UEFI Network Stack for EDK Getting Started Guide

January 31, 2008

Revision 0.2

Contents

§

Revision History

Revision Number	Description	Revision Date
0.1	Initial release.	August 28, 2006
0.2	Updated from EFI to UEFI; minor editing & formatting	January 31,2008

§

1 Introduction

The Unified Extensible Firmware Interface (UEFI) Network Stack is a collection of UEFI drivers that provide TCP/IP networking support for the EFI Developer Kit (EDK). The stack is a part of the EDK release, which you can download from www.TianoCore.org. This document provides instructions on how to build and use the network stack.

1.1 Overview

The UEFI Network Stack for the EDK (referred to in the remainder of this document as the “network stack” or the “stack”) implements the TCP/IP network interfaces defined in the *Unified Extensible Firmware Interface Specification,* Version 2.0. The interfaces implemented are *SNP, MNP, ARP, IP4, UDP4, DHCP4, MTFTP4, and TCP*. There are also two shell applications related to the network stack. One is the *ifconfig* used to configure the default address of the *IP* protocol. The other is the *ping* utility.

This document provides instructions to build and run the network stack under the EDK’s NT32 platform. You can follow similar steps to enable it for other platforms.

1.2 Related information

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

- *Extensible Firmware Interface Specification*, Version 1.10, Intel, 2001, http://developer.intel.com/technology/efi.
- *Unified Extensible Firmware Interface Specification*, Version 2.1, Unified EFI, Inc, 2007, http://www.uefi.org.
- *Intel® Platform Innovation Framework for EFI Specifications*, Intel, 2006, http://www.intel.com/technology/framework/.
- *EFI Developer Kit (EDK) Getting Started Guide*, Version 0.41. Intel, 2005, http://edk.tianocore.org/.
- *EFI Shell Getting Started Guide*, Version 0.31. Intel, 2005, http://efi-shell.tianocore.org/.

1.3 Terms

The following terms are used throughout this document to describe varying aspects of input localization:

DXE

Framework Driver Execution Environment phase.

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

The *Extensible Firmware Interface Specification* was published by Intel Corporation. Intel has donated the EFI specification to the Unified EFI Forum, and the UEFI now owns future updates of the EFI specification. See *UEFI Specification*.

Foundation

The set of code and interfaces that glue implementations of UEFI together.

Framework

Intel® Platform Innovation Framework for EFI consists of the Foundation, plus other modular components that characterize the portability surface for modular components designed to work on any implementation of the Tiano architecture.

GUID

Globally Unique Identifier. A 128-bit value used to uniquely name entities. A unique GUID can be generated by an individual without the help of a centralized authority. 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 EFI specification.

UEFI Application

An application that follows 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 that follows the UEFI specification.

UEFI Specification Version 2.1

Current version of the EFI specification released by the Unified EFI Forum. This specification builds on the EFI 1.10 and UEFI 2.0 specifications. Ownership of the EFI specification transferred from Intel to a non-profit, industry trade organization with version 2.0. See United EFI Forum (UEFI), below.

Unified EFI Forum

A non-profit collaborative trade organization formed to promote and manage the UEFI standard. For more information, see www.uefi.org.

1.4 Conventions used in this document

This document uses the typographic and illustrative conventions described below.

1.4.1 Pseudo-code conventions

Pseudo code is presented to describe algorithms in a more concise form. None of the algorithms in this document are intended to be compiled directly. The code is presented at a level corresponding to the surrounding text.

In describing variables, a list is an unordered collection of homogeneous objects. A queue is an ordered list of homogeneous objects. Unless otherwise noted, the ordering is assumed to be First In First Out (FIFO).

Pseudo code is presented in a C-like format, using C conventions where appropriate. The coding style, particularly the indentation style, is used for readability and does not necessarily comply with an implementation of the Extensible Firmware Interface Specification.

1.4.2 Typographic conventions

This document uses the typographic and illustrative conventions described below:

Plain text	The normal text typeface is used for the vast majority of the descriptive text in a specification.
Plain text (blue)	Any plain text that is underlined and in blue indicates an active link to the cross-reference. Click on the word to follow the hyperlink.
Bold	In text, a Bold typeface identifies a processor register name. In other instances, a Bold typeface can be used as a running head within a paragraph.
Italic	In text, an Italic typeface can be used as emphasis to introduce a new term or to indicate a manual or specification name.

`BOLD Monospace`	Computer code, example code segments, and all prototype code segments use a **`BOLD Monospace`** typeface with a dark red color. These code listings normally appear in one or more separate paragraphs, though words or segments can also be embedded in a normal text paragraph.
`Bold Monospace`	Words in a **`Bold Monospace`** typeface that is underlined and in blue indicate an active hyperlink to the code definition for that function or type definition. Click on the word to follow the hyperlink.
`Italic Monospace`	In code or in text, words in Italic Monospace indicate placeholder names for variable information that must be supplied (i.e., arguments).
`Plain Monospace`	In code, words in a **`Plain Monospace`** typeface that is a dark red color but is not bold or italicized indicate pseudo code or example code. These code segments typically occur in one or more separate paragraphs.

2 Development System Setup

This section describes the steps that are necessary to set up the local development system to build and run the UEFI network stack.

2.1 Install development tools

Several Microsoft tools are used to build the EDK with network support. The following tools must be installed on the development system:

- Microsoft Windows* 2000 operating system or Microsoft Windows XP operating system

For 32-bit Intel® architecture (IA-32) platform development:

- Microsoft Visual Studio .NET* 2003 Professional (7.1) development system

For Intel® Itanium® processor family development:

- Microsoft Windows Server 2003* Driver Development Kit (DDK) Build 3790.

2.2 Install the EDK

The UEFI Network Stack for the EDK has been included in the EDK release since Edk-Dev-Snapshot-20060807. You can download the EDK source trees from www.TianoCore.org. Please refer to the *EFI Developer Kit (EDK) Getting Started Guide* to build the EDK for NT32 platform.

§

3 Quick Start Guide

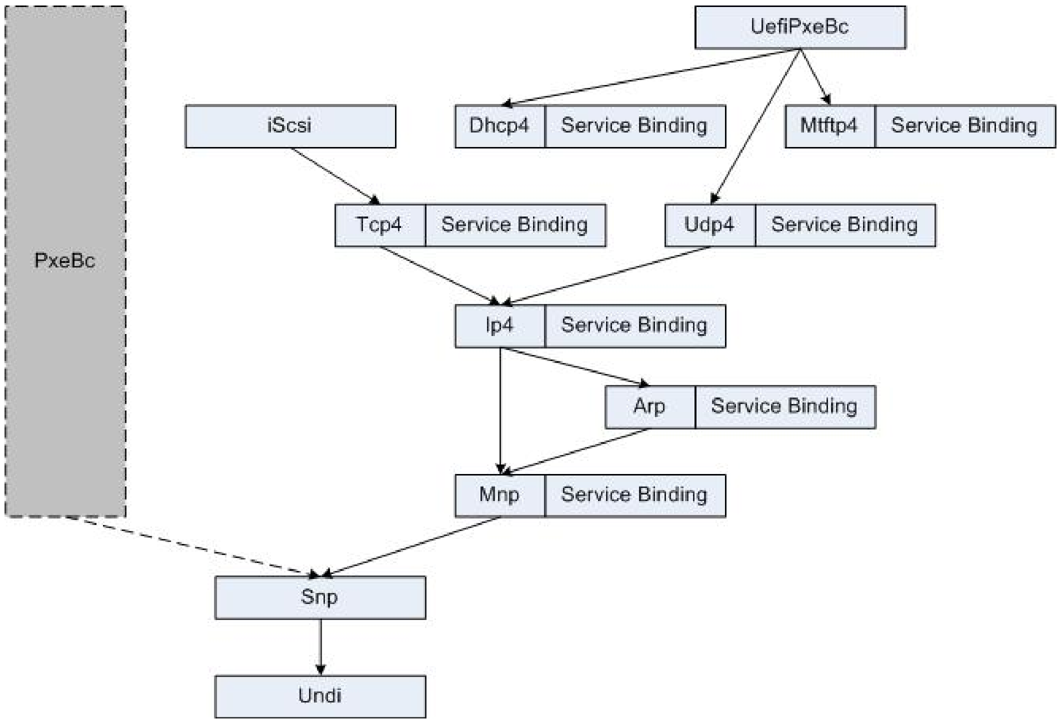
This guide assumes readers have installed the EDK and can build and run the NT32 tip. This guide gives detailed instruction to build and run the network stack for the NT32 platform and describes the differences for other platforms.

3.1 Building the EDK for NT32

Download the snapshots from www.TianoCore.org. Follow the steps in the *EFI Developer Kit (EDK) Getting Started Guide* to set up the NT32 tip build environment, assuming the EDK tree is put under C:\Edk. Build and run the system to make sure that the NT32 tip can boot to the shell.

3.2 Enabling the UEFI network source

There are two kinds of network stack in the EDK release. One is the original PXE stack. The other is the UEFI network stack. These two stacks conflict in that both of them will open the SNP driver exclusively. By default, the original PXE stack will be built into the firmware image, and the UEFI stack will be built as separate drivers. Thus, the original PXE stack has precedence over the UEFI stack. So, before building the UEFI network stack, you need to first disable the PXE stack.



The detailed steps are:

1. Open your platform's build file. For the NT32 platform in this example, the build file is c:\Edk\Sample\Platform\Nt32\Build\Nt32.dsc.

2. Comment out the PXE-related drivers, that is, add a '#' mark before the following lines:

```
Sample\Universal\Network\PxeBc\Dxe\BC.inf
Sample\Universal\Network\PxeDhcp4\Dxe\PxeDhcp4.inf
```

If you want to include the network stack in the firmware image, remove the **`FV=NULL`** from the following network stack related drivers. If not, skip to step 3.

```
Sample\Universal\Network\Dpc\Dxe\Dpc.inf                        FV=NULL
Sample\Universal\Network\Mnp\Dxe\Mnp.inf                        FV=NULL
Sample\Universal\Network\Arp\Dxe\Arp.inf                        FV=NULL
Sample\Universal\Network\Ip4\Dxe\Ip4.inf                        FV=NULL
Sample\Universal\Network\Ip4Config\Dxe\Ip4Config.inf            FV=NULL
Sample\Universal\Network\Udp4\Dxe\Udp4.inf                      FV=NULL
Sample\Universal\Network\Tcp4\Dxe\Tcp4.inf                      FV=NULL
Sample\Universal\Network\Dhcp4\Dxe\Dhcp4.inf                    FV=NULL
Sample\Universal\Network\Mtftp4\Dxe\Mtftp4.inf                  FV=NULL
Sample\Universal\Network\UefiPxeBc\Dxe\UefiPxeBc.inf            FV=NULL
Sample\Universal\Network\IScsi\Dxe\$(UEFI_PREFIX)IScsi.inf      FV=NULL
```

Note: *"UEFI_PREFIX" is a macro used in INF or DSC file to select source file or driver for EFI or UEFI build tip respectively.*

Note: *If you can't find the UEFI network stack build section in your platform's build file, follow the instructions in Section 5 on page 15 to add the section. Also make sure that your platform is using the EDK release after Dev-snapshot-20060807.*

3. Depending on the target platform for your build, choose either the SNP driver or the SNPNT32 driver:

- When building for the NT32 platform, comment out the SNP driver and enable the SnpNt32 driver, so the lines appear as follows:

```
Sample\Universal\Network\Snp32_64\Dxe\SNP.inf
Sample\Universal\Network\SnpNt32\Dxe\SnpNt32.inf  FV=NULL
```

- When building for a platform other than NT32, comment out the SnpNt32 driver and enable the SNP driver, as shown below:

```
Sample\Universal\Network\Snp32_64\Dxe\SNP.inf
#Sample\Universal\Network\SnpNt32\Dxe\SnpNt32.inf  FV=NULL
```

4. For platforms other than NT32, enable the UNDI driver for your particular LAN card. For example, if your LAN card is the Intel® PRO/100 Adapter, you should enable the following driver:

```
Sample\Bus\Pci\Undi\RuntimeDxe\Undi.inf
```

Get the UNDI driver for your particular LAN card from the card vendor. (The NT32 platform doesn't need the UNDI driver; it uses WinPcap* to transmit and receive network packets.)

5. Build the firmware for your platform. For example, to build the firmware for the NT32 platform, execute following commands:

```
c:\>  cd c:\Edk\Sample\Platform\Nt32\Build
c:\Edk\Sample\Platform\Nt32> set EDK_SOURCE=c:\Edk
c:\Edk\Sample\Platform\Nt32> nmake
```

If you are building for a platform other than NT32, this is all that you need to do to enable the network stack. Enabling the network stack for the NT32 platform requires the additional steps described in the following section.

3.3 Building the SnpNt32Io dynamic library

The SnpNt32 driver implements the EFI_SIMPLE_NETWORK_PROTOCOL for the NT32 platform. It depends on the WinPcap from www.winpcap.org to transmit and receive packets on Windows systems. To limit the number of symbols to import into the NT32 platform, SnpNt32 calls the function in the SnpNt32Io dynamic library to transmit or receive packets. The SnpNt32Io library in turn consumes the service provided by WinPcap.

To use the network stack on the NT32 platform, you need to install the WinPcap packet and build the SnpNt32Io library. The detailed steps are given in the following sections.

3.3.1 Downloading the SnpNt32Io source

The SnpNt32Io library is part of the Network-IO sub-project of the EDK project. You can find the URL linked to the project in the EDK project home. Download the source code and decompress it to c:\SnpNt32Io.

3.3.2 Installing WinPcap

You can find the WinPcap package at www.WinPcap.org. Follow the instructions there to download the *WinPcap auto-installer* which will install both the driver and its dynamic library.

You also need the *WinPcap developer's package* to build the SnpNt32Io library. Download it from the same site and decompress it to C:\SnpNt32Io. That is, the developer's package is located at C:\SnpNt32Io\WpdPack.

Note: *The SnpNt32Io library has been developed and tested based on the WinPcap 3.1 release. The WinPcap library contains two sets of functions: the pcap library functions and the packet library functions. The pcap library's interface is standardized and stable. However, the packet library may change from release to release. The SnpNt32Io DLL also uses the packet library to get the interface's MAC address. All of the calls to packet library functions are included in the* ***`SnpGetMac`*** *function. If you want to use a WinPcap release other than 3.1, please make sure that* ***`SnpGetMac`*** *works and that the* ***`SnpNt32`*** *driver can get the network adaptor's MAC address.*

3.3.3 Building the SnpNt32Io dynamic library

Execute the following commands to build the SnpNt32Io:

```
c:\>  cd c:\SnpNt32Io
c:\ SnpNt32Io> nmake
```

By default, this will build a debug version of the library. The generated DLL, SnpNt32Io.dLL, is created in c:\ SnpNt32Io\Debug. Use the following commands to build a release version of the library:

```
c:\>  cd c:\SnpNt32Io
c:\ SnpNt32Io> nmake TARGET=RELEASE
```

The release version of SnpNt32Io.dll is created in **`c:\SnpNt32Io\Release`**.

You need to copy SnpNt32Io.dll to the directory for NT32 drivers. If you are building the EDK for EFI, copy it to C:\Edk\Sample\Platform\Nt32\efi\IA32. Otherwise, copy it to C:\Edk\Sample\Platform\Nt32\uefi\IA32.

Note: *Whenever you clean the EDK's build by using the* ***`nmake clean`*** *command, everything under C:\Edk\Sample\Platform\Nt32\efi\Ia32 is deleted. You need to copy the SnpNt32Io.dll to that directory again after you execute the* ***`clean`*** *command.*

3.4 Testing the network stack with the NT32 emulator

Now, you can test the network stack with NT32 emulator. Type the following commands to start the NT32 emulator:

```
C:\>  cd C:\EDK\Sample\Platform\Nt32\Build
C:\EDK\Sample\Platform\Nt32\efi> System.Cmd
C:\EDK\Sample\Platform\Nt32\efi> nmake run
```

When the system has booted to the EFI shell, switch to the emulated disk containing the network drivers which is fsnt0: Type the following commands in the EFI shell to load the network drivers:

```
Shell   > fsnt0:
fsnt0:\> load SnpNt32.efi Mnp.efi Arp.efi Ip4.efi Ip4Config.efi
fsnt0:\> load Udp4.efi Dhcp4.efi Mtftp4.efi Tcp4.efi
```

Now, all the network stack drivers are loaded. You can use **`ifconfig`** to configure the IP4's default address. You can either assign it a static IP address or use DHCP protocol to retrieve the configuration from a DHCP server. Use the command **`ifconfig –h`** to find more information about how to use it.

For this example, we assume that your local network is 192.168.0.1/255.255.255.0. That is, 192.168.0.1 is the default gateway and 255.255.255.0 is the network mask.

Use the following commands to configure the network stack with DHCP and test whether the gateway is reachable:

```
fsnt0:\> ifconfig -s eth0 Dhcp
fsnt0:\> ping 192.168.0.1
```

If you want to use a static address, ask your network administrator to assign a static address for you. Assuming that the address assigned is 192.168.0.125, you can use the following commands to set the default address and ping the gateway or other host.

```
fsnt0:\> ifconfig -s eth0 static 192.168.0.125 255.255.255.0
           192.168.0.1

fsnt0:\> ping 192.168.0.1
```

Note: *Type the entire* ***`ifconfig`*** *command, including the address of the default gateway (192.168.0.1 in the example above) on the same line. In the example shown above, it is displayed as two lines because the entire command doesn't fit on one line.*

§

Directory Structure of the Network Stack

4.1 The top-level directory

The top-level directory structure for the EDK network stack is shown below.

```
Edk\
   Foundation\
   Sample\
```

The Foundation directory contains all of the protocol definitions, both the UEFI-defined network protocols and one implementation-specific protocol. All network drivers are in the Sample directory.

4.1.1 The Foundation directory

The Edk\Foundation directory contains all of the protocol definitions, including both the UEFI-defined protocols, such as protocols for **`EFI_IP4_PROTOCOL`**, and implementation-specific protocol definitions. There is only one implementation-specific protocol, the **`EFI_NIC_IP4_CONFIG_PROTOCOL`** protocol. This protocol is used by **`ifconfig`** to configure the default IP address.

The Edk\Foundation directory is organized as follows:

```
Foundation\
   Efi\
      Protocol\
         ARP\
         DHCP\
         IP4\
         IP4Config\
         ManagedNetwork\
         MTFTP4\
         ServiceBinding\
         TCP4\
         UDP4\
         IScsiInitiatorName\
         PxeBaseCode\
         PxeBaseCodeCallBack\
    Protocol\
       NicIp4Config\
       Dpc\
```

The UEFI-defined protocols are placed in the Foundation\Efi\Protocol directory, and the implementation-specific protocol is placed in the Foundation\Protocol directory.

4.1.2 The Sample directory

The Edk\Sample directory contains all of the drivers for the EDK network stack. The structure of this directory is as follows:

```
Sample\
   Universal\
      Network\
          ARP\
          DHCP4\
          DPC\
          IP4\
          IP4Config\
          Library\
          MNP\
          MTFTP4\
          SnpNt32\
          TCP4\
          UDP4\
          IScsi\
          UefiPxeBc\
```

All drivers are placed in the Sample\Universal\Network directory. Each driver has its own sub-directory. For example, the Sample\Universal\Network\IP4 directory contains the implementation for **`EFI_IP4_PROTOCOL`**. The SnpNt32 directory contains the **`EFI_SIMPLE_NETWORK_RPTOCOL`** implementation for the NT32 platform. The Library directory has the common support functions and definitions for the network stack, including buffer management functions, debug support routines, and other supporting modules.

§

5 Porting Guide

The first EDK release with the UEFI network stack is the 20060807 development snapshot. Any release after Aug 7, 2006 already includes the network stack code. However, the code might not be included in your platform's build tip. This section gives instructions on how to port the UEFI network stack to new platforms. After adding the UEFI network stack to the platform's build tip, you can follow the instructions in Section 3 to build the firmware with network stack support for your platform.

To add the network drivers to your tip, first open your platform's make file (in the examples, we'll assume the file is Build.dsc). Then, add the INF files for NetLib and the network drivers (MNP, ARP, IP4, IP4Config, UDP4, TCP4, DHCP4, MTFTP4, DPC, ISCSI and UEFIPXEBC) to it. The detailed steps are:

1. Add the following lines to the network library section of Build.dsc:

```
#
# Network library
#
Sample\Universal\Network\Library\NetLib.inf
```

Note: *You can find the library section by searching in your platform's build tip for phrase, such as "Lib." The NetLib.inf must be added to the correct PROCESSOR section that matches the build tip's processor type; for example, if you are building an X64 tip, the NetLib.inf should be added to the library section where the PROCESSOR is defined to be X64. Adding NetLib to the proper section keeps the build file clean.*

2. Add the following lines to the UEFI network drivers section of Build.dsc:

```
#
# UEFI network stack drivers
#
Sample\Universal\Network\Dpc\Dxe\Dpc.inf                         FV=NULL
Sample\Universal\Network\SnpNt32\Dxe\SnpNt32.inf                 FV=NULL
Sample\Universal\Network\MNP\Dxe\Mnp.inf                         FV=NULL
Sample\Universal\Network\ARP\Dxe\Arp.inf                         FV=NULL
Sample\Universal\Network\IP4\Dxe\IP4.inf                         FV=NULL
Sample\Universal\Network\IP4Config\Dxe\Ip4Config.inf             FV=NULL
Sample\Universal\Network\Udp4\Dxe\Udp4.inf                       FV=NULL
Sample\Universal\Network\Tcp4\Dxe\Tcp4.inf                       FV=NULL
Sample\Universal\Network\Dhcp4\Dxe\Dhcp4.inf                     FV=NULL
Sample\Universal\Network\Mtftp4\Dxe\Mtftp4.inf                   FV=NULL
Sample\Universal\Network\UefiPxeBc\Dxe\UefiPxeBc.inf             FV=NULL
Sample\Universal\Network\IScsi\Dxe\$(UEFI_PREFIX)IScsi.inf       FV=NULL
```

The DPC (Deferred Procedure Call) driver provides service for an UEFI Driver to queue a deferred procedure call at a lower TPL, which is typically used by UEFI network drivers to signal notify functions for sending and receiving packets. It's critical for the UEFI network stack and must be loaded in the first order. You have to include this driver in the firmware image, or manually load it ahead of others UEFI network modules in the shell.

The iSCSI (Internet Small Computer System Interface) driver provides iSCSI service in the preboot environment and supports booting over iSCSI.

The UefiPxeBc (UEFI Preboot Execution Environment Base Code) driver provides PXE boot service on UEFI network stack, while the original PxeBc driver is independent with UEFI network stack. The UefiPxeBc is introduced to co-work with other UEFI network stack drivers such as iSCSI, and it’s going to replace the original PxeBc driver in the long-term. You may enable only one of them for PXE boot service in a given build, and currently by default, the PxeBc is included in the generated firmware.

§