Lab 1: A Basic Driver

# Overview

Before beginning Kernel development, the environment has to be set up. It consists of a Host PC (Development tools, debugger) running the development environment, and a Target PC (Client, Target, Debugee) for running and testing the driver. In this lab the target PC is a virtual machine.

Then, we’ll use the skeleton KMDF driver that Visual Studio 2012 wizard generates for us.

After successful compilation we’ll copy the package to the target machine and install it there.

Finally, we’ll use WinDBG to debug our driver.

## Goal

* Build the wizard generated driver
* Install and debug

## Setup



# Setting up the build environment

This chapter makes the students familiar with the WDK development environment in Visual Studio 2012

## Step 1: Installation of the development environment on the Host machine

Install the software development tools below on the Host (Debugger) machine in exactly this sequence:

* Visual Studio 2012
* Windows 8 SDK
* Windows 8 WDK
* Windows 8 WDK Co Installer

## Step 2: Setting up the virtual Target machine environment

1. Install the virtual machine environment on the Host machine:
   * VMWare Workstation 9.0
   * VMWare Player 5.0 or above
2. Install the guest operating system on the virtual Target Machine:
   * Microsoft Windows 8
3. Install “VMWare Tool”. This is necessary for copy-paste from the host to the virtual machine.

## Step 3: Configure machine for debugging

1. Set configuration for debugging. Open an elevated command prompt and type:

C:\> bcdedit /debug ON

C:\> bcdedit /dbgsettings NET HOSTIOP:x.x.x.x PORT:50000 KEY:1.2.3.4

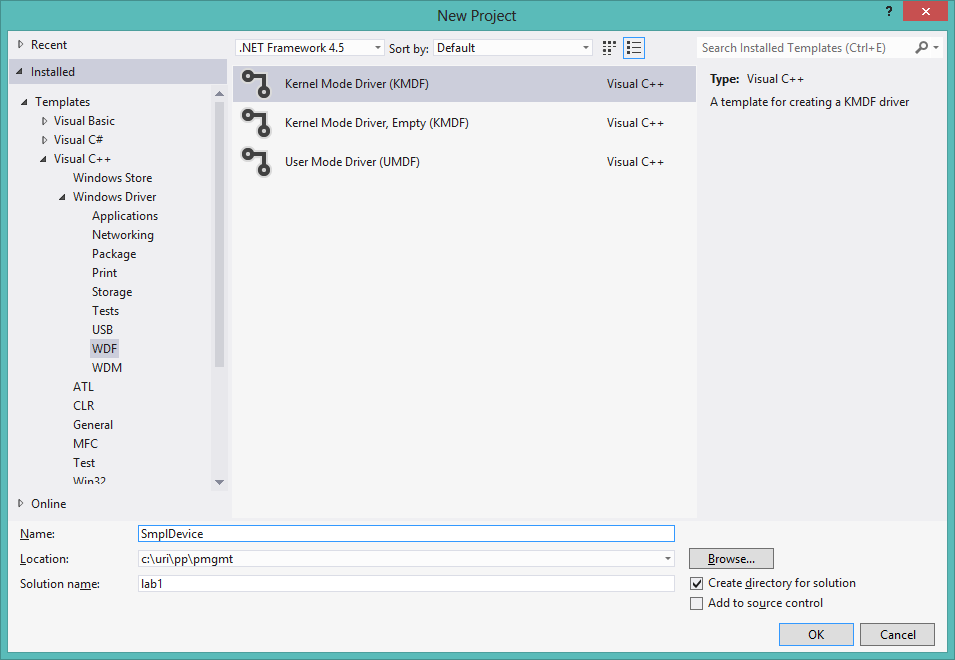
Replace x.x.x.x with the IP address of the host computer.

1. Hint: In case the target is a 64 bit operating system, the setting below might help installing unsigned drivers:

bcdedit -set loadoptions DDISABLE\_INTEGRITY\_CHECKS

## Step 4: Creating and compiling a driver template on the Host machine

1. Create a working directory named lab1. In this directory, create a new Visual Studio project of type “Windows Driver” -> “Kernel Mode Driver (KMDF)”. The new project shall be named SmplDevice.



1. Optional: Rename the project ‘SmplDevice Package’ to ‘Package’. Typing paths will be simpler this way. Do this by exit Visual Studio, and rename the folder ‘SmplDevice Package’ as well as rename the \*.vcxproj file. Restart Visual Studio.
2. Change the configuration to ‘Win8 Debug’

After compiling you’ll have a Package folder ready for installation.

# Setting up the test environment

This chapter makes the students familiar with the WDK test environment in Visual Studio 2012.

## Step 1: Prepare the virtual target machine for deployment, debugging and testing

Enable the existing local Administrator account (it is disabled by default) on the virtual Target machine and assign a password to the local administrator account.



In VMWare “Virtual Machine Settings” set the network adapter to NAT (Network Address Translation) and make sure that a network connection exists between Host and virtual Target machine (using ipconfig and ping commands).



Make sure you can connect to the Target's share “users” from the host. In case the connection to the share is not successful, disconnect all other network connections on the host (e.g. company network, wireless, internet, etc.).

Make sure you can connect the Host computer from the Client computer.

## Step 2: Manual install the driver to the target machine

1. Create a folder on the host machine and copy the compiled package

C:\> mkdir t

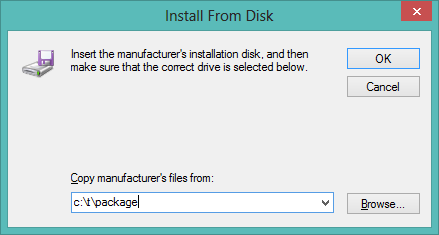
C:\> cd c:\t

C:\t> xcopy [\\host\c$\pp\pmgmt\lab1\Win8Debug\Package](file:///\\host\c$\pp\pmgmt\lab1\Win8Debug\Package) Package /efi

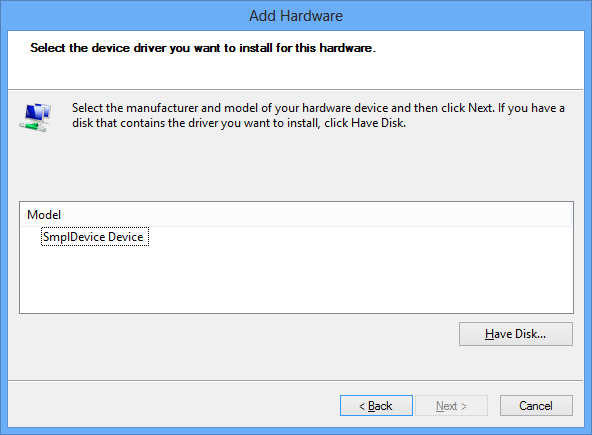
1. Open device manage: devmgmt.msc
2. Right click on the computer icon, and select ‘Add legacy hardware’
3. Click Next
4. Select ‘Install the hardware that I manually select from a list (Advanced)

## 

1. Click Next twice
2. Click ‘Have Disk’
3. Insert the path of the package



1. Select the ‘SmplDevice Device’ and click Next

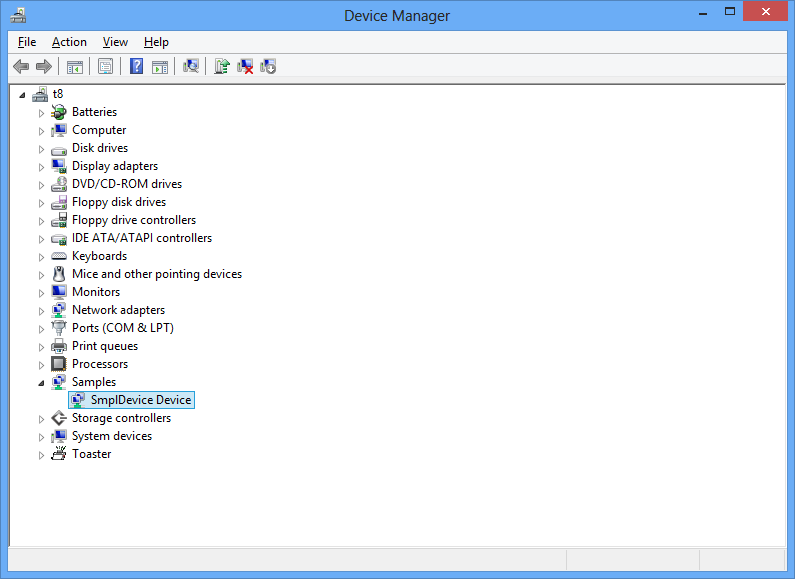


1. Click Next twice and Finish.

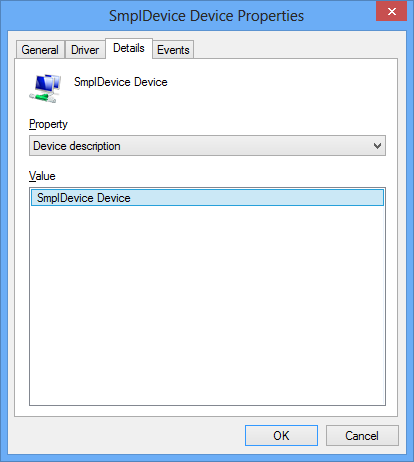
## Step 3: Inspecting, enabling and disabling driver operation using Device Manager:

The driver is already fully Plug and Play capable. It can be inspected, activated and deactivated using the Device Manager.

1. Open the device manager
2. Identify the SmpleDevice device under the Sample category



1. Right click, and select Properties
2. Switch to the details tab



1. Note the 'Device Class', 'Power Policy' and other settings.

# First inspection of source code and object model

This chapter makes the students familiar with the KMDF wizard generated code skeleton and the operating system driver and device object model.

## Step 1: Inspection of the wizard generated function driver skeleton

The wizard generated minimal function driver template needs to be inspected. It has no functionality yet. It is neither interacting with an application nor with any hardware.

For support of static code analysis, the following function prototypes are defined in the header file:

DRIVER\_INITIALIZE DriverEntry;

EVT\_WDF\_DRIVER\_DEVICE\_ADD SmplDeviceEvtDriverDeviceAdd;

The DriverEntry routine is the only function which is explicitly called by exported name.

During DriverEntry the KMDF driver has to create the WDFDRIVER object.

During the WDFDRIVER object's event handler EvtDriverDeviceAdd the driver has to create a WDFDEVICE object.

Hint: Always consult the WDK documentation (Help) and look up data structures and functions. These can then be copied and pasted directly from the WDK documentation to the driver source code.

## Step 2: Inspecting the driver object and the device object

The tool WinObj.exe (download from www.sysinternals.com) can be used to look at the driver object and the device object on the virtual Target machine.

# Setup of the WinDbg debugger environment

During this lab, the debugger will use a Local Area Network connection (only available in Windows 8). The environment of the debugger WinDbg will be set up on the Host machine. On the virtual Target machine kernel debugging has to be enabled using bcdedit to allow the Target to be controlled by the external debugger running on the Host machine.

## Step 1: Inspection of the debugger settings on the target

Run bcdedit from an elevated command line window on the Target. In this command line window please enter:

C:\> bcdedit /dbgsettings

If network debugging was already done by Visual Studio in the previous exercise, then nothing needs to be done any more.

## Step 2: Enable kernel mode debugging on the virtual Target machine

ONLY if Visual Studio provisioning of the test machine was NOT successful, the following commands need to be run to enable kernel debugging over network on the virtual Target machine.

Run bcdedit from an elevated command line window on the Target. In this command line Window please enter:

bcdedit /dbgsettings NET hostip:xxx.xxx.xxx.xxx port:yyyyy

bcdedit /debug on

If this is successful, then a key is printed on the console. Please copy paste this key into a file. It

will subsequently be needed by the debugger on the host machine to connect (see step below).

Target needs to be shut down and started again (caution: reboot might not be enough). After

that, the kernel debugger will be active and WinDbg can connect.

## Step 3: Setup of the debugger WinDbg on the Host machine

After starting WinDbg, the following settings have to be chosen:

* File menu → Kernel Debug → NET IP port number and key
* File menu → Symbol File Path (only path to own driver symbols, OS symbols NOT necessary)
* File menu → Source File Path

When compiling the driver, the debug symbols are compiled into a separate .pdb (Program Database) file.

Hint: For further info about WinDbg commands, consult the Appendix of this manual (e.g.: WinDbg useful commands).

## Step 4: Adjustment of “Debug Print Filter” registry setting

Open or create the following registry key on the virtual Target machine:

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Debug Print Filter

Add a registry value of type REG\_DWORD named IHVDRIVER. Set its value to 0xFFFFFFFF

## Step 5: Instrumantation of source code using DbgPrintEx(…)

Use the statement DbgPrintEx(DPFLTR\_IHVDRIVER\_ID, 1234,"…", …) to trace the call of DriverEntry and SmplDeviceEvtDriverDeviceAdd. The driver just has to be disabled and enabled in Device Manager while the debugger is connected. Then this debug output shows up on the debugger output screen.

On Windows XP DPFLTR\_DEFAULT\_ID can be used with DbgPrintEx.

Hint: WinDbg and the tool DbgView.exe (download from [www.sysinternals.com](http://www.sysinternals.com)) can both be used to visualize DbgPrintEx traces on the virtual Target machine.

## Step 6: Instantiation of breakpoints

The debugger needs the correct driver symbols loaded (Windows operating system symbols are not needed at this stage).

The option “Symbol File Path” has to point to the directory where the driver symbols are located. They reside in a separate .pdb file (program database). WinDbg loads symbols at driver startup. If connected later, symbols can be loaded using the command “.reload”.

When the driver is started and the symbols are loaded, the source file Driver.c can be opened and a breakpoint can be set in the function SmplDeviceEvtDriverContextCleanup using the F9 key. If successful, then the breakpoint is shown in red color.

Disable the device in Device Manager, then SmplDeviceEvtDriverContextCleanup will be called and the breakpoint will be triggered. Single stepping in the debugger can be done using the F10 key.