Compiling a UEFI Driver using the Intel® UEFI Development Kit 2010

November 2011

Revision 1.00

Contents

Figures

Introduction

This document describes how to compile a UEFI driver using the UEFI Developer's Kit 2010 (UDK2010), an open-source driver & application development kit containing EDK II (second generation EFI development kit) validated common-core sample code. The open-source UDK2010 is a stable build of the EDKII project, and has been validated on a variety of Intel platforms, operating systems, and application software. More information can be found on the UDK2010 SourceForge page.

This document is a simple reference list based on the UDK2010 and UEFI specification. More information about required and recommended protocols, the driver binding model, and boot services is available in the Intel® UEFI Driver Writer's Guide. Complete information on all required protocols and other elements for UEFI drivers is provided in the current UEFI specification.

The general steps to compile a UEFI driver are:

1. Download the UDK 2010 source from Tianocore.org
2. Configure the UDK 2010 build environment
3. Execute UDK 2010 build command
4. Add new UEFI driver to the UDK 2010 project
5. Add reference to new driver in platform description (.dsc)
6. Rebuild UDK 2010 with new driver included

After compiling a UEFI driver, basic functionality testing is recommended. A basic testing plan is covered later in this document:

1. Start the UDK 2010 NT32 Simulator
2. Verify Driver Load
3. Verify Driver Unload

Building a UEFI Driver from the Windows Command Line

This section describes how to build a functioning UEFI driver from a Microsoft Windows Command Prompt using UDK 2010 and Microsoft Visual Studio. The steps to successfully compile a UEFI Driver are:

- Download UDK 2010 Source from Tianocore.org
- Configure UDK 2010 Build Environment
- Execute UDK 2010 Build Command
- Add a new driver to the source tree
- Rebuild the UDK 2010 project to compile the new driver

Download UDK 2010 Source from Tianocore.org

The UDK2010 source code tree is available on the tianocore.org website hosted by SourceForge. There are two download methods available:

1. Download a ZIP file that contains a snapshot release of UDK 2010.
2. Download the UDK 2010 project directly from the SourceForge Subversion (SVN) source control server.

A link for direct download of the ZIP file can be found on the UDK2010 project website at SourceForge. Connecting to the SVN repository is covered below.

> *Note: Using source from the ZIP file requires an additional configuration step to configure the build tools. See "Configure UDK 2010 Build Environment" below for details.*

Using SVN to Download UDK 2010 Source

The following steps show how to download UDK2010.UP4 from the SVN repository.

1. Install the Subversion control tool TortoiseSVN from http://tortoisesvn.net/. Please refer to the Appendix ("Installing SVN") for more details.
2. Create a Tianocore.org user account if you do not have one already: https://sourceforge.net/account/registration/
3. Create a directory for the source (recommended: **`C:\UDK2010.UP4`**)
4. Right click on source directory and select TortoiseSVN -> Export from the pop-up menu. The SVN Export dialog will be displayed.

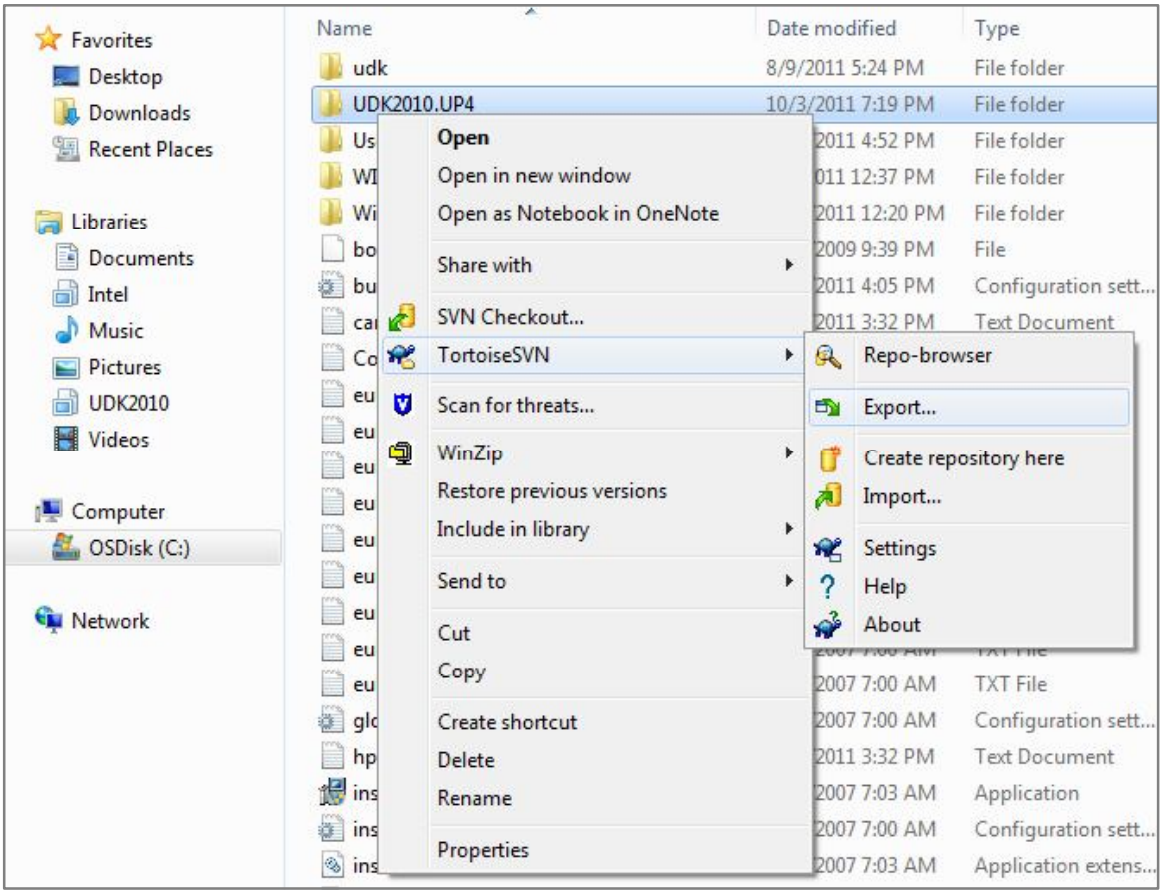

Figure 1 - TortoiseSVN Menu

5. In the URL of Repository field enter: **`http://edk2.svn.sourceforge.net/svnroot/edk2/branches/UDK2010`**
6. Enter the local working directory in the "Export directory" field. This document uses **`C:\UDK2010.UP4\`** in the examples.
7. In the Revision field enter **11579** (revision number for UDK2010.UP4)
8. Click **OK** to start downloading the UDK2010.UP4 source project.

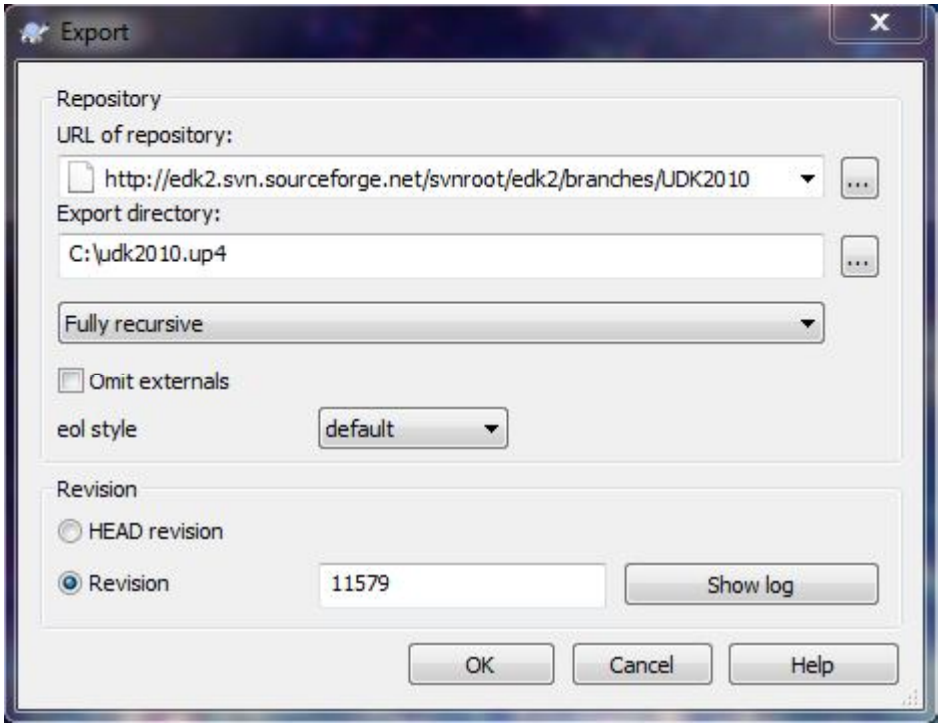

Figure 2 - TortoiseSVN Export Dialog

Configure UDK 2010 Build Environment

Configuring the UDK 2010 build environment is a key part of the build process. Configuration of the environment differs based on operating system and compiler.

BaseTools Configuration

Projects downloaded from SVN will already contain the build tools directories (**`BaseTools`** and **`Conf`**). Users who install the project from ZIP file must unpack the tools for their operating system environment.

- Microsoft Windows - **`BaseTools(Windows).zip`**
- Linux - **`BaseTools(Unix).tar`**

Examples in this document refer to the Microsoft Windows tools, but the process is similar for Linux-based systems. The most recent procedure for setting up the UDK 2010 environment is available in the release notes for each build.

Additional Information

The release notes can be viewed by opening **`ReleaseNotes.txt`** found in the top level of the source directory (example "**`C:\UDK2010.UP4\ReleaseNotes.txt`**"). Please refer to the "UDK 2010 Supported Tool Chains" section of this document for additional details.

Execute UDK 2010 Build Command

Before starting development, it is a good to make sure the project builds "out of the box" (works properly before changes are made). This section describes how to compile the NT32 environment from the Microsoft Windows command prompt. NT32 allows the UEFI Shell to execute under Microsoft Windows as an intermediate development environment.

This process uses the UDK 2010 build tools and Microsoft Visual Studio 2008 C compiler. Refer to the Appendix ("UDK 2010 Supported Tool Chains") for more information on supported build environments.

Basic EDK Setup

From the Start Menu, open the Visual Studio Command Prompt. From the Command Prompt, navigate to root directory of the UDK 2010 installation and run the **edksetup.bat** command.

> *Note: For this example using the NT32 platform, the **edksetup –nt32** command will be executed. For other environments, use the **edksetup** command without any parameters.*

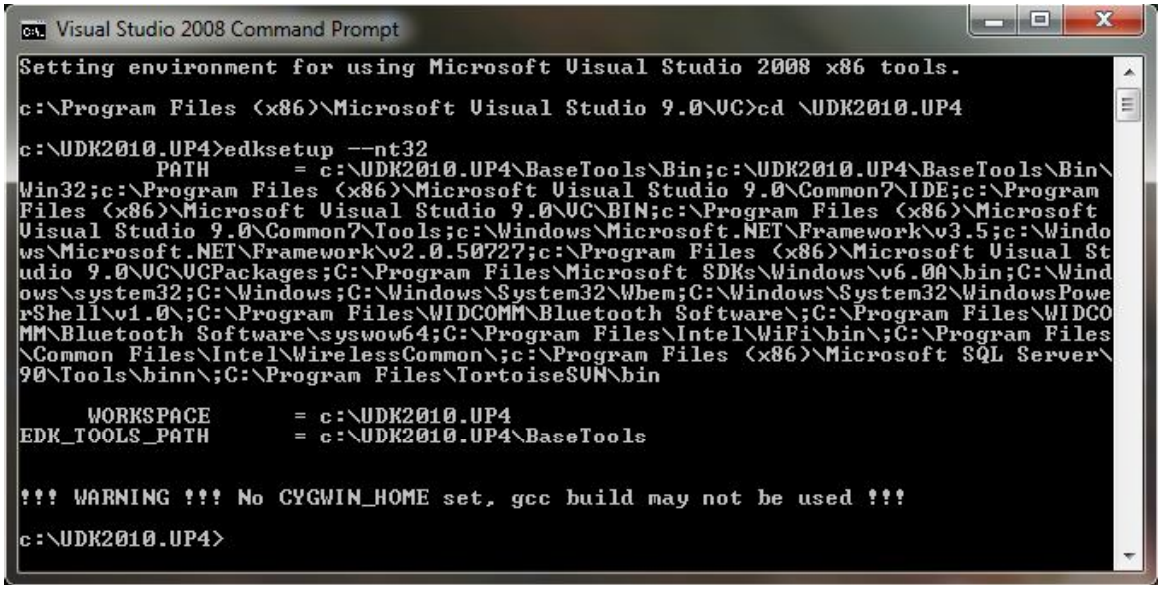

Figure 3 - Visual Studio Command Prompt with edksetup for NT32

The **edksetup** command configures the required environment variables to support the UDK 2010 build process.

Build Target Configuration

Files in the Conf directory (**C:\UDK2010.UP4\Conf**) are used to define default build parameters for UDK2010 projects. Use the **target.txt** file to set the following

default parameters, based on the NT32 platform and Microsoft Visual Studio 2008 tool chain:

```
ACTIVE_PLATFORM       = Nt32Pkg/Nt32Pkg.dsc
TARGET                = DEBUG
TARGET_ARCH           = IA32
TOOL_CHAIN_TAG        = VS2008x86
```

Note that the TOOL_CHAIN_TAG value depends on your compiler and system configuration. Refer to **`Conf\tools_def.txt`** for details on supported configurations.

Issue Build Command

With the environment variables and build configuration set, the environment can be compiled by simply using the build command. Once **`Conf/target.txt`** is properly configured, type **`build`** at the command prompt and hit **`Enter`**.

The build process will return any errors in the Command Prompt window. The initial build should result in no errors if the environment is configured correctly.

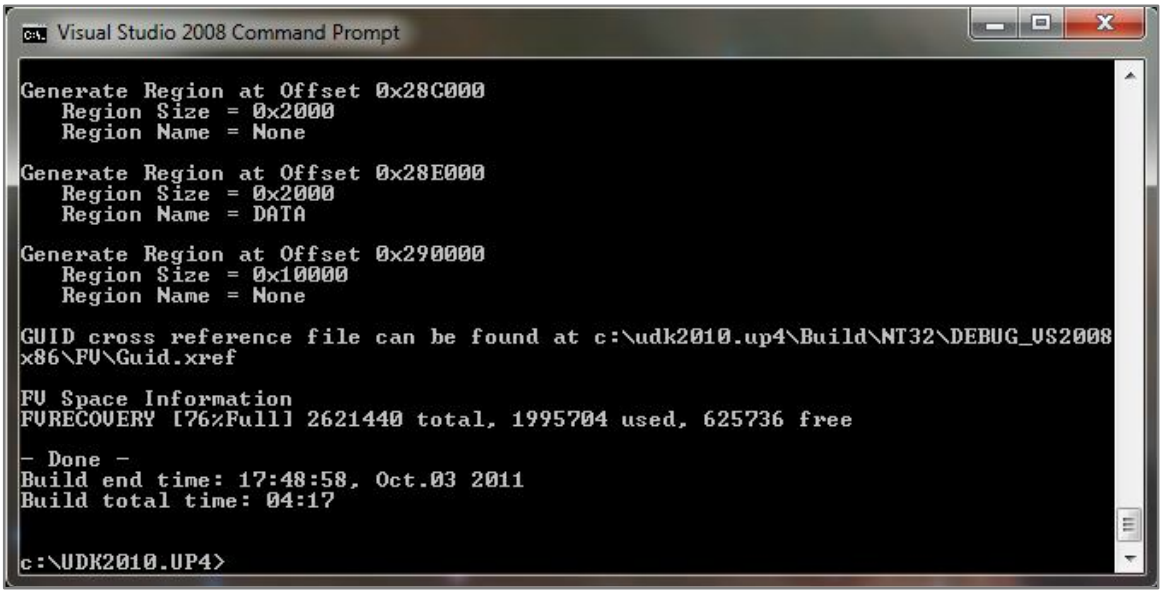

Figure 4 - Results of Successful Build for NT32 Platform

Run the NT32 Environment

Unlike other UEFI firmware, NT32 does not have to be flashed onto physical hardware to run. After building NT32, type **`build run`** at the command prompt.

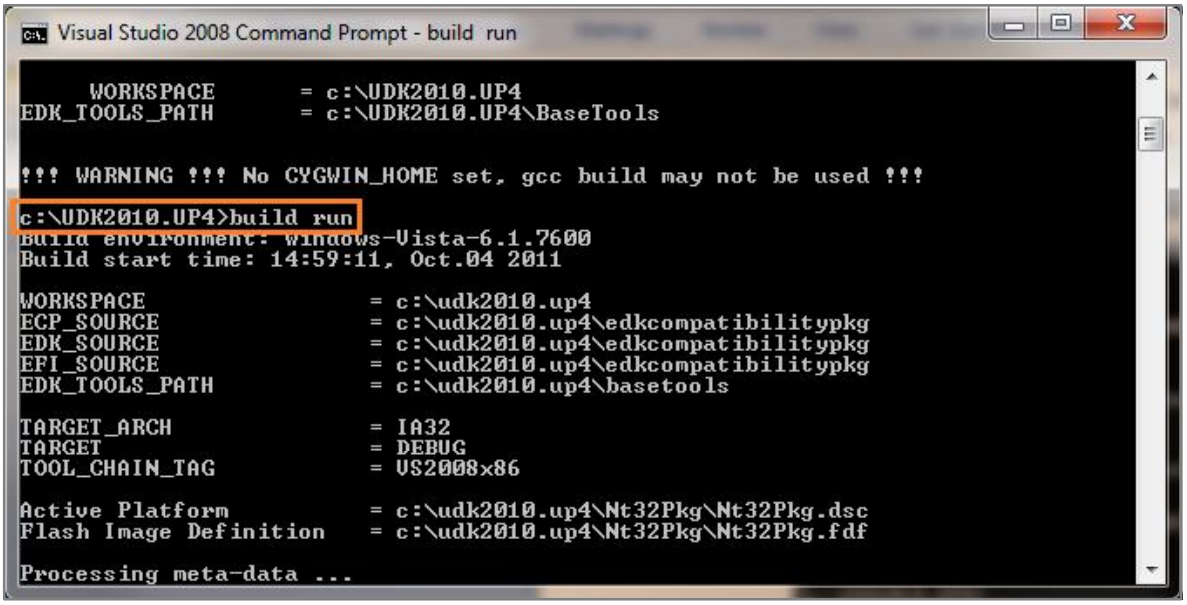

Figure 5 - Using 'build run' command to launch NT32

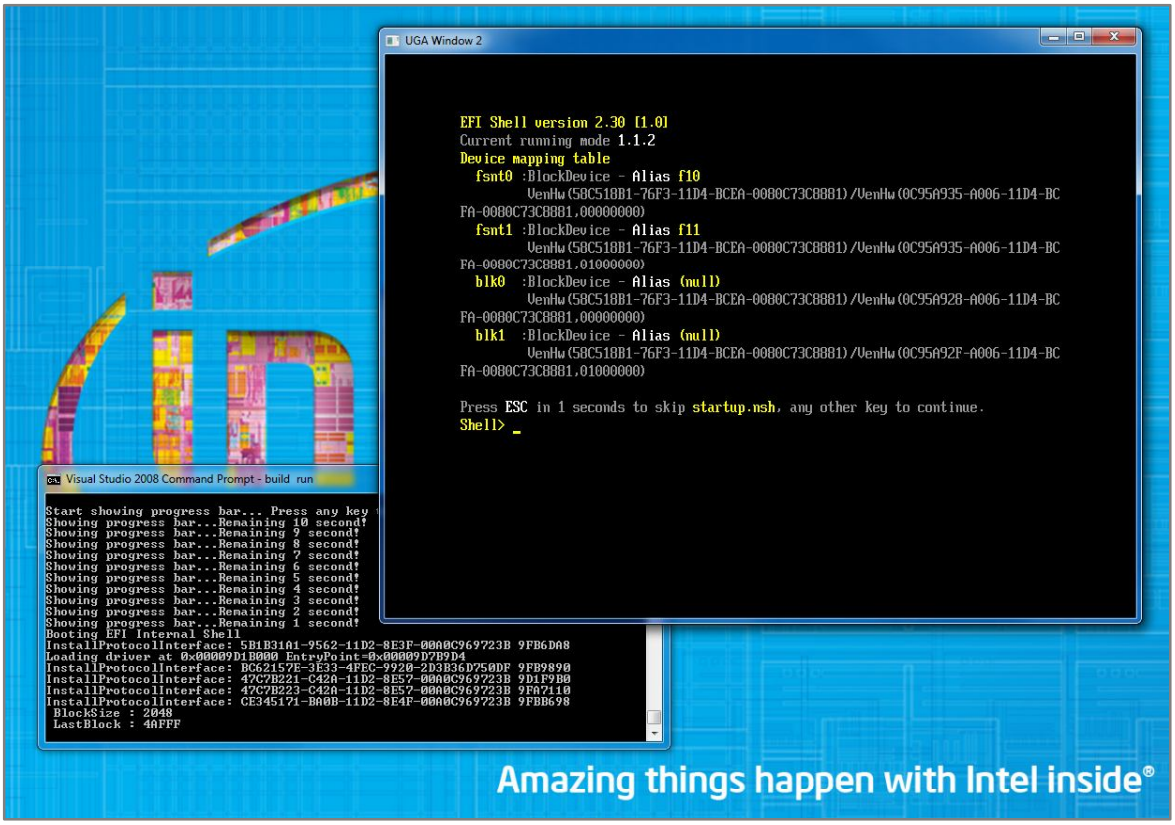

Figure 6 - NT32 environment (right) with debug messages (left)

Add a New Driver to the Source Tree

A sample UEFI driver (**BlankDrv**) is available to use as a starting point for UEFI development. **BlankDrv** is a general purpose driver that contains required and recommended UEFI protocols but manages no hardware. The BlankDrv.zip package is available for download from tianocore.org.

> *Note: make sure the NT32 environment in the previous example has been closed before making any changes to the UDK 2010 project.*

Create a new directory named **BlankDrv** in the **OptionRomPkg** directory under the root UDK2010 folder (**C:\UDK2010.UP4\OptionRomPkg**) and extract the ZIP file.

Once unpacked, check the contents of the **BlankDrv** directory:

C:\UDK2010.UP4\OptionRomPkg\BlankDrv\BlankDrv.c.txt
C:\UDK2010.UP4\OptionRomPkg\BlankDrv\BlankDrv.h.txt
C:\UDK2010.UP4\OptionRomPkg\BlankDrv\BlankDrv.inf.txt
C:\UDK2010.UP4\OptionRomPkg\BlankDrv\ComponentName.c.txt
C:\UDK2010.UP4\OptionRomPkg\BlankDrv\Readme.txt

Rename the code example files (**.c, .h, .inf**) to remove the **.txt** extensions:

C:\UDK2010.UP4\OptionRomPkg\BlankDrv BlankDrv.c
C:\UDK2010.UP4\OptionRomPkg\BlankDrv BlankDrv.h
C:\UDK2010.UP4\OptionRomPkg\BlankDrv BlankDrv.inf
C:\UDK2010.UP4\OptionRomPkg\BlankDrv ComponentName.c
C:\UDK2010.UP4\OptionRomPkg\BlankDrv\Readme.txt

Add UEFI Driver Reference to Target Platform

In order to build a UEFI driver, a reference to the driver must first be added to a target platform. Several target reference platforms included in the default UDK2010 distribution. For the purposes of this document, we will use the open source NT32 simulation reference platform. NT32 is an emulation environment designed to run on a Microsoft Windows platform and is not available for other operating systems.

To build the sample driver downloaded in the previous section a reference to the driver **.inf** file must be inserted into the component's section of the target platform Description File (.DSC). This example builds a driver for the IA32 architecture under the NT32 platform.

Perform the following steps to add **BlankDrv** to the NT32 platform:

1. Open **C:\UDK2010.UP4\Nt32Pkg\Nt32Pkg.dsc** in a text editor.
2. Find the **[Components.IA32]** section in the **.dsc** file. This list contains all of the drivers assembled when compiling the NT32 platform. Note that the path references are relative to the root project directory (**C:\UDK2010.UP4**).
3. Add a reference to UEFI driver by entering path to the driver's **.inf** file at the end of the module list (**OptionRomPkg/BlankDrv/BlankDrv.inf**).

```
[Components.IA32]
  ##
  #  SEC Phase modules
  ##
  Nt32Pkg/Sec/SecMain.inf

  …

MdeModulePkg/Universal/PlatformDriOverrideDxe/PlatformDriOverrideDxe.inf

OptionRomPkg/BlankDrv/BlankDrv.inf
```

4. Save and close the **.dsc** file.

Build New Driver in NT32 Project

In this situation, the build command can be issued with additional parameters to compile a specific driver (**-m**) in the platform (**-p**) instead of the entire platform.

```
build -p Nt32Pkg\Nt32Pkg.dsc -m OptionRomPkg\BlankDrv\BlankDrv.inf
```

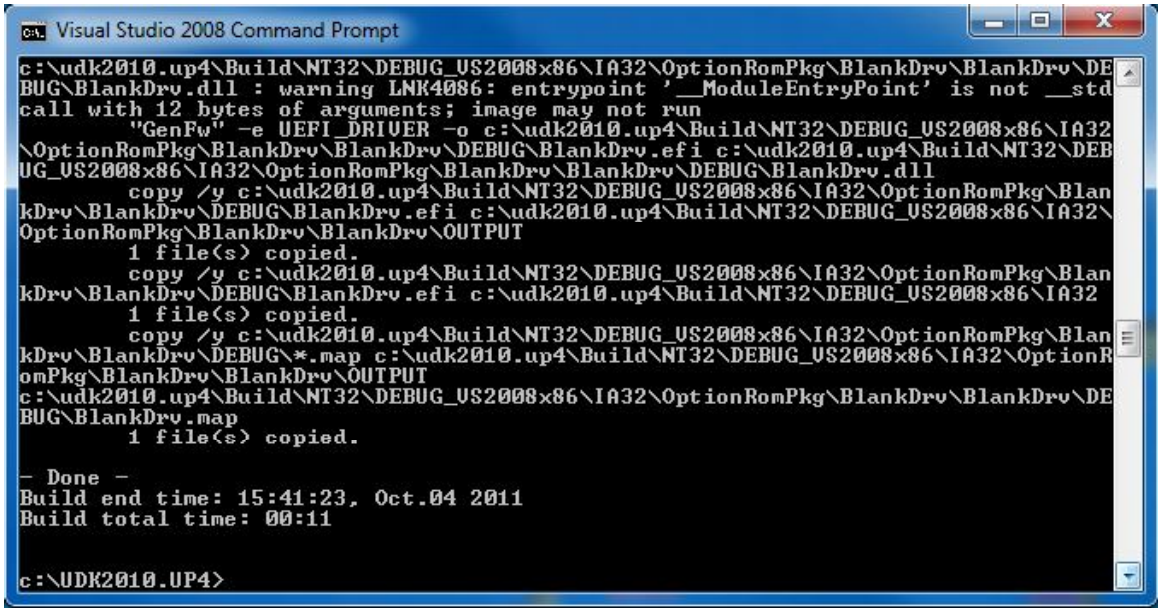

Figure 7 - Results of successful build (BlankDrv)

After successfully completing the build process for **BlankDrv**, the driver will be part of the platform's build directory. In this example, **BlankDrv** is used by a debug build of the NT32 platform:

```
C:\UDK2010.UP4\Build\NT32\DEBUG_VS2008x86\IA32\BlankDrv.efi
```

Testing the UEFI Driver

This section details how to perform simple testing of your UEFI driver using the NT32 simulator and UEFI Shell. Use the information from the "Run the NT32 Environment" section to launch the NT32 environment after building the sample driver (**BlankDrv**) described in the "Build New Driver in NT32 Project" section.

Verify Driver Load

The driver **load** command causes the system to load the driver into the UEFI driver handle data base. After launching NT32 using the build run command in the Visual Studio Command Prompt, execute these commands from the UEFI Shell:

```
fsnt0:
load BlankDrv.efi
```

Figure 8 - Result of load command (BlankDrv.efi)

The **drivers** command displays a list of all drivers currently managed. The image below displays a sample output from the **drivers** shell command. If the load command was successful, the recently loaded device driver should be listed.

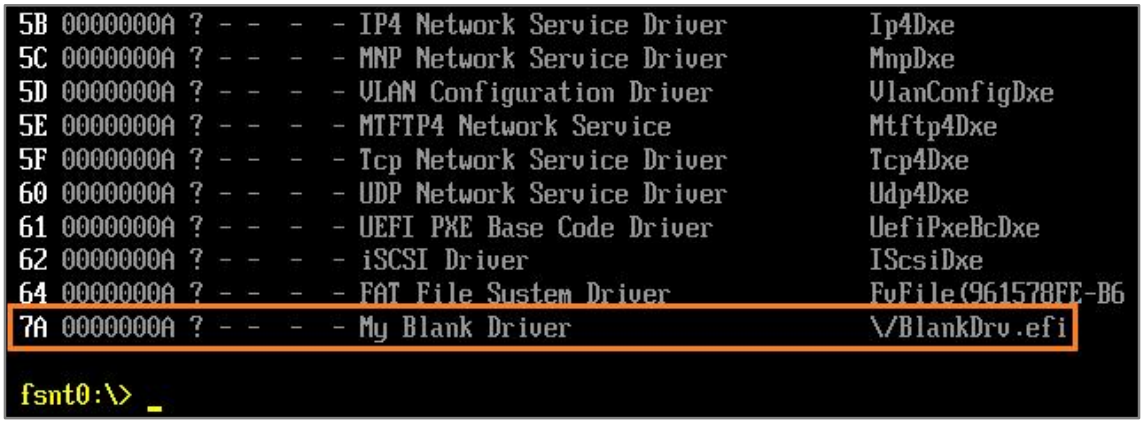

Figure 9 - Results of drivers command for BlankDrv.efi

In this example, **BlankDrv** is the last driver displayed by the drivers command. Each loaded driver is issued a unique driver handle (for **BlankDrv** this is **7A**). Use the UEFI Shell driver handle (**dh**) command to see additional information (example below).

```
fsnt0:\> dh 7A
Handle 7A (09EA5E10)
   Image (9EC6C28)   File:\/BlankDrv.efi
     ParentHandle..: 9EC9910
     SystemTable...: 5E50F90
     DeviceHandle..: 9ECFA10
     FilePath......: \/BlankDrv.efi
     PdbFileName...:
c:\udk2010.up4\Build\NT32\DEBUG_VS2008x86\IA32\OptionRomPkg\BlankDrv\Bl
ankDrv\DEBUG\BlankDrv.pdb
     ImageBase.....: 9E92000 - 9E99000
     ImageSize.....: 7000
     CodeType......: BS_code
     DataType......: BS_data
   ImageDpath (9EA9290)
      Hardware Device Path for Vendor-Defined
       Guid WinNtThunk
      Hardware Device Path for Vendor-Defined
       Guid 0C95A935-A006-11D4-BCFA-0080C73C8881
       00000000: 00 00 00 00
*....*

      Media Device Path for File Path
       File '\'
      Media Device Path for File Path
       File 'BlankDrv.efi'
      AsStr: 'VenHw(58C518B1-76F3-11D4-BCEA-
0080C73C8881)/VenHw(0C95A935-A006-11D4-BCFA-
0080C73C8881,00000000)/\/BlankDrv.efi'
   DriverBinding (A974090)
   ComponentName (A9740A8)
   ComponentName2 (A9740B4)
```

Verify Driver Unload

Use the **unload** command to unload a driver from the driver handle data base. The result of the command should be successful. After unloading a driver, use the driver handle command (**dh**) to verify that the driver has been unloaded.

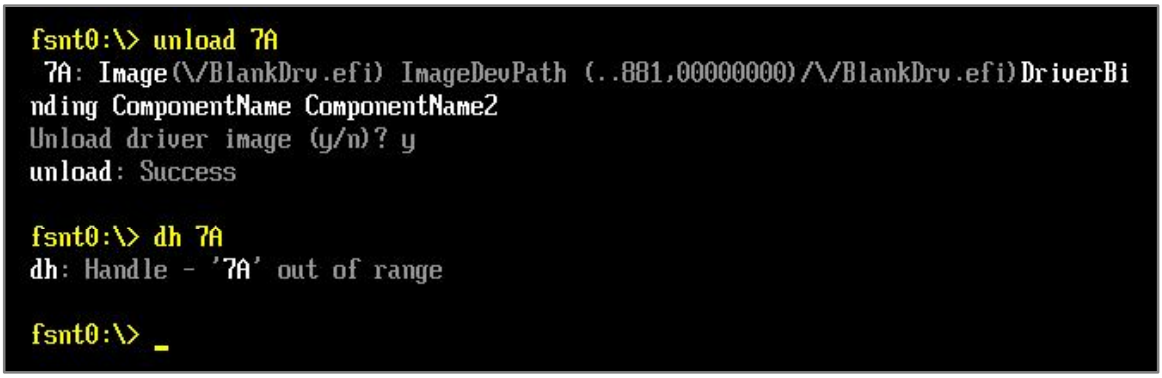

Figure 10 - Successful driver unload command (BlankDrv.efi)

The command should return that the specified device handle is out of range.

Continuing Driver Development

The **BlankDrv** template is a good point for starting development on UEFI drivers. For additional information, please refer to the "UEFI Driver and Application Tool Resources" page at intel.com or the UDK2010 project page at SourceForge.net.

Appendix

UDK 2010 Supported Tool Chains

This section describes how to view the tool chains supported by a given release of UDK 2010 and details how to update the build command line and build target. For using other tools see the "getting started section" of tianocore.org: http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=Getting_Started_with_EDK_II

To view the list of supported tool chains for the UDK2010 build process:

1. Open a Microsoft Visual Studio Command prompt and navigate to the root directory for the UDK 2010 installation (**C:\UDK2010.UP4**).
2. Execute the batch file **edksetup.bat** to configure the build environment.
3. Open **Conf\tools_def.txt** in a text editor.
4. Navigate to section titled **Supported Tool Chains**.

```
# Supported Tool Chains
# ====================
#   VS2003 - win32 - Microsoft Visual Studio .NET 2003, Intel EBC, Intel ASL
#   VS2005 - win32 - Microsoft Visual Studio 2005 Team Suite Edition, Intel EBC,
Intel ASL (also compatible with VS 2005 Express, Standard, and Pro)
#   VS2008 - win32 - Microsoft Visual Studio 2008 Team Suite with Intel ASL
#   DDK3790 - win32 - Microsoft Windows DDK 3790.1830, Intel EBC, Intel ASL
#   UNIXGCC - UNIX GCC, No EBC, Intel ASL
#   GCC44  - Linux - GCC 4.4 (Native), No EBC, Intel ASL
#   ELFGCC - Linux ELF GCC, No EBC, Intel ASL
#   CYGGCC - win32 - CygWin GCC, Intel EBC, Intel ASL
#   ICC - win32 - Intel C Compiler V9.1, Intel EBC, Intel ASL
#   ICC11 - win32 - Intel C Compiler V11.1, Intel EBC, Intel ASL
#   MYTOOLS - win32 - VS2008 for IA32/X64, WINDDK 3790.1830 for IPF, Intel EBC,
Intel ASL
#   VS2003xASL - win32 - Microsoft Visual Studio .NET 2003, Intel EBC,
Microsoft ASL
#   VS2005xASL - win32 - Microsoft Visual Studio 2005 Team Suite Edition, Intel
EBC, Microsoft ASL
#   VS2008xASL - win32 - Microsoft Visual Studio 2008 Team Suite with Microsoft
ASL
#   DDK3790xASL - win32 - Microsoft Windows DDK 3790.1830, Intel EBC, Microsoft
ASL
#   CYGGCCxASL - win32 - CygWin GCC, Intel EBC, Microsoft ASL
#   ICCxASL - win32 - Intel C Compiler V9.1, Intel EBC, Microsoft ASL
#   ICC11xASL - win32 - Intel C Compiler V11.1, Intel EBC, Microsoft ASL
#   VS2005x86 - win64 - Microsoft Visual Studio 2005 Team Suite Edition (x86),
Intel EBC, Intel ASL (also compatible with VS 2005 Express, Standard, and Pro)
#   VS2008x86 - win64 - Microsoft Visual Studio 2008 (x86) w/ Intel ASL
#   ICCx86                - win64 - Intel C Compiler V9.1 (x86), Intel EBC,
Intel ASL
```

```
#   ICC11x86  - win64 - Intel C Compiler V11.1 (x86), Intel EBC, Intel ASL
#   VS2005x86xASL - win64 - Microsoft Visual Studio 2005 Team Suite Edition
(x86), Intel EBC, Microsoft ASL (also compatible with VS 2005 Express, Standard,
and Pro)
#   VS2008x86xASL - win64 - Microsoft Visual Studio 2008 (x86) w/ Microsoft ASL
#   ICCx86xASL  - win64 - Intel C Compiler V9.1 (x86), Intel EBC, Microsoft ASL
#   ICC11x86xASL - win64 - Intel C Compiler V11.1 (x86), Intel EBC, Microsoft
ASL
#   CYGGCCx86 - win64 - CygWin GCC (x86), Intel EBC (x86), Intel ASL
#   CYGGCCx86xASL - win64 - CygWin GCC (x86), Intel EBC (x86), Microsoft ASL
```

The section consists of several rows, where each row consists of a supported tool chain name and followed by a brief description of the architecture, and compiler. The tool chain can be set from a configuration file or the command line.

Configuration File (target.txt)

The tool chain used by the UDK 2010 build environment can be set from the command line and in the file **Conf/target.txt**.

The default value of **TOOL_CHAIN_TAG** is MYTOOLS, which may not match the compiler you have installed. Two examples using Microsoft Visual Studio 2008 are shown below.

- Microsoft Visual Studio 2008 on Microsoft Windows XP (32-bit):
  `TOOL_CHAIN_TAG = VS2008`
- Microsoft Visual Studio 2008 on Microsoft Windows 7 (64-bit):
  `TOOL_CHAIN_TAG = VS2008x86`

Command Line Configuration

The same values used in **TOOL_CHAIN_TAG** can be used on the command line to override the compiler configuration in **Conf/target.txt**.

```
build -p Nt32Pkg\Nt32Pkg.dsc -m OptionRomPkg\BlankDrv.inf –t
VS2008x86
```

This example sets the platform (**-p Nt32Pkg\Nt32Pkg.dsc**), tool chain (**–t VS2008x86**) and module (**-m OptionRomPkg\BlankDrv.inf**) from the command line.

Compiler Installation Path

Note the difference in VS2008 and VS2008x86 are the compiler install path, which changes for a 32-bit application on a 64-bit OS. The paths are assumed to be the default path provided by the installation program. These paths can be altered by changing the values in **Conf/tools_def.txt** but this is not recommended.

Installation Tips for Microsoft Visual Studio 2008

When installing Microsoft Visual Studio 2008, be aware that the x64 compiler tools are not installed by default. These tools are required for x64 platforms and drivers.

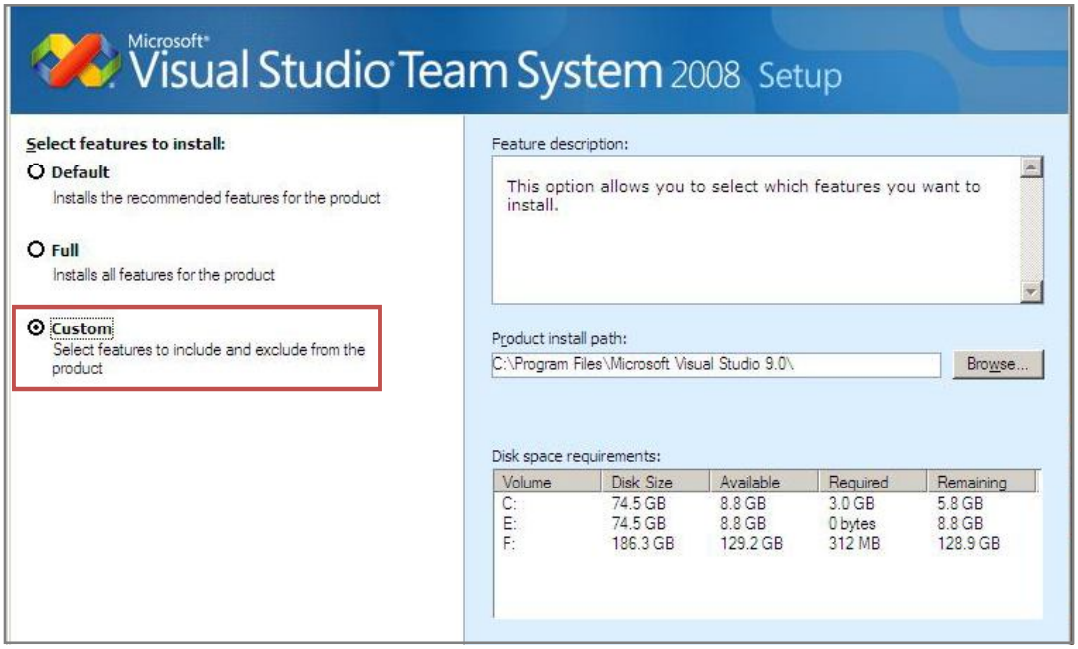

Figure 11 - Custom install process for VS2008

Installing the x64 tools requires a custom installation, even on a native x64 operating system such as Microsoft Windows 7.

Figure 12 - Installing x64 tools for VS2008

Installing SVN

This chapter details steps to download, install, and configure the Tortoise SVN client.

1. Open a Web browser to the URL: **`http://tortoisesvn.net/downloads`**

2. Download the appropriate application installation package based on operating system (32bit version for Microsoft Windows XP or 64bit version for Microsoft Windows 7 x64).
3. Install TortoiseSVN by double clicking on the downloaded **`.msi`** file.

Figure 13 - Selecting TortoiseSVN version

4. Restart the machine as requested.
5. Based on your network it may be necessary to update the proxy settings for SVN. Open File explorer, right click to open the popup menu, choose **`TortoiseSVN -> Settings.`**

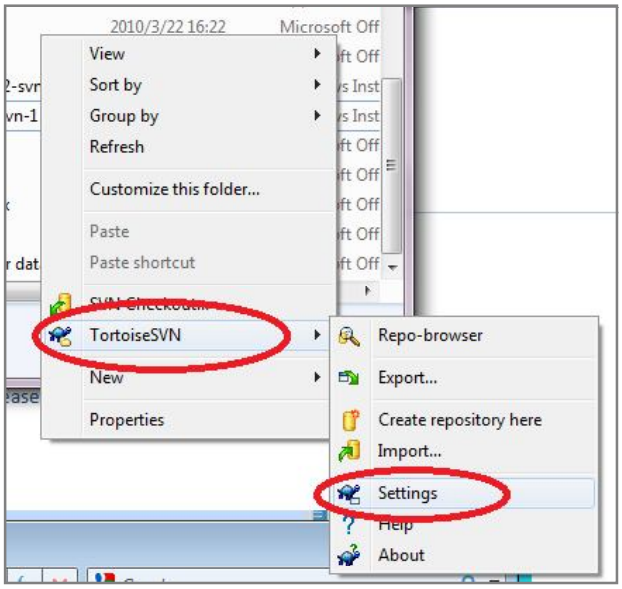

Figure 14 – TortoiseSVN Settings Menu

6. Click “Network” from the left side of the dialog.
7. If necessary, check the “Enable Proxy Server” box. Update the proxy settings server address, port and credentials as required by your network.

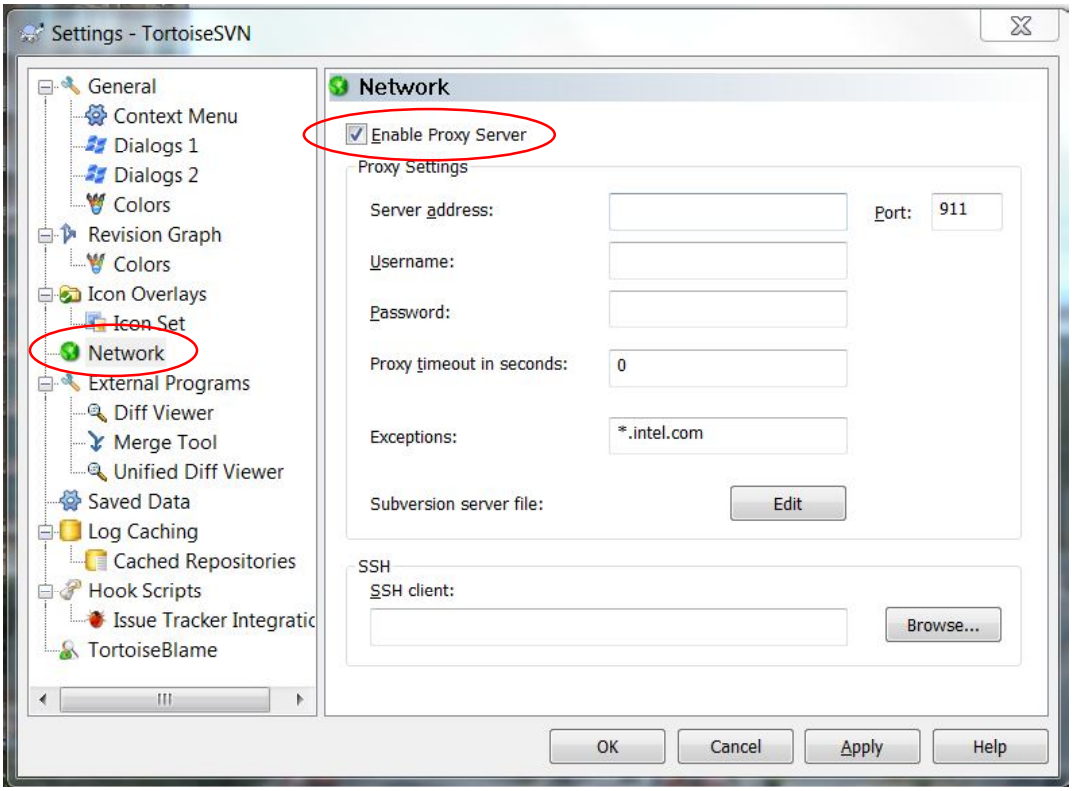

Figure 15 - TortoiseSVN network configuration

8. Click OK to save the configuration.

Additional Resources

Intel® UEFI Development Kit Debugger Tool

The Intel® UDK Debugger Tool provides the ability to debug UDK based firmware running on an IA32 family processor using WinDbg through a debug cable, by co-working with the target side component of the debug solution. The target side components (debug agent) could be found in as the SourceLevelDebugPkg under EDK II project, which is maintained on www.TianoCore.org
http://www.intel.com/content/www/us/en/architecture-and-technology/unified-extensible-firmware-interface/intel-uefi-development-kit-debugger-tool.html