
  UINT8                   MacAddress [6];       ///< The MAC address of the PCI/PCIe network device.
  UINT16                  SegmemtGroupNumber;   ///< The Segment Group Number of the PCI/PCIe.
  UINT8                   BusNumber;            ///< The Bus Number of the PCI/PCIe device.
  UINT8                   DeviceFunctionNumber; ///< The Device/Function Number of the PCI/PCIe.
} PCI_OR_PCIE_INTERFACE_DEVICE_DESCRIPTOR_V2;
```

收起 ^

You can see that it contains basic PCI information such as PCI Bus/Dev/Fun/VendorId/DeviceId/SubVendorId/SubDeviceId, as well as MAC address.

The number of protocols is currently 1, and its type is generally "Redfish over IP Protocol". Its specific contents are as follows:

Offset	Name	Length	Description
00h	Protocol Type	1 byte	The value of Redfish over IP Protocol is 04h
01h	Protocol Type Specific Data Length	1 byte	See subsequent instructions
02h	Protocol Specific Record Data	P Bytes	See subsequent instructions

The specific contents are as follows:

c AI generated projects 登录复制 run

```c
//
//  the protocol-specific data for the "Redfish Over IP" protocol
//
typedef struct {
  EFI_GUID            ServiceUuid;  //same as Redfish Service UUID in Redfish Service Root resource

  //
  //  Unknown=00h,
  //  Static=01h,
  //  DHCP=02h,
  //  AutoConfigure=03h,
  //  HostSelected=04h,
  //  other values reserved
  //
  UINT8               HostIpAssignmentType;

  //
  //  Unknown=00h,
  //  Ipv4=01h,
  //  Ipv6=02h,
  //  other values reserved
  //
  UINT8               HostIpAddressFormat;

  //
  //  Used for Static and AutoConfigure.
  //  For IPV4, use the first 4 Bytes and zero fill the remaining bytes.
  //
  UINT8               HostIpAddress[16];

  //
  //  Used for Static and AutoConfigure.
  //  For IPV4, use the first 4 Bytes and zero fill the remaining bytes.
  //
  UINT8               HostIpMask[16];

  //
  //  Unknown=00h,
  //  Static=01h,
  //  DHCP=02h,
  //  AutoConfigure=03h,
  //  HostSelected=04h,
  //  other values reserved
  //
  UINT8               RedfishServiceIpDiscoveryType;

  //
  //  Unknown=00h,
  //  Ipv4=01h,
  //  Ipv6=02h,
  //  other values reserved
  //
  UINT8               RedfishServiceIpAddressFormat;

  //
  //  Used for Static and AutoConfigure.
  //  For IPV4, use the first 4 Bytes and zero fill the remaining bytes.
  //
  UINT8               RedfishServiceIpAddress[16];

  //
  //  Used for Static and AutoConfigure.
  //  For IPV4, use the first 4 Bytes and zero fill the remaining bytes.
  //
  UINT8               RedfishServiceIpMask[16];
  UINT16              RedfishServiceIpPort;  // Used for Static and AutoConfigure.
  UINT32              RedfishServiceVlanId;  // Used for Static and AutoConfigure.
  UINT8               RedfishServiceHostnameLength;   // length of the following hostname string
  UINT8               RedfishServiceHostname[1];  // hostname of Redfish Service
} REDFISH_OVER_IP_PROTOCOL_DATA;
```

收起 ^

The above is the content of Type42 SMBIOS information. This module is used to build this SMBIOS. The data is determined according to the actual situation, such as whether to use PCIE or USB network card, what is the MAC address, what is the IP type, etc., and describes the detailed information of the interface used by Redfish.

Some of this information is determined directly in the code (both common and platform-specific), while others are written via variables that are related to the module corresponding to RedfishPlatformConfig.inf.

RedfishPlatformConfig.inf

This module has been mentioned before. It is a UEFI application that directly obtains data through command parameters and stores it in variables. The following is an example:

bash AI generated projects 登录复制

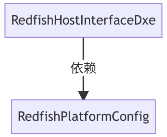
```bash
RedfishPlatformConfig.efi -s 192.168.10.101 255.255.255.0 192.168.10.123 255.255.255.0
```

It sets a range of IP addresses for Redfish to use.

Currently supported variables include:

- HostIpAssignmentType
- HostIpAddress
- HostIpMask
- RedfishServiceIpAddress
- RedfishServiceIpMask
- RedfishServiceIpPort

The main thing is the IP type and address of the Client and Service, which are included in `REDFISH_OVER_IP_PROTOCOL_DATA` this structure and used by RedfishDiscoverDxe.inf.



RedfishRestExDxe.inf

This module implements a HTTP-based transmission protocol for Redfish data communication. It installs the following two interfaces for subsequent data transmission:

c AI generated projects 登录复制 run

```c
EFI_SERVICE_BINDING_PROTOCOL mRedfishRestExServiceBinding = {
  RedfishRestExServiceBindingCreateChild,
  RedfishRestExServiceBindingDestroyChild
};

EFI_REST_EX_PROTOCOL  mRedfishRestExProtocol = {
  RedfishRestExSendReceive,
  RedfishRestExGetServiceTime,
  RedfishRestExGetService,
  RedfishRestExGetModeData,
  RedfishRestExConfigure,
  RedfishRestExAyncSendReceive,
  RedfishRestExEventService
};
```

收起 ^

RedfishDiscoverDxe.inf

The purpose of this module is to discover the Redfish Service end, such as BMC or the program that implements Redfish, and it depends on the various variables that need to be set mentioned in the previous module.

The RedfishDiscoverDxe module is a UEFI Driver, so it has interfaces such as Supported/Start:

c AI generated projects 登录复制 run

```c
EFI_DRIVER_BINDING_PROTOCOL gRedfishDiscoverDriverBinding = {
  RedfishDiscoverDriverBindingSupported,
  RedfishDiscoverDriverBindingStart,
  RedfishDiscoverDriverBindingStop,
  REDFISH_DISCOVER_VERSION,
  NULL,
  NULL
};
```

Here is a brief explanation:

`RedfishDiscoverDriverBindingSupported()` The function mainly determines whether the dependent module exists. The dependent parts are as follows:

c AI generated projects 登录复制 run

```c
static REDFISH_DISCOVER_REQUIRED_PROTOCOL gRequiredProtocol[] = {
  {
    ProtocolTypeTcp4,
    L"TCP4 Service Binding Protocol",
    &gEfiTcp4ProtocolGuid,
    &gEfiTcp4ServiceBindingProtocolGuid,    // RequiredServiceBindingProtocolGuid
    &mRedfishDiscoverTcp4InstanceGuid,      // DiscoveredProtocolGuid
    Tcp4GetSubnetInfo
  },
  {
    ProtocolTypeTcp6,
    L"TCP6 Service Binding Protocol",
    &gEfiTcp6ProtocolGuid,
    &gEfiTcp6ServiceBindingProtocolGuid,    // RequiredServiceBindingProtocolGuid
    &mRedfishDiscoverTcp6InstanceGuid,      // DiscoveredProtocolGuid
    Tcp6GetSubnetInfo
  },
  {
    ProtocolTypeRestEx,
    L"REST EX Service Binding Protocol",
    &gEfiRestExProtocolGuid,
    &gEfiRestExServiceBindingProtocolGuid,  // RequiredServiceBindingProtocolGuid
    &mRedfishDiscoverRestExInstanceGuid,    // DiscoveredProtocolGuid
    NULL
  }
};
```

收起 ^

Its judgment is based on:

1. Determine whether RequiredServiceBindingProtocolGuid exists;
2. If it exists, then determine whether the corresponding DiscoveredProtocolGuid exists;
3. If it already exists, it means that the UEFI Driver has been executed, so there is no need to execute it again;
4. If it does not exist, the UEFI Driver needs to be executed again.

In fact, only one of the three transmission methods here needs to be supported. The first two are different versions of TCP methods, and the last one is a transmission implementation based on HTTP, which is implemented in the module RedfishRestExDxe.inf.

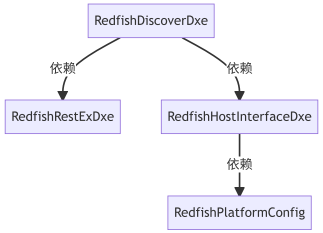
`RedfishDiscoverDriverBindingStart()` Build the network interface and finally install the following Protocol for subsequent use:

c AI generated projects 登录复制 run

```c
EFI_REDFISH_DISCOVER_PROTOCOL mRedfishDiscover = {
  RedfishServiceGetNetworkInterface,
  RedfishServiceAcquireService,
  RedfishServiceAbortAcquire,
  RedfishServiceReleaseService
};
```

These interfaces will be used by RedfishConfigHandlerDriver.inf.

Dependencies so far:



RedfishConfigHandlerDriver.inf

This module is the processing center of UEFI Redfish. It is also a UEFI Driver, so it contains the following interfaces:

c AI generated projects 登录复制 run

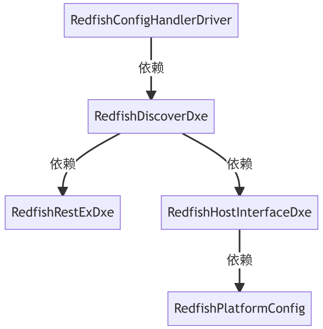
```c
///
/// Driver Binding Protocol instance
///
EFI_DRIVER_BINDING_PROTOCOL gRedfishConfigDriverBinding = {
  RedfishConfigDriverBindingSupported,
  RedfishConfigDriverBindingStart,
  RedfishConfigDriverBindingStop,
  REDFISH_CONFIG_VERSION,
  NULL,
  NULL
};
```

收起 ^

`RedfishConfigDriverBindingSupported()` Detect dependencies. In addition to the RedfishDiscoverDxe module mentioned above, the code mainly depends on the interface from the RedfishRestExDxe module, and the corresponding GUIDs are `gEfiRestExProtocolGuid` and `gEfiRestExServiceBindingProtocolGuid` .

`RedfishConfigDriverBindingStart()` There is a callback function inside, which will be called during installation, and the function corresponding `gEdkIIRedfishConfigHandlerProtocolGuid` to the GUID is executed in the callback function . The specific implementation of this protocol is not yet in the current EDK2 code. For details, see [EDK2 Redfish feature driver](#EDK2 Redfish feature driver). `EDKII_REDFISH_CONFIG_HANDLER_PROTOCOL Init()`

Dependencies so far:



RestJsonStructureDxe.inf

This module is a helper module used to process JSON data. It installs the following protocols:

c AI generated projects 登录复制 run

```c
EFI_REST_JSON_STRUCTURE_PROTOCOL mRestJsonStructureProtocol = {
  RestJsonStructureRegister,
  RestJsonStructureToStruct,
  RestJsonStructureToJson,
  RestJsonStructureDestroyStruct
};
```

RedfishCredentialDxe.inf

This module involves authentication-related content and provides the following Protocol:

c AI generated projects 登录复制 run

```c
EDKII_REDFISH_CREDENTIAL_PROTOCOL  mRedfishCredentialProtocol = {
  RedfishCredentialGetAuthInfo,
  RedfishCredentialStopService
};
```

Redfish-related authentication information is defined in the Redfish Host Interface Specification, but according to the specification, it has been deprecated, so it will not be introduced here.

EDK2 Redfish feature driver

The previous section introduces the Redfish framework in EDK, but does not introduce any real Redfish operations, nor does it implement the Redfish Client. This part of the content has not yet been put into the official version of UEFI, but is in the GitHub - tianocore/edk2-staging at edk2-redfish-client library.

This part will be introduced in subsequent articles.