The WDK Build Environment

June 8, 2009

Abstract

This paper provides information about the Windows® Driver Kit (WDK) build environment. It describes how the tools work to provide developers a better understanding of what occurs when a project is built. With a better understanding of the build process, developers can use the tools more effectively.

This paper is helpful for any developer, beginner to advanced, who writes drivers or test applications for any category of device. This paper assumes that the reader has a general understanding of how to build a driver, but would like more information about what the tools are doing during the build process.

This information applies for the following operating systems:  
 Windows 7

Windows Server® 2008  
 Windows Vista®  
 Windows Server 2003  
 Windows XP

References and resources discussed here are listed at the end of this paper.

The current version of this paper is maintained on the Web at:   
 <http://www.microsoft.com/whdc/WDK_Build.mspx>

Disclaimer: This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2009 Microsoft Corporation. All rights reserved.

Microsoft, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Change |  |  |  |
| June 8, 2009 | First publication | | | |

Contents

[Introduction 4](#_Toc231982708)

[WDK Build Tools and Operation 4](#_Toc231982709)

[Setting Up the Build Environment 5](#_Toc231982710)

[How the Build Process Works 6](#_Toc231982711)

[Scanning the Project Files 6](#_Toc231982712)

[Pass 0: Producing Source Code 9](#_Toc231982713)

[Pass 1: Creating Object Files and Libraries 11](#_Toc231982714)

[Pass 2: Creating Final Binaries 12](#_Toc231982715)

[Optimizing the Order of Build Tasks 14](#_Toc231982716)

[Techniques for Building Drivers 14](#_Toc231982717)

[Building on a Multiprocessor Machine 14](#_Toc231982718)

[Using Custom Build Steps 17](#_Toc231982719)

[Code Optimization 18](#_Toc231982720)

[Viewing Additional Information about the Build Process 18](#_Toc231982721)

[Creating a Preprocessed Listing 18](#_Toc231982722)

[Viewing Generated Assembly Code 20](#_Toc231982723)

[Displaying the Build Environment Configuration 22](#_Toc231982724)

[Viewing Verbose Linker Information 22](#_Toc231982725)

[Storing Driver Package Files in a Specified Directory 23](#_Toc231982726)

[New WDK Build Environment Features 24](#_Toc231982727)

[Using the Default Separate Build Output Directories 24](#_Toc231982728)

[Using Custom Build Output Directories 25](#_Toc231982729)

[Using a Custom Object Root for Build Output 26](#_Toc231982730)

[Placing Binary Files in a Custom Location 26](#_Toc231982731)

[Placing Package Files Not Built during a Build 28](#_Toc231982732)

[Summary of Build Output Macros and Parameters 30](#_Toc231982733)

[Resources 31](#_Toc231982734)

[Appendix. Sample Build Log 32](#_Toc231982735)

# Introduction

This paper describes the build process for drivers and associated applications that developers create by using the Windows® Driver Kit (WDK). The interfaces for the build tools are documented in the WDK. This paper provides information about how the tools work to give you a better understanding of what occurs when you build your project. This knowledge can help you troubleshoot build problems and give you greater control over the build process. For example, this paper helps you understand how the build process is organized into passes and how to place driver output files in custom locations.

# WDK Build Tools and Operation

Table 1 provides a summary of the tools and files in the WDK build environment.

Table 1. Summary of Build Environment Components

|  |  |  |
| --- | --- | --- |
| Description | Component | Consumes |
| Build utilities | build.exe  nmake.exe | sources files, dirs files  makefile, makefile.def, makefile.inc |
| File that controls the build environment | SetEnv.bat |  |
| Files that identify what is to be built | dirs, sources, and makefile files |  |
| Compiler | cl.exe | .cpp files, .c files, .h files |
| Linker | link.exe | .obj files, .lib files |
| Supporting tools | midl.exe  rc.exe  binplace.exe  stampinf.exe  mofcomp.exe | .idl files  .rc files  .inx files  .mof files |
| Windows Auto Code Review (OACR) tool | oacrcl.exe, oacrlink.exe | oacr.ini, oacruser.ini |

For descriptions of each of the build tools and files, see the WDK.

Beginning with the Windows 7 WDK, you are not required to supply a *makefile* for your project. You can specify all dependencies and macros in your *sources* files, and the rest is done in the *makefile.def* file that is included with the WDK. You can still create your own build rules by using a *makefile.inc* file.

The OACR tool is enabled by default in the Windows 7 WDK. OACR is a set of tools that use static analysis to find potential defects in your driver source code. The OACR programs are wrappers around the C/C++ compiler and the linker. For more information on OACR, see “[Resources](#_Resources)” at the end of this paper.

The following sections review how to set up the build environment and then discuss what occurs during the build process.

## Setting Up the Build Environment

You open the build environment by using the following shortcut from the **Start** menu:

**Start** > **All Programs** > **Windows Driver Kits** > ***WDK\_Version*** > **Build Environments** > ***Operating System*** > ***Build Environment***

*WDK\_Version* is the version of the WDK that you are using. *Operating System* is the version of Windows that your driver will run on. *Build Environment* specifies the target processor architecture for which you are building your driver and whether you are building a checked (debug) or a free (release) version of your driver.

For example, consider the following shortcut:

**Start** > **All Programs** > **Windows Driver Kits** > **WDK 7001.0** > **Build Environments** > **Windows Win7** > **x86 Checked Build Environment**

The preceding shortcut opens a build environment window for using the Windows 7 Release Candidate (RC) WDK to build the debug version of a driver to run on an x86 Windows 7 system. This shortcut runs the following command:

C:\Windows\System32\cmd.exe /k C:\WinDDK\7100.0.0\bin\setenv.bat C:\WinDDK\7100.0.0\ chk x86 Win7

The examples in this paper show 7100.0 as the version number for the WDK. Your version might vary, depending on which version of the WDK you use.

Each build environment shortcut opens a command window and runs the *SetEnv.bat* file that sets a number of environment variables. The shortcut you choose sets parameters for *SetEnv.bat*. The four mandatory switches are:

* Root installation path for the WDK.
* Type of build: **chk** (checked) or **fre** (free).
* Processor architecture: **x86**, **amd64**, or ia**64** (Itanium 64).
* Operating system: **WinXP**, **WinNET** (Windows Server® 2003), **WinLH** (Windows Vista® or Windows Server 2008), and **Win7** (Windows 7).

The following command-line parameters for *SetEnv.bat* are new in Windows 7:

* **oacr** and **no\_oacr**

The default is **oacr**, which enables the Windows OACR tool. You can change this parameter by editing the shortcut for your build environment.

* **separate\_object\_root**

This parameter enables separate directories for target objects and binaries. You can change this parameter by editing the shortcut for your build environment. For more information, see “[Storing Driver Package Files in a Specified Directory](#_Storing_Driver_Package)” later in this paper. Also see the WDK.

The *SetEnv.bat* file runs automatically when you open a build environment window to build a driver. Usually you can use the default values that *SetEnv.bat* sets for the environment variables. However, in some situations you might want to set different values for these parameters. To customize a build environment shortcut, make a copy of the shortcut from the **Start** menu and modify the copy for any of the four switches mentioned earlier.

## How the Build Process Works

The Build utility—build.exe—performs its tasks in steps: a scanning phase that is followed by several other phases that are called passes (see Figure 1).



Figure 1. Steps in the build process

The Build utility completes each phase of the build before moving on to the next step.

The Build utility performs certain tasks in a given pass. For example, the Build utility typically compiles source files into object files during pass 1. For a small- or medium-sized project that resides in a single directory, the fact that the Build utility performs its steps in a set of distinct passes is insignificant. For more complicated projects, structuring the build tasks into separate passes enables the Build utility to better coordinate dependencies in the project when you use multiple threads.

Sometimes, especially when building a large project, the Build utility optimizes its actions and performs certain build tasks during a different pass than usual. For more information, see “[Optimizing the Order of Build Tasks](#_Optimizing_the_Order)” later in this paper.

You can write scripts that you run before the Build utility runs (prebuild) or after the Build utility exits (postbuild). For example, you might run a script after a build completes to sign your driver. You can also add custom build steps during the build passes by defining them in your project’s *sources* and *makefile.inc* files.

The following sections describe what occurs in each step of the build process.

### Scanning the Project Files

In its first step, the Build utility scans the project source tree and determines what targets must be built.

If a project is in a single directory, the Build utility gathers the information it needs from the *sources* file. The Build utility reads the *sources* file and records information from macros in the file such as TARGETNAME, TARGETTYPE, SOURCES, and INCLUDES.

If a project consists of files in a tree of directories, the Build utility scans the tree. When the Build utility finds a *dirs* file in a directory, it uses information in the DIRS and OPTIONAL\_DIRS macros in the file to identify the subdirectories to scan. In each subdirectory, the Build utility gathers information from the *sources* file as described earlier. A subdirectory in a project source tree that contains a *sources* file is called a leaf directory or a leaf node of the project.

The OPTIONAL\_DIRS macro describes directories that are not built all the time. For example, if a *dirs* file contains “DIRS=lib sauron driver” and ”OPTIONAL\_DIRS=app”, then the command line “build –cz” builds the projects *lib*, *sauron,* and *driver* and the command line “build –cz app” builds the projects *lib*, *sauron*, *driver,* and *app*. With OPTIONAL\_DIRS, you do not control the build order. You always build optional directories after you build required directories.

The Build utility scans the top-level directory and then recursively scans all subdirectories in *dirs* files. A project tree can be multiple levels deep. Each directory can have a *dirs* file or a *sources* file, but not both.

For each directory, the Build utility determines which passes are required to build each directory's targets and how to schedule the building of each directory. The scanning is a one-time step. The Build utility does not do any further scanning during subsequent build passes.

Figure 2 shows a sample set of directories for building the firefly project, which is a sample in the WDK.



Figure 2. Sample directories and files for a project

When you execute the build command in the root directory of the *firefly* project that is shown in Figure 2, the Build utility scans the files, beginning with that directory level. The *firefly* directory has a *dirs* file with a DIRS macro that specifies ”DIRS= lib sauron driver”. The *dirs* file indicates that subdirectories exist in the project. The Build utility scans the first subdirectory, scans the *lib* directory, and finds a *sources* file. The Build utility records the TARGETNAME and other information from the *sources* file, and then scans the rest of the subdirectories (*sauron* and *driver*). No other *dirs* files exist in the *firefly* project, so after the Build utility has scanned the *lib*, *sauron*, and *driver* directories, the scan is complete. The actual WDK *firefly* sample also has an *app* directory, but that directory is omitted in this example.

Note that when you first install the WDK, no *dirs* file exists in the firefly directory. The WDK samples ship this way to save space. The first time you run the Build utility in a samples directory, the WDK runs a makedirs utility that creates the main *dirs* file for the project.

When the scanning phase is complete, the Build utility has determined what passes are required to build the project targets and how the build activities will be scheduled. Figure 3 lists how the build might proceed for the *firefly* project in Figure 2.

Pass 0:

Run NMAKE for firefly\sauron project at pass 0 to run the tools:

Midl.exe->

Input: firefly\sauron\effects.idl

Output: effects.\*

Midl.exe->

Input: firefly\sauron\isauron.idl

Output: isauron.\*

Run NMAKE for firefly\driver project at pass 0 to run the tools:

mofcomp.exe->

Input: firefly\driver\firefly.mof, fireflymof.h

Output: firefly.bmf

---------------------------------------------------------------

Pass 1:

Run NMAKE for firefly\lib project at pass 1 to run the tools:

cl.exe->

Input : firefly\lib\luminous.cpp

Output :luminous.lib

Run NMAKE for firefly\sauron project at pass 1 to run the tools:

cl.exe->

Input: firefly\sauron\\*.c and \*.h

Output: sauron.lib and sauron.exp (the import library for sauron.dll)

cl.exe->

Input: firefly\sauron\stdafx.h

Output: stdafx.pch and pch\_hdr.src

Run NMAKE for firefly\driver project at pass 1 to run the tools:

cl.exe->

Input :firefly\driver\\*.c and \*.h

Output: device.obj, driver.obj, wmi.obj vfeature.obj

rc.exe->

Input: firefly\driver\firefly.rc

Output: firefly.res

Stampinf.exe->

Input: firefly\driver\firefly.inx

Output: firefly.inf

---------------------------------------------------------------

Pass 2:

Run NMAKE for firefly\driver project at pass 2 to run the tools:

Link.exe(oacrlink.exe)->

Input: firefly\driver\obj…\device.obj, driver.obj, wmi.obj vfeature.obj

Output:firefly.sys

Run NMAKE for firefly\sauron project at pass 2 to run the tools:

Link.exe(oacrlink.exe)->Input: firefly\sauron\obj…\sauron.lib and firefly\lib\obj\luminous.lib Output: sauron.dll

Rc.exe->

Input:firefly\sauron\saurondll.rc

Output:saurondll.res

Figure 3. Sample build task list for the firefly project

As shown in Figure 3, the build activities are organized by build pass. The Build utility determines what must be done during which passes, based on the types of the files in the project and on macros and other directives. The Build utility maintains the list of build tasks internally and does not write it to a file.

Based on the list of tasks, the Build utility runs a new instance of NMAKE for each project in the source tree for each pass. NMAKE calls the necessary tools to build the project, as shown in Figure 3. At the end of each pass, the Build utility waits for all instances of NMAKE to complete before it starts the next build pass. For more information on the NMAKE utility, see “[Resources](#_Resources).”

During the scan phase, the Build utility also creates an *\_objects.mac* text file for each leaf node in the project (or one \_*objects.mac* file if the project is in a single directory). The *\_objects.mac* file is a text file that contains the object dependencies for each platform, including the library path and the output path. For more information about the *\_objects.mac* file, see the WDK.

After the Build utility has a complete task list for the project to be built, it begins to build the project.

### Pass 0: Producing Source Code

In pass 0, the Build utility produces source code. The Build utility runs NMAKE in any directory that requires that source code be created. NMAKE might call one or more of the following tools during pass 0:

* Microsoft Interface Description Language (MIDL) compiler (Midl.exe).
* Other WDK support tools such as the Stamp INF tool (Stampinf.exe) and Managed Object Format (MOF) compiler (Mofcomp.exe).
* The BinPlace utility (Binplace.exe) to place files.

The Build utility does not usually call BinPlace during pass 0, but it will do so if, for example, you use the PASS0\_BINPLACE macro in a *sources* file. For more information about the BinPlace utility, see ”[Storing Driver Package Files in a Specified Directory](#_Storing_Driver_Package)” later in this paper.

For the sample driver in Figure 2, the Build utility runs NMAKE in the *sauron* directory during pass 0, passing the build pass number and certain other parameters to NMAKE. In turn, NMAKE calls Midl.exe to generate *.h*, *.c*, and *.tlb* (type library) files. Example 1 shows the information that is written to the build log for compiling *effects.idl* during pass 0.

The ellipses (”...”) in the log excerpt represent lines of the build log that are omitted from this example to save space and to focus on the driver compilation tasks. The lines starting with ”//” were added to the log excerpt for explanation; they do not appear in the actual build log file.

...

//Start building firefly\sauron on Proc2 (the running thread is designated by 2>)

2>Building generated files in c:\winddk\7100.0\src\hid\firefly\sauron \*\*\*\*\*\*\*\*\*\*\*\*\*

2>'nmake.exe /nologo BUILDMSG=Stop. -i BUILD\_PASS=PASS0 NOLINK=1 PASS0ONLY=1 MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\sauron'

2>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\sauron

//Start Midl.exe for firefly\sauron\effects.idl at Pass0 on Proc2

2> midl /Zp8 /IC:\WinDDK\7100.0\inc\atl30 /I..\shared /Ic:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\WDKSamples\inc /Ic:\winddk\7100.0\WDKSamples\inc\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\internal\WDKSamples\inc /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\crt /char unsigned /ms\_ext /c\_ext /proxy c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\effects\_p.c /dlldata c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\dlldata.c /iid c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\effects\_i.c /tlb c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\effects.tlb /header c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\effects.h /cpp\_cmd

/DNTDDI\_VERSION=0x06010000 /Di386 /D\_X86\_ /D\_WCHAR\_T\_DEFINED /no\_stamp /nologo -sal /win32 -target NT60 effects.idl

...

Example 1. Build log excerpt that shows midl compilation in pass 0

As shown in Example 1, the Build utility runs NMAKE and specifies that the BUILD\_PASS is PASS0 and that the directory is *sauron*. The Build utility determines the parameters to NMAKE from rules in the project *makefiles* and *sources* files.

The ”2>” at the beginning of some of lines in the example indicates a thread number for the NMAKE process. This build was executed on a multiprocessor machine, and the Build utility used multiple threads. All threads write to the same build log file, so in a full build log on a multiprocessor machine you see multiple interleaved tasks. You can use the thread numbers at the beginning of the line numbers to help identify the interleaved tasks.

NMAKE calls Midl.exe with the correct parameters and the files that will be generated. In this example, the files are *effects .c*, *.h* and *.tlb* and *isauron .c*, *.h,* and *.tlb*.

For a complete build log for building this sample, see the “[Appendix](#_Appendix_A_–).”

By default, the build log, (build\*.log) is placed in the working directory from which you run the build command.

When the Build utility completes all pass 0 tasks for a source tree, it moves on to pass 1.

### Pass 1: Creating Object Files and Libraries

In pass 1, the Build utility compiles source code into object files and libraries.

During pass 1, the Build utility runs NMAKE in any directory that requires object files or libraries to be created. If a *sources* file contains the PRECOMPILE\*=1 macro, then precompiled headers are built during pass 1. NMAKE might call one or more of the following tools during pass 1:

* The C/C++ compiler (Cl.exe).
* The Resource compiler (Rc.exe).
* Other WDK support tools such as the Stampinf.exe and Mofcomp.exe.
* The BinPlace utility (BinPlace.exe) to place files.

The Build utility calls BinPlace during pass 1 if you use the PASS1\_BINPLACE macro in a *sources* file. For more information about the BinPlace utility, see “[Storing Driver Package Files in a Specified Directory](#_Storing_Driver_Package) .”

The Build utility runs NMAKE, which runs the C/C++ compiler. For performance reasons, the Build utility internally splits pass 1 into two separate calls to NMAKE: pass 1a and pass 1b. All source compilation is done in pass 1a, and library creation is done in pass 1b. When the build is running in a single thread, it does not split pass 1 into two parts.

The compiler might be called through the OACR wrapper (Oacrcl.exe).

The resource compiler compiles .rc files into .res files, which are then linked into the final image as the resources.

Table 2 lists some common types of files that are built during pass 1.

Table 2. Common File Types Created During Pass 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TARGETTYPE value in *sources* file | Description | Files input to  pass 1 | Files created by pass 1 | Is target complete after pass 1? |
| LIBRARY | Static library | .h, .c, .cpp | .lib | Yes |
| DYNLINK | Dynamic link library or user-mode driver | .h, .c, .cpp | .lib  (import library) | No |
| DRIVER | Kernel-mode driver | .h, .c | .obj | No |

For example, for the sample project shown in Figure 2, the Build utility runs NMAKE in the *driver* subdirectory during pass 1. NMAKE calls the compiler to generate the object files. Example 2 shows the information that is written to the build log for compiling the driver files.

...

// Processing firefly\driver at pass 1 on Proc1

1>Compiling (NoSync) c:\winddk\7100.0\src\hid\firefly\driver \*\*\*\*\*\*\*\*\*\*\*\*\*

1>'nmake.exe /nologo BUILDMSG=Stop. -i BUILD\_PASS=PASS1 NOLINK=1 PASS1\_NOLIB=1 MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\driver'

1>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\driver

...

\\ Continue building firefly\driver at pass 1 on Proc1

...

1>Copyright (C) Microsoft Corporation. All rights reserved.

1> C:\WinDDK\7100.0\Bin\x86\oacr\oacrcl @c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\cl.rsp

1>Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.207 for 80x86

1>Copyright (C) Microsoft Corporation. All rights reserved.

1>cl /Fo"c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386/"

1> /FC

... \\ other switches deleted to save space

1> /wd4627

1> /typedil-

1> /FIC:\WinDDK\7100.0\inc\api\warning.h

1> .\driver.c .\device.c .\wmi.c .\vfeature.c

1>driver.c

1>device.c

1>wmi.c

1>vfeature.c

...

// Continue firefly\driver

1>Compiling c:\winddk\7100.0\src\hid\firefly\driver \*\*\*\*\*\*\*\*\*\*\*\*\*

...

Example 2. Build log excerpt that shows driver compilation in pass 1

As shown in Example 2, The Build utility runs NMAKE in thread 1 and specifies that BUILD\_PASS is PASS1 and the directory is *driver*. NMAKE calls the OACR wrapper for the compiler (Oacrcl.exe) that, in turn, calls the compiler (Cl.exe).

When the Build utility completes all pass 1 tasks for the project, it moves on to pass 2.

### Pass 2: Creating Final Binaries

In pass 2, the Build utility links objects and libraries into final binaries.

During pass 2, the Build utility runs NMAKE in any project that requires final binaries to be created. NMAKE might call one or more of the following tools during pass 2:

* Linker (Link.exe).
* Resource compiler (Rc.exe).
* BinPlace utility (BinPlace.exe), to place files.

The linker might be called through the OACR wrapper (Oacrlink.exe).

The build tools usually call the resource compiler in pass 1, but they might call it in pass 2, particularly for PROGRAM and DRIVER targets. Linking when the .res file is “hot” in the file system cache improves build times.

The Build utility calls BinPlace during pass 2 if PASS2\_BINPLACE or TARGET\_DESTINATION macros are defined in a *sources* file. For more information about the BinPlace utility, see “Storing Driver [Package](#_Storing_Driver_Package) Files in a Specified Directory” later in this paper.

Table 3 lists some common types of files that are built during pass 2.

Table 3. Common File Types Created during Pass 2

|  |  |  |  |
| --- | --- | --- | --- |
| TARGETTYPE value in *sources* file | Description | Files input to  pass 2 | Files created by pass 2 |
| DYNLINK | DLL or user-mode driver | .lib, obj | .dll |
| PROGRAM | User-mode application | .obj, .lib | .exe |
| DRIVER | Kernel-mode driver | .obj, lib | .sys |

The files that are input to pass 2 might not all be created during the same build and might reside in different directories.

For an example of pass 2 processing, consider the sample driver in Figure 2. During pass 2, the Build utility runs NMAKE in the *driver* directory. NMAKE calls the linker to generate the binaries. Example 3 shows the information in the build log for linking the driver binary.

As shown in Example 3, Build runs NMAKE on Processor 1 and specifies that the BUILD\_PASS is PASS2 and the directory is *driver*. NMAKE calls the OACR wrapper for the linker (oacrlink.exe) which, in turn, calls the linker to generate the driver binary. The /out linker parameter specifies the output file name for the generated binary. For more information about the location of build output files, see “[Storing Driver Package Files in a Specified Directory](#_Storing_Driver_Package).”

When the Build utility completes all of the pass 2 tasks for the project, the build is finished.

...

1>Linking for c:\winddk\7100.0\src\hid\firefly\driver \*\*\*\*\*\*\*\*\*\*\*\*\*

1>'nmake.exe /nologo BUILDMSG=Stop. -i BUILD\_PASS=PASS2 LINKONLY=1 NOPASS0=1 MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\driver'

1>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\driver

...

\\ Start Linking for firefly\driver at pass 2 on Proc1

1> C:\WinDDK\7100.0\Bin\x86\oacr\oacrlink /out:c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.sys /machine:ix86 @c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\lnk.rsp

...

\\ Continue linking of firefly\driver\all .obj at pass 2 on Proc1

1>Microsoft (R) Incremental Linker Version 9.00.30729.207

1>Copyright (C) Microsoft Corporation. All rights reserved.

1>/MERGE:\_PAGE=PAGE

1>/MERGE:\_TEXT=.text

1>/SECTION:INIT,d

... \\ other switches deleted to save space

1>/out:c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.sys

1>c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.res

1>c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\driver.obj

1>c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\device.obj

1>c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\wmi.obj

1>c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\vfeature.obj

1>C:\WinDDK\7100.0\lib\win7\i386\BufferOverflowK.lib

1>C:\WinDDK\7100.0\lib\win7\i386\ntoskrnl.lib

1>C:\WinDDK\7100.0\lib\win7\i386\hal.lib

1>C:\WinDDK\7100.0\lib\win7\i386\wmilib.lib

1>C:\WinDDK\7100.0\lib\wdf\kmdf\i386\1.9\WdfLdr.lib

1>C:\WinDDK\7100.0\lib\wdf\kmdf\i386\1.9\WdfDriverEntry.lib

1>C:\WinDDK\7100.0\lib\win7\i386\hidparse.lib

...

Example 3. Build log excerpt that shows driver linking in pass 2

## Optimizing the Order of Build Tasks

The previous sections describe a build process during which the build tools always perform certain tasks during specific passes. Sometimes, however, the Build utility optimizes the building of your project and runs certain tasks in a different pass than usual. For example, you might notice in your log file that the Build utility called NMAKE, which called the compiler to compile a file during pass 2 instead of pass 1. The Build utility might rearrange the order of build tasks for a project to improve build speed and to optimize resource usage. No matter how the Build utility rearranges the order of the build tasks, it always preserves dependencies.

For a complete example of a build log, see the “[Appendix](#_Appendix_A_–).” This build log includes the complete firefly project, including the flicker application. For the *firefly\app* directory, the Build utility optimizes its tasks and runs both the compiler and the linker in pass 2.

# Techniques for Building Drivers

The following sections provide additional information about certain aspects of the build process, such as building on a multiprocessor machine and using custom build steps.

## Building on a Multiprocessor Machine

For most projects, you can use the WDK build environment on a multiprocessor machine without making any changes to your files. However, if you have dependencies between directories within a single build pass, you must identify those dependencies in your *sources* file. Dependencies between directories that are not in a single build pass do not need any special attention, because the Build utility automatically manages dependencies across build passes.

#### How the Build Utility Works with Multiple Processors

The *SetEnv.bat* files that are included with the WDK set the BUILD\_MULTIPROCESSOR environment variable to 1. With this setting, the Build utility uses the same number of threads as available processors when processing each build pass. The /M*x* option on the build command overrides the BUILD\_MULTIPROCESSOR variable and instructs the Build utility to use *x* threads when processing each build pass. The Windows 7 WDK is the first WDK to set multiprocessing by default. Previous versions of the WDK defaulted to uniprocessor mode.

When you run the Build utility on a multiprocessor machine, the Build utility ensures that all the tasks from one build pass complete before it starts the next build pass. You are not required to specify any synchronization to ensure that, for example, a library that is produced in pass 1 is completed before being included in the creation of a driver *.sys* file during pass 2.

#### Specifying Dependencies in a Build Pass

The one situation where you must modify your *sources* files for building on a multiprocessor machine is when you must synchronize the building of components in different directories during the same build pass. For example, if you are building two static libraries where one library depends on the other, you must identify that dependency to the Build utility.

To specify such a dependency, use an *Xxx\_*PRODUCES macro in your *sources* file to identify a component that another component uses. Then use an *Xxx\_*CONSUMES macro to identify a dependency on another component that is built during the same build pass. An *Xxx*\_PRODUCES macro can have one or more matching *Xxx*\_CONSUMES macros. Table 4 lists the macros for identifying dependencies in each build pass.

Table 4. Xxx\_PRODUCES and Xxx\_CONSUMES Macros

|  |  |  |
| --- | --- | --- |
| Build Pass | *Xxx*\_PRODUCES macro | *Xxx*\_CONSUMES macro |
| Pass 0 | BUILD\_PASS0\_PRODUCES | BUILD\_PASS0\_CONSUMES |
| Pass 1 | BUILD\_PRODUCES | BUILD\_CONSUMES |
| Pass 2 | BUILD\_PASS2\_PRODUCES | BUILD\_PASS2\_CONSUMES |

The Build utility matches producers with consumers by matching the string in the macros. For example, consider the sample project in Figure 4.



Figure 4. Sample project with a build dependency

Example 4 shows excerpts from the *sources* files for the sample project in Figure 4.

;sources file for Geometry library

...

TARGETNAME = Geometry

TARGETTYPE = LIBRARY

BUILD\_PRODUCES = geometry\_lib

SOURCES = \

Cos.c \

Sin.c

...

;sources file for Math library

...

TARGETNAME = Math

TARGETTYPE = LIBRARY

BUILD\_CONSUMES = geometry\_lib

SOURCES = Math.c

OBJLIBFILES = $(OBJ\_PATH)\..\Geometry\$(O)\geometry.lib

...

;sources file for Plot project

...

TARGETNAME = Plot

TARGETTYPE = DRIVER

SOURCES = Plot.c

TARGETLIBS = $(OBJ\_PATH)\..\Geometry\$(O)\Math.lib

...

Example 4. Sources files with a build dependency

In the preceding example, the Math library that is built during pass 1 depends on the Geometry library that is also built during pass 1. The BUILD\_PRODUCES and BUILD\_CONSUMES macros identify this dependency to the Build utility.

When dependencies exist within a build pass, the order of the directories in your *dirs* files is important. You must list the directories for prerequisite components before the directories for components that consume them. In the preceding example, you must list the Geometry directory in the *dirs* file before the Math directory. The Build utility searches *dirs* files depth-first to find all directories with *sources* files. If your project source tree is multiple layers deep, make sure that a depth-first search will find any prerequisite components before components that consume them.

If the Build utility encounters an *Xxx*\_CONSUMES macro before it encounters a matching *Xxx*\_PRODUCES macro, it ignores the *Xxx*\_CONSUMES macro. If a component is missing dependent components at link time, the linker generates an error.

Listing dependent components in the correct order in your *dirs* files does not eliminate the need to explicitly specify the dependencies within a build pass. You must still identify dependencies in your *sources* files using *Xxx*\_PRODUCES and *Xxx*\_CONSUMES macros so that the Build utility can optimize the processing of dependent components on a multiprocessor machine.

## Using Custom Build Steps

You can run a custom build step during any build pass by using an NTTARGETFILEx macro in your *sources* file and providing a *makefile.inc* file. The macro identifies that a custom build step is required and the *makefile.inc* file provides the information for performing the custom build step.

The NTTARGETFILE0, NTTARGETFILE1, and NTTARGETFILE2 macros identify custom build steps for each of the three build passes. For more information on how to use these macros, see the WDK.

We recommend that you use one of the preceding macros and do not use the NTTARGETFILES macro. The NTTARGETFILES macro does not specify the build pass during which the custom build step should be performed, so the Build utility tries to build the custom target during every pass. In this situation, you must add conditional code to your *sources* file if the custom step is not relevant to all build passes.

You can use a *makefile.inc* file to define steps to build header files that subdirectories require, to generate additional files such as text files, to copy or delete files, or to perform any other driver-specific build steps. Figure 5 illustrates how the build components work together.



Figure 5: Initiating a custom build step

As shown in Figure 5, the Build utility runs NMAKE and passes the path to the project *makefile*. The project *makefile* includes the system-supplied *makefile.def* file. If the project does not supply a makefile, which is optional beginning with the Windows 7 WDK, then the Build utility passes NMAKE the path to *makefile.def*. The *makefile.def* file includes *makefile.new*, which includes the project *sources* file. In this example, the project *sources* file contains an NTTARGETFILEx macro, which directs *makefile.def* (through *makefile.new*) to look for *makefile.inc* in the project directory. The *makefile.inc* file contains the information for the custom build step.

## Code Optimization

The WDK provides build environments for producing checked (debug) builds to use during development and free (release) builds to generate final versions of your code. By default, the WDK build environment enables code optimization in a free build and disables code optimizations in a checked build.

The WDK uses a macro to disable code optimizations in a checked build. To disable code optimizations in a free build, add the following macro to your *sources* file:

MSC\_OPTIMIZATION=/Od /Oi

You seldom disable code optimizations in a free build. However, this macro is useful if you must disable code optimizations in just one part of a large project.

You can use the following platform-specific macros in your *sources* file to override the WDK build environment setting for MSC\_OPTIMIZATION:

* 386\_OPTIMIZATION
* IA64\_OPTIMIZATION
* AMD64\_OPTIMIZATION

For example, you could specify 386\_OPTIMIZATION=/Od in your *sources* file to disable code optimization on 386 platforms.

If you want to do source-level debugging, use a checked build that disables code optimization. Debugging optimized code can be extremely difficult.

# Viewing Additional Information about the Build Process

This section describes techniques for gathering additional information about how the WDK build tools build your project.

## Creating a Preprocessed Listing

A preprocessed listing of a source file can help you debug problems such as errors in macros and duplicate structure definitions. You can create a preprocessed listing by specifying the *.pp* file extension to the NMAKE command. In the resulting output, you can see the structures, where they are defined, and whether NMAKE retrieved headers from the correct location. To create a preprocessed listing, run the following command in a WDK build environment window:

nmake source-file-name.pp

The NMAKE utility calls the C/C++ preprocessor and sets a rule that directs the preprocessor to create a preprocessed listing. For example, if you have a source file that is named *toaster.c*, you can create a preprocessed listing of that source file by running the following command:

nmake toaster.pp

Preprocessing adds line numbers to the preprocessed listing file. You can use these line numbers to determine whether NMAKE retrieved headers from the correct location and which #define directives and structures it added.

Preprocessing also adds #line directives to the output file at the beginning and end of each included file and around lines that were removed by preprocessor directives that specify conditional compilation.

The preprocessed listing file is identical to the original source file, except that the compiler carries out all preprocessor directives and performs macro expansions. The #line directives are the line numbers relative to the file that they are in. Each #include is like a copy-and-paste of the included file into the main source file at the line where the #include directive is located.

Example 5 shows an excerpt from the preprocessor listing created by the ”nmake toaster.pp” command that was run on the *toaster* sample driver in the WDK. During preprocessing, the compiler includes a portion of the *wdftypes.h* header file, starting at line 61. Blanks lines in a *.pp* file represent blank lines or comments in the original source file.

...

#line 61 "c:\\winddk\\7100.0\\inc\\wdf\\kmdf\\1.9\\wdftypes.h"

typedef enum \_WDF\_TRI\_STATE {

    WdfFalse = 0,

    WdfTrue = 1,

    WdfUseDefault = 2,

} WDF\_TRI\_STATE, \*PWDF\_TRI\_STATE;

typedef PVOID WDFCONTEXT;

typedef struct WDFDEVICE\_INIT \*PWDFDEVICE\_INIT;

typedef struct \_WDF\_OBJECT\_ATTRIBUTES \*PWDF\_OBJECT\_ATTRIBUTES;

...

Example 5. Sample preprocessor listing (excerpt from toaster.pp)

Example 6 shows an excerpt from the *wdftypes.h* header file, beginning at line 61.

...

//

// Do not create an invalid value for this enum in case driver writers // mix up the usage of WdfFalse/FALSE and WdfTrue/TRUE.

//

typedef enum \_WDF\_TRI\_STATE {

    WdfFalse = FALSE,

    WdfTrue = TRUE,

    WdfUseDefault = 2,

} WDF\_TRI\_STATE, \*PWDF\_TRI\_STATE;

typedef PVOID WDFCONTEXT;

//

// Forward declare structures needed later header files

//

typedef struct WDFDEVICE\_INIT \*PWDFDEVICE\_INIT;

typedef struct \_WDF\_OBJECT\_ATTRIBUTES \*PWDF\_OBJECT\_ATTRIBUTES;

...

Example 6. Excerpt from the wdftypes.h header file

Note that the blank lines and comment lines in the *wdftypes.h* header file in Example 6 result in blank lines in the *toaster.pp* file in Example 5.

## Viewing Generated Assembly Code

You can create a listing of the assembly code that the compiler generates from your source code. This listing contains both the source code and the generated assembly code, which enables you to see line numbers, functions, and assembly code that can be helpful during debugging. To create an assembly code listing, issue the following command in the WDK build environment window:

nmake source-file-name.cod

Another way to generate a .cod file is to add the “/Fc” switch to your compile command line.

For example, if you have a source file that is named *queue.cpp* in the UMDF toaster project, you can generate an assembly listing of that file by running the following command:

nmake queue.cod

The NMAKE utility calls the compiler, and the compiler generates the assembly code listing.

Example 7 shows an excerpt from the *queue.cod* file that shows the source code and the generated assembly code.

...

?OnRead@CQueue@@UAGXPAUIWDFIoQueue@@PAUIWDFIoRequest@@K@Z PROC ; CQueue::OnRead, COMDAT

; 222 : {

00000 8b ff npad 2

00002 55 push ebp

00003 8b ec mov ebp, esp

\_\_annotation$73771:

; 223 : Trace(TRACE\_LEVEL\_INFORMATION,"%!FUNC!");

00005 a1 00 00 00 00 mov eax, DWORD PTR \_WPP\_GLOBAL\_Control

0000a 3d 00 00 00 00 cmp eax, OFFSET \_WPP\_GLOBAL\_Control

0000f 74 1e je SHORT $LN3@OnRead

00011 f6 40 1c 01 test BYTE PTR [eax+28], 1

00015 74 18 je SHORT $LN3@OnRead

00017 80 78 19 04 cmp BYTE PTR [eax+25], 4

0001b 72 12 jb SHORT $LN3@OnRead

0001d 68 00 00 00 00 push OFFSET \_WPP\_Queue\_cpp\_Traceguids

00022 6a 0c push 12 ; 0000000cH

00024 ff 70 14 push DWORD PTR [eax+20]

00027 ff 70 10 push DWORD PTR [eax+16]

0002a e8 00 00 00 00 call \_WPP\_SF\_@16

$LN3@OnRead:

; 224 :

; 225 : //

; 226 : // No need to check for zero-length reads.

; 227 : //

; 228 : // The framework queue is created with the flag bAllowZeroLengthRequests = FALSE.

; 229 : // FALSE indicates that the framework completes zero-length I/O requests instead

; 230 : // of putting them in the I/O queue.

; 231 : //

; 232 :

; 233 : //

; 234 : // TODO: Put your Read request processing here

; 235 : //

; 236 :

; 237 : pRequest->CompleteWithInformation(S\_OK, SizeInBytes);

0002f ff 75 14 push DWORD PTR \_SizeInBytes$[ebp]

00032 8b 45 10 mov eax, DWORD PTR \_pRequest$[ebp]

00035 8b 08 mov ecx, DWORD PTR [eax]

00037 6a 00 push 0

00039 50 push eax

0003a ff 51 20 call DWORD PTR [ecx+32]

; 238 :

; 239 : return;

; 240 :

; 241 : }

0003d 5d pop ebp

0003e c2 10 00 ret 16 ; 00000010H

?OnRead@CQueue@@UAGXPAUIWDFIoQueue@@PAUIWDFIoRequest@@K@Z ENDP ; CQueue::OnRead

Example 7. Sample .cod file that shows generated assembly for queue.cpp

In an assembly listing file, the source code lines are preceded by ”; *line-number* :”, such as ”; 239 : return” in Example 7. A line of generated assembly code contains the byte offset (such as 0003e), the OpCode or machine code for the instruction, and the mneumonic for the assembly instructions (mov, push, call, and so on). For a C++ source file, the *.cod* file also shows the “mangled” function names such as the one in Example 7 for the OnRead callback function.

## Displaying the Build Environment Configuration

You can obtain information about the current configuration of your build environment by running nmake with the /P option. This command displays the values of environment variables and macros in *SetEnv.bat*, values of macros in your *sources* file, and the values of inference rules such as the number of processors that are used during the build. When you specify the /P option, NMAKE outputs information to standard output but does not build the project.

To capture the output in a file, run the following command in your WDK build environment window, with your context set to the directory where your project *makefile* resides:

nmake /P >build-config-output-file

The NMAKE utility writes a large amount of information to standard output. We recommend that you specify an output file name to write the listing to a file.

Example 8 shows an excerpt of the output that is generated by running nmake /P on the *toaster* sample driver in the WDK.

...

MSC\_WARNING\_LEVEL = $(MSC\_WARNING\_LEVEL) $(COMPILER\_WX\_SWITCH)

                                /W4

NUMBER\_OF\_PROCESSORS = 2

\_NT\_TARGET\_VERSION\_WIN7 = 0x601

                                0x601

// The following line shows where build output will go

 OBJ\_PATH = c:\winddk\7100.0\src\general\toaster\kmdf\func\featured

                                $(MAKEDIR)

// This is the directory for binplaced binaries

TARGET\_DESTINATION = wdf

USE\_OBJECT\_ROOT = 1

 ATL\_INC\_PATH = C:\WinDDK\7100.0\inc

 SDK\_LIB\_DEST = C:\WinDDK\7100.0\lib\win7

 SDK\_INC\_PATH = C:\WinDDK\7100.0\inc\api

...

Example 8. Sample build configuration listing

Example 8 shows that the *toaster* project is built by using warning level W4, on two processors, and generates a target that runs on Windows 7. The OBJ\_PATH specifies where build stores the output files, and the TARGET\_DESTINATION indicates that the BinPlace feature is used to place the binary files in a directory that is named *wdf*.

## Viewing Verbose Linker Information

You might have questions such as the following when you build a target:

* Why did the linker pull in function *Xxx* from that object?
* Why did the linker use function *Yyy* from library B instead of from library A?
* Why was function *Zzz* not included in the final image?
* What libraries are searched to produce this image?
* What switches and objects are in the image?

You can use the VERBOSE linker option to obtain information to answer these questions. To set the verbose flag, run the following command in the WDK build environment window before you run the build:

set LINKER\_FLAGS=$(LINKER\_FLAGS) /VERBOSE

Example 9 shows an excerpt from a build log for the *toaster* sample in the WDK when it was built with the verbose linker output enabled.

...

Searching C:\WinDDK\7100.0\lib\win7\i386\ntoskrnl.lib:

1> Found \_memset

1> Referenced in toaster.obj

1> Referenced in wmi.obj

1> Loaded ntoskrnl.lib(ntoskrnl.exe)

1> Found \_memcpy

1> Referenced in wmi.obj

1> Loaded ntoskrnl.lib(ntoskrnl.exe)

1> Found \_\_imp\_\_ExFreePoolWithTag@8

1> Referenced in wmi.obj

1> Loaded ntoskrnl.lib(ntoskrnl.exe)

1> Found \_\_imp\_\_IoWMIWriteEvent@4

1> Referenced in wmi.obj

1> Loaded ntoskrnl.lib(ntoskrnl.exe)

1> Found \_\_imp\_\_KeQuerySystemTime@4

1> Referenced in wmi.obj

1> Loaded ntoskrnl.lib(ntoskrnl.exe)

1> Found \_\_imp\_\_ExAllocatePoolWithTag@12

1> Referenced in wmi.obj

1> Loaded ntoskrnl.lib(ntoskrnl.exe)

...

Example 9. Sample verbose linker output in the build log

The verbose linker output in Example 9 shows that the linker found the memset and memcopy functions in the *ntoskrnl.lib* library.

# Storing Driver Package Files in a Specified Directory

When you build a project, the build tools place the output files in a subdirectory of your project directory unless you specify a different location. By default, the build tools create two levels of subdirectories that have the following name:

obj[fre|chk]\_*OS*\_*build-architecture*\*processor-type*

The build tools use $(O) to refer to this two-level subdirectory.

For example, you might run a checked build of the WDK KMDF *toaster* sample in the *src\general\toaster\kmdf\func\featured* directory. If you build the driver for Windows 7 on an x86 machine, the build tools place the build output in the following directory under the base directory of the WDK installation:

src\general\toaster\kmdf\func\featured\objchk\_win7\_x86\i386

The build tools refer to this directory by using the following variable:

$(OBJ\_PATH)\$(O)

The OBJ\_PATH is the path to the directory where your *sources* file and *makefile* reside (also called the current working directory). In the preceding example, the OBJ\_PATH is src\general\toaster\kmdf\func\featured, and O is objchk\_win7\_x86\i386.

The new **separate\_object\_root** build parameter in Windows 7 enables you to specify an alternate location for the build output files. By using **separate\_object\_root**, you can keep your source directory tree clean and control where your build output files are placed.

The following sections describe how the build tools place the build output files and how you can control the placement of these files.

## New WDK Build Environment Features

The WDK build environment in the Windows 7 WDK includes two new optional features:

* Separate object root.

All build output files, including object files and binary files, can be placed in a separate directory tree.

* Separate binary root.

All binaries and associated package files can be placed in a separate directory tree.

These two features enable your source tree to be immutable and possibly placed on an external, read-only disk. These features also make it easier to clean up the output from a particular build.

The separate object root feature specifies a location outside the source tree for placing all build output. For more information on using a separate object root, see “[Using the Default Separate Build Output Directories](#_Using_the_Default)” and “[Using Custom Build Output Directories](#_Using_Custom_Build)” later in this paper.

The separate binary root specifies a uniform target destination to place binaries in a common directory. To place files in a separate binary root, you can use BinPlace.exe either from a command line or by using macros in the *sources* file (TARGET\_DESTINATION and possibly PASS*n*\_BINPLACE). For more information on using a separate binary root, see “[Placing Binary Files in a Custom Location](#_Placing_Binary_Files)” and “[Placing Package Files Not Built during a Build](#_Placing_Package_Files)” later in this paper.

## Using the Default Separate Build Output Directories

When you use the Windows 7 WDK to build projects, you can direct the build tools to place the output files in a separate location by setting the new **separate\_object\_root** parameter. Setting this parameter enables a group of macros that control the placement of output files. To set this parameter, edit your WDK build environment shortcut in the Start menu and add **separate\_object\_root** to the *SetEnv.bat* command line. To display the value of the **separate\_object\_root** parameter, run the set command in your WDK build environment window with no parameters. If the **separate\_object\_root** parameter is enabled, the set command displays ”separate\_object\_root=TRUE”.

If **separate\_object\_root** is specified in your WDK build environment shortcut, *SetEnv.bat* defines two related macros, OBJECT\_ROOT and \_NTTREE, when you open a build window:

OBJECT\_ROOT = %BASEDIR%.obj  
\_NTTREE= \ %BASEDIR%.binaries\%DDK\_TARGET\_OS%\%BUILD\_ARCH%%DDKBUILDENV%

The OBJECT\_ROOT macro defines the root of a directory tree for storing all output that the build generates, including object files, binaries, INF files, catalog files and other output files. The \_NTTREE macro defines the root of a directory tree for storing final files to be packaged. The BinPlace utility copies the binary files from OBJECT\_ROOT to \_NTTREE.

By default, both directory trees are located under the current WDK installation directory, which is represented by BASEDIR. The other variables in the \_NTTREE definition specify the operating system and the target machine architecture on which the resulting binary files run and whether it is a free or checked build.

For example, if you enable **separate\_object\_root** and then open a WDK build environment window to run a free build of a project for Windows 7 on an x86 machine, *SetEnv.bat* defines these macros as follows:

OBJECT\_ROOT = %BASEDIR%.obj  
\_NTTREE = %BASEDIR%.binaries\Win7\x86fre

The build tools use OBJECT\_ROOT to define the path to the object files, as follows:

$(OBJECT\_ROOT)\$(OBJ\_PATH)\$(O)

The project package files are placed under \_NTTREE .

For example, if you run a free build of the KMDF toaster driver sample in the WDK (located in src\general\toaster\kmdf\func\featured) for Windows 7 that is running on AMD64 machines with **separate\_object\_root** enabled, the build tools place the build output files in the following directories:

All build output files:  
C:\WinDDK\7100.0.obj\src\general\toaster\kmdf\func\featured\objfre\_win7\_amd64\amd64

Project package files:  
C:\WinDDK\7100.0.binaries\Win7\amd64fre

Using the default separate object root locations when you build WDK sample drivers keeps your build output files out of the WDK sample source tree and places them in convenient locations for testing and investigation.

## Using Custom Build Output Directories

The default separate object root directories are appropriate for projects where the source files are in the WDK tree. However, you should specify your own separate object root directory for projects that are not part of the WDK.

### Using a Custom Object Root for Build Output

In addition to setting **separate\_object\_root**, with the Windows 7 WDK you can also specify your own location for your build output files. You redefine the OBJECT\_ROOT macro by issuing a command such as the following in the WDK build environment window:

set OBJECT\_ROOT = c:\myobjectroot

The build tools now place your build output files in subdirectories under the new OBJECT\_ROOT directory. This includes object files, intermediate files, binary files, and other files that the build tools generate. The complete name of the build output directory is as follows:

$(OBJECT\_ROOT)\$(OBJ\_PATH)\$(O)

For example, for an AMD64 checked build of a project in *d:\src\myproject*, with OBJECT\_ROOT set to *c:\myobjectroot*, the build tools place your build output files in the following directory:

c:\myobjectroot\src\myproject\objchk\_win7\_amd64\amd64

In this example, OBJECT\_ROOT is c:\myobjectroot, OBJ\_PATH is \src\myproject, and O is objchk\_win7\_amd64\amd64. Note that the output files are placed on the c: drive, as specified by OBJECT\_ROOT, not on the d: drive with the source files.

When you define OBJECT\_ROOT, use all lowercase letters. If you use mixed case, the build tools convert the text to lowercase.

By enabling **separate\_object\_root** and defining your own OBJECT\_ROOT, you can keep your build output files out of your source tree and in an easily-located directory.

### Placing Binary Files in a Custom Location

In addition to using the OBJECT\_ROOT macro to specify a custom location for all your build output files, you can also use the \_NTTREE macro to specify a custom location for your project package files that is separate from the rest of the build output. By using \_NTTREE together with the TARGET\_DESTINATION macro, you can place your project package files under a separate binary root and avoid having to manually navigate and pick up the files for your driver package from various directories. The build tools call the BinPlace utility to copy files from the build output files directory to the specified binary location.

The BinPlace utility has been included in the WDK for a long time, but only with a command-line interface. However, with the build tools in the Windows 7 WDK, you can have the build tools run the BinPlace utility through macros in your *sources* file. The rest of this section focuses on using macros to cause the build tools to run the BinPlace utility. For information on using BinPlace from the command line, see the WDK.

A common reason for why you might be required to use BinPlace is that you must have multiple files in a package to run tests. You can direct the BinPlace utility to conveniently place all these files in one directory. For example, for a device driver package you might be required to place the driver file, the INF file, the co-installer, and the catalog file in a single directory. The benefits of using BinPlace include the following:

* BinPlace creates a *binplace.log* file with the source and destination of each file that it copies.
* You can use BinPlace to copy any number of files into the desired directories.
* BinPlace can be run automatically by using macros in a *sources* file.
* You are not required to manually run a BinPlace command each time that you build your project.

To run BinPlace through macros in your *sources* file, follow these steps:

1. Set the **separate\_object\_root** parameter in your build shortcut, as described in “[Using the Default Separate Build Output Directories](#_Using_the_Default)” earlier in this paper.

This enables you to have a separate directory tree for project package files in addition to a separate directory tree for the rest of your build output files.

2. Define \_NTTREE and OBJECT\_ROOT in your WDK build environment window.

Setting a custom value for \_NTTREE specifies a separate directory tree for your project package files. To redefine the \_NTTREE macro, issue a command such as the following in the WDK build environment window:

set \_NTTREE = c:\mybinaryroot

3. Specify a TARGET\_DESTINATION macro in your *sources* file.

Include the TARGET\_DESTINATION macro in your *sources* file to specify a subdirectory under the \_NTTREE root. If the TARGET\_DESTINATION macro is defined, the build tools copy TARGETNAME files from the build output location to the following directory:

$(\_NTTREE)\$(TARGET\_DESTINATION)

For example, if the \_NTTREE macro is set to c:\mybinaryroot and the TARGET\_DESTINATION macro is set to mybin in the *sources* file, the build tools place the TARGETNAME binary files in the following directory:

c:\mybinaryroot\mybin

By using \_NTTREE together with TARGET\_DESTINATION, you can direct the BinPlace utility to place your project package files in a separate directory. The build tools call the BinPlace utility to copy files from the build output directories to the specified directories.

Depending on how many different TARGET\_DESTINATION values you define in all your projects, you can place your project package files in one subdirectory under \_NTTREE or in a hierarchy of subdirectories. You can use \_NTTREE and TARGET\_DESTINATION to create a flat directory structure for your build packages or to create a more complex directory structure, depending on your specific needs. You can define one TARGET\_DESTINATION for each *sources* file.

The Build utility ignores the TARGET\_DESTINATION macro if **separate\_object\_root** is not enabled. You can instrument your *sources* files for binplacing files, and then choose whether to use that feature when you run a particular build of the project based on whether you enable **separate\_object\_root**.

The BinPlace utility creates a log file that is named *binplace.log* and places it in a directory that the BINPLACE\_LOG environment variable specifies. *SetEnv.bat* sets BINPLACE\_LOG to %\_NTTREE%\build\_logs when you open a build command window. If you customize \_NTTREE, you should also customize the BINPLACE\_LOG environment variable to specify a directory for the BinPlace log file.

Using \_NTTREE with TARGET\_DESTINATION places only TARGETNAME binaries. For information on binplacing other files, see the following section, “[Placing Package Files Not Built during a Build](#_Placing_Package_Files).”

Using the BinPlace utility through macros places files in the desired directories, but it does not strip symbol files or split files. Use the BinPlace utility from the command line if you must strip or split files.

### Placing Package Files Not Built during a Build

Some of the files that you must place in a project package, such as INF files and catalog files, are not TARGETNAME binaries. To place files that are not TARGETNAME binaries, enable **separate\_object\_root** in the WDK build environment shortcut and use PASS*n*\_BINPLACE macros in your *sources* file. The value of *n* can be 0, 1, or 2 and specifies the build pass during which the files should be placed.

If you enable **separate\_object\_root** and you include a PASS*n*\_BINPLACE macro in your *sources* file, the Build utility runs the BinPlace utility at the end of pass *n* to copy the specified files to the desired location. You can specify PASS*n*\_BINPLACE macros for one or more passes. The BinPlace utility copies the specified files from the OBJECT\_ROOT directories and places them in the appropriate directory under the \_NTTREE location.

You must specify the destination location for files that are placed by using PASS*n*\_BINPLACE macros. You can either specify a general TARGET\_DESTINATION in your *sources* file or specify an explicit destination in the PASS*n*\_BINPLACE macro by using the -:DEST option.

The BinPlace utility uses the following rules to determine the file destination:

* By default, BinPlace uses the destination that is specified by a TARGET\_DESTINATION macro in your *sources* file.
* A PASS*n*\_BINPLACE macro can specify a -:DEST option to override the TARGET\_DESTINATION.
* If the *sources* file does not contain a TARGET\_DESTINATION, the PASS*n*\_BINPLACE macro must explicitly specify a destination by using the -:DEST option.
* Built-in destination values, such as ”retail”, can be used with the ‑:DEST option and with TARGET\_DESTINATION. For a list of these values, see the WDK.
* If you override TARGET\_DESTINATION by specifying the -:DEST option in a PASS*n*\_BINPLACE macro, the new destination value stays in force for the duration of that macro unless it is changed with another -:DEST option. For example, you can specify ”-:DEST dest1 file1 file2” to place two files to a specific destination or you can specify ”-:DEST dest1 file1 -:DEST dest2 file2” to place two files to separate destinations.

Example 10 shows a *sources* file that places the INF file for the WDK KMDF *toaster* sample driver.

TARGETNAME=wdffeatured

TARGETTYPE=DRIVER

KMDF\_VERSION\_MAJOR=1

INF\_NAME=wdffeatured

INCLUDES = $(INCLUDES);..\..\inc;..\shared

NTTARGETFILE1=$(OBJ\_PATH)\$(O)\$(INF\_NAME).inf

PASS1\_BINPLACE=$(NTTARGETFILE1)

NTTARGETFILE0 = $(OBJ\_PATH)\$(O)\toaster.bmf

#

# List of source files to compile.

#

SOURCES= \

toaster.rc \

toaster.c \

power.c \

wmi.c

C\_DEFINES=

TARGET\_DESTINATION=wdf

Example 10. Sample *sources* file that uses PASS1\_BINPLACE

In Example 10, NTTARGETFILE1 generates a custom target, which is the INF file. The *sources* file in Example 10 has three targets:

* The *wdffeatured.sys* driver, which is identified by the TARGETNAME macro.
* The *wdffeatured.inf* file (a custom target).
* The *toaster.bmf* file (a custom target).

The example BinPlaces two of the targets:

* *wdffeatured.sys*

Because a TARGET\_DESTINATION is specified, the Build utility places any TARGETNAME targets to that destination.

* *wdffeatured.inf*

The PASS1\_BINPLACE macro causes the Build utility to call the BinPlace utility at the end of pass 1 for the INF file. Th*e* binplace macro does not specify an explicit destination, so the INF is placed to the TARGET\_DESTINATION.

This sample does not place the *toaster.bmf* target because it is used to generate another file but it is not included on its own in the driver package.

For example, if you run an x86 free build of the WDK KMDF toaster sample for Windows 7 with **separate\_object\_root** enabled, the build tools place the *wdffeatured.sys* and the *wdffeatured.inf* files in the following location:

$(\_NTTREE)\$(TARGET\_DESTINATION)

By using the default value for \_NTTREE and the TARGET\_DESTINATION=wdf as defined in the *sources* file, this location would be the following:

C:\WinDDK\7100.0.binaries\Win7\x86fre\wdf

For an example of how to use the TARGET\_DESTINATION and PASS*n*\_BINPLACE macros and for further information, see the WDK sample files in \src\general\build. For descriptions of the macros and utilities, see the WDK.

## Summary of Build Output Macros and Parameters

Table 5 provides a summary of the parameters, macros, and environment variables that are relevant to the placement of build output files.

Table 5. Parameters and Macros for Storing Driver Package Files

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| separate\_object\_root | build parameter | Enables a group of macros (OBJECT\_ROOT, \_NTTREE, and PASS*n*\_BINPLACE) for storing output files in separate locations.  Set by adding it to the build shortcut. |
| BASEDIR | environment variable | In the WDK, sets the root directory of the WDK installation. |
| OBJECT\_ROOT | macro | Defines the root of a directory tree for storing all output that a build generates.  Only used if separate\_object\_root is enabled.  Set in the build command window. |
| OBJ\_PATH | macro | Defines the path to the directory where the project *sources* file and *makefile* reside or to the current working directory. |
| \_NTTREE | macro | Defines the root of a directory tree for storing binary files and other package files.  Only used if separate\_object\_root is enabled.  Set in the build command window. |
| TARGET\_DESTINATION | macro | Defines a subdirectory under \_NTTREE that is the destination for BinPlace files.  Include in *sources* file. |
| NTTARGETFILE*n* | macro | Specifies a custom build step for build pass *n*.  Include in *sources* file. |
| PASS*n*\_BINPLACE | macro | Specifies files that the BinPlace utility places at the end of build pass *n*.  Only used if separate\_object\_root is enabled.  Include in *sources* file. |
| BINPLACE\_LOG | environment variable | Specifies a custom location for the *binplace.log* file. |

# Resources

For answers to any questions about the WDK build environment, send e‑mail to [wdkteam@microsoft.com](mailto:wdkteam@microsoft.com).

This section lists resources for the WDK build environment.

#### MSDN

Tools for Building Drivers

<http://msdn.microsoft.com/en-us/library/ms797164.aspx>

Windows Auto Code Review (OACR)

<http://msdn.microsoft.com/en-us/library/dd445214.aspx>

NMAKE Reference

<http://msdn.microsoft.com/en-us/library/dd9y37ha.aspx>

#### Kits and Tools

Windows 7 WDK

The Windows 7 WDK contains the header files, build environment, and documentation for building drivers.

<http://www.microsoft.com/whdc/DevTools/WDK/WDKpkg.mspx>

# Appendix. Sample Build Log

The following is an annotated build log that illustrates the information that you might see in a build log. This log was generated from a free build of the *firefly* sample in the WDK that was built for Windows 7 running on x86 machines. Important lines are highlighted in color. Lines that start with “//” are added here for explanation, but are not part of the actual log file.

This build was executed on a multiprocessor machine. The “>1” and “>2” marks indicate the two different threads that ran on the two processors.

BUILD: Examining c:\winddk\7100.0\src\hid\firefly directory tree for files to compile.

oacr invalidate wdksamples:x86fre /autocleanqueue

// Start building firefly\driver

1>Building generated files in c:\winddk\7100.0\src\hid\firefly\driver \*\*\*\*\*\*\*\*\*\*\*\*\*

1>'nmake.exe /nologo BUILDMSG=Stop. –i BUILD\_PASS=PASS0 NOLINK=1

PASS0ONLY=1 MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\driver'

1>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\driver

// Start building firefly\sauron

2>Building generated files in c:\winddk\7100.0\src\hid\firefly\sauron \*\*\*\*\*\*\*\*\*\*\*\*\*

2>'nmake.exe /nologo BUILDMSG=Stop. –i BUILD\_PASS=PASS0 NOLINK=1

PASS0ONLY=1 MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\sauron'

2>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\sauron

// Start makefile.inc in firefly\driver at pass 0

1> mofcomp -B:c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.bmf firefly.mof

// Start Midl.exe for firefly\sauron\effects.idl at pass 0

2> midl /Zp8 /IC:\WinDDK\7100.0\inc\atl71 /I..\shared /Ic:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\WDKSamples\inc /Ic:\winddk\7100.0\WDKSamples\inc\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\internal\WDKSamples\inc /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\crt /char unsigned /ms\_ext /c\_ext /proxy c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\effects\_p.c /dlldata c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\dlldata.c /iid c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\effects\_i.c /tlb c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\effects.tlb /header c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\effects.h /cpp\_cmd C:\WinDDK\7100.0\Bin\x86\oacr\oacrcl -DUNICODE -D\_UNICODE /DNTDDI\_VERSION=0x06010000 /Di386 /D\_X86\_ /D\_WCHAR\_T\_DEFINED /no\_stamp /nologo -sal /win32 -target NT60 effects.idl

1>Microsoft (R) MOF Compiler Version 6.1.7017.0

1>Copyright (c) Microsoft Corp. 1997-2006. All rights reserved.

1>Parsing MOF file: firefly.mof

1>MOF file has been successfully parsed

1>Storing Binary MOF data in c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.bmf

1>Done!

1> wmimofck c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.bmf

// Continue with idl processing at pass 0

2>Processing .\effects.idl

1>Microsoft (R) WDM Extensions To WMI MOF Checking Utility Version 1.50.0000

1>Copyright (c) Microsoft Corp. 1997-2000. All rights reserved.

1>Binary mof file c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.bmf expanded to 1232 bytes

1> wmimofck -hc:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\fireflymof.h c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.bmf

1>Microsoft (R) WDM Extensions To WMI MOF Checking Utility Version 1.50.0000

1>Copyright (c) Microsoft Corp. 1997-2000. All rights reserved.

1>Binary mof file c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.bmf expanded to 1232 bytes

2>effects.idl

2>Processing C:\WinDDK\7100.0\inc\api\oaidl.idl

2>oaidl.idl

2>Processing C:\WinDDK\7100.0\inc\api\objidl.idl

2>objidl.idl

2>Processing C:\WinDDK\7100.0\inc\api\unknwn.idl

2>unknwn.idl

2>Processing C:\WinDDK\7100.0\inc\api\wtypes.idl

2>wtypes.idl

2>Processing C:\WinDDK\7100.0\inc\api\basetsd.h

2>basetsd.h

2>Processing C:\WinDDK\7100.0\inc\api\guiddef.h

2>guiddef.h

2>Processing C:\WinDDK\7100.0\inc\api\ocidl.idl

2>ocidl.idl

2>Processing C:\WinDDK\7100.0\inc\api\oleidl.idl

2>oleidl.idl

2>Processing C:\WinDDK\7100.0\inc\api\servprov.idl

2>servprov.idl

2>Processing C:\WinDDK\7100.0\inc\api\urlmon.idl

2>urlmon.idl

2>Processing C:\WinDDK\7100.0\inc\api\msxml.idl

2>msxml.idl

// Start processing firefly\sauron\isauron.idl at pass 0

2> midl /Zp8 /IC:\WinDDK\7100.0\inc\atl71 /I..\shared /Ic:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\WDKSamples\inc /Ic:\winddk\7100.0\WDKSamples\inc\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\internal\WDKSamples\inc /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\crt /char unsigned /ms\_ext /c\_ext /proxy c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\isauron\_p.c /dlldata c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\dlldata.c /iid c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\isauron\_i.c /tlb c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\isauron.tlb /header c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\isauron.h /cpp\_cmd C:\WinDDK\7100.0\Bin\x86\oacr\oacrcl -DUNICODE -D\_UNICODE /DNTDDI\_VERSION=0x06010000 /Di386 /D\_X86\_ /D\_WCHAR\_T\_DEFINED /no\_stamp /nologo -sal /win32 -target NT60 isauron.idl

2>Processing .\isauron.idl

2>isauron.idl

2>Processing C:\WinDDK\7100.0\inc\api\oaidl.idl

2>oaidl.idl

2>Processing C:\WinDDK\7100.0\inc\api\objidl.idl

2>objidl.idl

2>Processing C:\WinDDK\7100.0\inc\api\unknwn.idl

2>unknwn.idl

2>Processing C:\WinDDK\7100.0\inc\api\wtypes.idl

2>wtypes.idl

2>Processing C:\WinDDK\7100.0\inc\api\basetsd.h

2>basetsd.h

2>Processing C:\WinDDK\7100.0\inc\api\guiddef.h

2>guiddef.h

2>Processing C:\WinDDK\7100.0\inc\api\ocidl.idl

2>ocidl.idl

2>Processing C:\WinDDK\7100.0\inc\api\oleidl.idl

2>oleidl.idl

2>Processing C:\WinDDK\7100.0\inc\api\servprov.idl

2>servprov.idl

2>Processing C:\WinDDK\7100.0\inc\api\urlmon.idl

2>urlmon.idl

2>Processing C:\WinDDK\7100.0\inc\api\msxml.idl

2>msxml.idl

// Continue processing of firefly\driver at pass 1

1>Compiling (NoSync) c:\winddk\7100.0\src\hid\firefly\driver \*\*\*\*\*\*\*\*\*\*\*\*\*

1>'nmake.exe /nologo BUILDMSG=Stop. –i BUILD\_PASS=PASS1 NOLINK=1

PASS1\_NOLIB=1 MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\driver'

1>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\driver

// Start processing of firefly\lib at pass 1

2>Compiling (NoSync) c:\winddk\7100.0\src\hid\firefly\lib \*\*\*\*\*\*\*\*\*\*\*\*\*

2>'nmake.exe /nologo BUILDMSG=Stop. –i BUILD\_PASS=PASS1 NOLINK=1

MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\lib'

2>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\lib

2> C:\WinDDK\7100.0\Bin\x86\oacr\oacrcl @c:\winddk\7100.0\src\hid\firefly\lib\objfre\_win7\_x86\i386\cl.rsp

// Start the resource compiler for firefly\driver at pass 1

1> rc.exe -l 409 -DSXS\_PROCESSOR\_ARCHITECTURE="""x86""" -DSXS\_TARGET="""firefly.sys""" -DSYSTEM\_COMPATIBLE\_ASSEMBLY\_NAME="""Microsoft.Windows.SystemCompatible""" -DLSYSTEM\_COMPATIBLE\_ASSEMBLY\_NAME=L"""Microsoft.Windows.SystemCompatible""" -DSXS\_ASSEMBLY\_VERSION="""""" /r /fo c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.res /D\_X86\_=1 /Di386=1 /DSTD\_CALL /DCONDITION\_HANDLING=1 /DNT\_UP=1 /DNT\_INST=0 /DWIN32=100 /D\_NT1X\_=100 /DWINNT=1 /D\_WIN32\_WINNT=0x0601 /DWINVER=0x0601 /D\_WIN32\_IE=0x0800 /DWIN32\_LEAN\_AND\_MEAN=1 /DDEVL=1 /D\_\_BUILDMACHINE\_\_=WinDDK /DFPO=0 /DNDEBUG /D\_DLL=1 /DNDEBUG /DNTDDI\_VERSION=0x06010000 /Ic:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\WDKSamples\inc /Ic:\winddk\7100.0\WDKSamples\inc\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\internal\WDKSamples\inc /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\ddk /IC:\WinDDK\7100.0\inc\ddk /IC:\WinDDK\7100.0\inc\wdf\kmdf\1.9 /IC:\WinDDK\7100.0\inc\crt .\firefly.rc

// Continue building firefly\lib at pass 1

2>Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.207 for 80x86

2>Copyright (C) Microsoft Corporation. All rights reserved.

2>cl /Fo"c:\winddk\7100.0\src\hid\firefly\lib\objfre\_win7\_x86\i386/"

2> /FC

2> /MT

2> /U\_MT

2> /Ii386

2> /I.

2> /I..\shared

2> /Ic:\winddk\7100.0\src\hid\firefly\lib\objfre\_win7\_x86\i386

2> /Ic:\winddk\7100.0\WDKSamples\inc

2> /Ic:\winddk\7100.0\WDKSamples\inc\objfre\_win7\_x86\i386

2> /Ic:\winddk\7100.0\internal\WDKSamples\inc

2> /IC:\WinDDK\7100.0\inc\api

2> /IC:\WinDDK\7100.0\inc\api

2> /IC:\WinDDK\7100.0\inc\crt

2> /D\_X86\_=1

2> /Di386=1

2> /DSTD\_CALL

2> /DCONDITION\_HANDLING=1

2> /DNT\_UP=1

2> /DNT\_INST=0

2> /DWIN32=100

2> /D\_NT1X\_=100

2> /DWINNT=1

2> /D\_WIN32\_WINNT=0x0601

2> /DWINVER=0x0601

2> /D\_WIN32\_IE=0x0800

2> /DWIN32\_LEAN\_AND\_MEAN=1

2> /DDEVL=1

2> /D\_\_BUILDMACHINE\_\_=WinDDK

2> /DFPO=0

2> /DNDEBUG

2> -DUNICODE

2> -D\_UNICODE

2> /DNTDDI\_VERSION=0x06010000

2> /c

2> /Zc:wchar\_t-

2> /Zl

2> /Zp8

2> /Gy

2> /Gm-

2> /W4

2> /WX

2> /WX

2> /Gz

2> /hotpatch

2> /EHs-c-

2> /GR-

2> /GF

2> /GS

2> /Z7

2> /Oxs

2> /Oy-

2> /Z7

2> /DKMDF\_MAJOR\_VERSION\_STRING=01

2> /DKMDF\_MINOR\_VERSION\_STRING=009

2> /wd4603

2> /wd4627

2> /typedil-

2> /FIC:\WinDDK\7100.0\inc\api\warning.h

2> .\luminous.cpp

2>luminous.cpp

// Continue building firefly\driver at pass 1

1>Microsoft (R) Windows (R) Resource Compiler Version 6.1.6908.0

1>Copyright (C) Microsoft Corporation. All rights reserved.

1> C:\WinDDK\7100.0\Bin\x86\oacr\oacrcl @c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\cl.rsp

1>Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.207 for 80x86

1>Copyright (C) Microsoft Corporation. All rights reserved.

1>cl /Fo"c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386/"

1> /FC

1> /Ii386

1> /I.

1> /Ic:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386

1> /Ic:\winddk\7100.0\WDKSamples\inc

1> /Ic:\winddk\7100.0\WDKSamples\inc\objfre\_win7\_x86\i386

1> /Ic:\winddk\7100.0\internal\WDKSamples\inc

1> /IC:\WinDDK\7100.0\inc\api

1> /IC:\WinDDK\7100.0\inc\api

1> /IC:\WinDDK\7100.0\inc\ddk

1> /IC:\WinDDK\7100.0\inc\ddk

1> /IC:\WinDDK\7100.0\inc\wdf\kmdf\1.9

1> /IC:\WinDDK\7100.0\inc\crt

1> /D\_X86\_=1

1> /Di386=1

1> /DSTD\_CALL

1> /DCONDITION\_HANDLING=1

1> /DNT\_UP=1

1> /DNT\_INST=0

1> /DWIN32=100

1> /D\_NT1X\_=100

1> /DWINNT=1

1> /D\_WIN32\_WINNT=0x0601

1> /DWINVER=0x0601

1> /D\_WIN32\_IE=0x0800

1> /DWIN32\_LEAN\_AND\_MEAN=1

1> /DDEVL=1

1> /D\_\_BUILDMACHINE\_\_=WinDDK

1> /DFPO=0

1> /DNDEBUG

1> /D\_DLL=1

1> /DNDEBUG

1> /DNTDDI\_VERSION=0x06010000

1> /c

1> /Zc:wchar\_t-

1> /Zl

1> /Zp8

1> /Gy

1> /Gm-

1> -cbstring

1> /W3

1> /WX

1> /Gz

1> /hotpatch

1> /EHs-c-

1> /GR-

1> /GF

1> /GS

1> /Zi

1> /Oxs

1> /Oy-

1> /Zi

1> /Fdc:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\

1> /DKMDF\_MAJOR\_VERSION=1

1> /DKMDF\_MINOR\_VERSION=9

1> /DKMDF\_MAJOR\_VERSION\_STRING=01

1> /DKMDF\_MINOR\_VERSION\_STRING=009

1> /wd4603

1> /wd4627

1> /typedil-

1> /FIC:\WinDDK\7100.0\inc\api\warning.h

1> .\driver.c .\device.c .\wmi.c .\vfeature.c

1>driver.c

1>device.c

1>wmi.c

// Continue with firefly\lib at pass 1 – the result is Luminous.lib

2> C:\WinDDK\7100.0\Bin\x86\oacr\oacrlink /lib /out:c:\winddk\7100.0\src\hid\firefly\lib\objfre\_win7\_x86\i386\Luminous.lib /IGNORE:4198,4010,4037,4039,4065,4070,4078,4087,4089,4221 /WX /nodefaultlib /machine:ix86 @c:\winddk\7100.0\src\hid\firefly\lib\objfre\_win7\_x86\i386\lib.rsp

1>vfeature.c

2>Microsoft (R) Library Manager Version 9.00.30729.207

2>Copyright (C) Microsoft Corporation. All rights reserved.

2>c:\winddk\7100.0\src\hid\firefly\lib\objfre\_win7\_x86\i386\luminous.obj

// Continue firefly\sauron at pass 1

2>Compiling (NoSync) c:\winddk\7100.0\src\hid\firefly\sauron \*\*\*\*\*\*\*\*\*\*\*\*\*

2>'nmake.exe /nologo BUILDMSG=Stop. –i BUILD\_PASS=PASS1 NOLINK=1

MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\sauron'

2>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\sauron

1>Generating Code...

2>C:\WinDDK\7100.0\Bin\x86\oacr\oacrcl /Ii386 /I. /IC:\WinDDK\7100.0\inc\atl71 /I..\shared /Ic:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\WDKSamples\inc /Ic:\winddk\7100.0\WDKSamples\inc\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\internal\WDKSamples\inc /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\crt /D\_X86\_=1 /Di386=1 /DSTD\_CALL /DCONDITION\_HANDLING=1 /DNT\_UP=1 /DNT\_INST=0 /DWIN32=100 /D\_NT1X\_=100 /DWINNT=1 /D\_WIN32\_WINNT=0x0601 /DWINVER=0x0601 /D\_WIN32\_IE=0x0800 /DWIN32\_LEAN\_AND\_MEAN=1 /DDEVL=1 /D\_\_BUILDMACHINE\_\_=WinDDK /DFPO=0 /DNDEBUG /D\_DLL=1 /D\_MT=1 -DUNICODE -D\_UNICODE /DNTDDI\_VERSION=0x06010000 /D\_ATL\_STATIC\_REGISTRY /c /Zc:wchar\_t- /Zl /Zp8 /Gy /Gm- /W4 /WX /WX /Gz /hotpatch /EHsc /GR /GF /GS /Zi /Oxs /Oy- /Zi /Fdc:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\ /DKMDF\_MAJOR\_VERSION\_STRING=01 /DKMDF\_MINOR\_VERSION\_STRING=009 /wd4603 /wd4627 /typedil- /FIC:\WinDDK\7100.0\inc\api\warning.h /YlSAURON /Ycstdafx.h /Fpc:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\stdafx.pch /Fo"c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\stdafx.obj" /Tp

// Generate precompiled headers for firefly\sauron

2>#include "stdafx.h"

2>Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.207 for 80x86

2>Copyright (C) Microsoft Corporation. All rights reserved.

2>cl

2> /Ii386

2> /I.

2> /IC:\WinDDK\7100.0\inc\atl71

2> /I..\shared

2> /Ic:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386

2> /Ic:\winddk\7100.0\WDKSamples\inc

2> /Ic:\winddk\7100.0\WDKSamples\inc\objfre\_win7\_x86\i386

2> /Ic:\winddk\7100.0\internal\WDKSamples\inc

2> /IC:\WinDDK\7100.0\inc\api

2> /IC:\WinDDK\7100.0\inc\api

2> /IC:\WinDDK\7100.0\inc\crt

2> /D\_X86\_=1

2> /Di386=1

2> /DSTD\_CALL

2> /DCONDITION\_HANDLING=1

2> /DNT\_UP=1

2> /DNT\_INST=0

2> /DWIN32=100

2> /D\_NT1X\_=100

2> /DWINNT=1

2> /D\_WIN32\_WINNT=0x0601

2> /DWINVER=0x0601

2> /D\_WIN32\_IE=0x0800

2> /DWIN32\_LEAN\_AND\_MEAN=1

2> /DDEVL=1

2> /D\_\_BUILDMACHINE\_\_=WinDDK

2> /DFPO=0

2> /DNDEBUG

2> /D\_DLL=1

2> /D\_MT=1

2> -DUNICODE

2> -D\_UNICODE

2> /DNTDDI\_VERSION=0x06010000

2> /D\_ATL\_STATIC\_REGISTRY

2> /c

2> /Zc:wchar\_t-

2> /Zl

2> /Zp8

2> /Gy

2> /Gm-

2> /W4

2> /WX

2> /WX

2> /Gz

2> /hotpatch

2> /EHsc

2> /GR

2> /GF

2> /GS

2> /Zi

2> /Oxs

2> /Oy-

2> /Zi

2> /Fdc:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\

2> /DKMDF\_MAJOR\_VERSION\_STRING=01

2> /DKMDF\_MINOR\_VERSION\_STRING=009

2> /wd4603

2> /wd4627

2> /typedil-

2> /FIC:\WinDDK\7100.0\inc\api\warning.h

2> /YlSAURON /Ycstdafx.h /Fpc:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\stdafx.pch /Fo"c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\stdafx.obj"

2>pch\_hdr.src

// Continue building firefly\driver at pass 1

1>Compiling c:\winddk\7100.0\src\hid\firefly\driver \*\*\*\*\*\*\*\*\*\*\*\*\*

1>'nmake.exe /nologo BUILDMSG=Stop. –i BUILD\_PASS=PASS1 NOLINK=1

MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\driver'

1>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\driver

1> copy .\firefly.inx c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.inf

1> 1 file(s) copied.

1> stampinf -f c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.inf -a x86 -k 1.9

1>Using version information from C:\WinDDK\7100.0\inc\api\ntverp.h

1>Stamping c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.inf [Version] section with DriverVer=03/23/2009,6.1.7100.0

// Continue compiling firefly\sauron at pass 1

2> C:\WinDDK\7100.0\Bin\x86\oacr\oacrcl @c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\cl.rsp

2>Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.207 for 80x86

2>Copyright (C) Microsoft Corporation. All rights reserved.

2>cl /Fo"c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386/"

2> /FC

2> /Ii386

2> /I.

2> /IC:\WinDDK\7100.0\inc\atl71

2> /I..\shared

2> /Ic:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386

2> /Ic:\winddk\7100.0\WDKSamples\inc

2> /Ic:\winddk\7100.0\WDKSamples\inc\objfre\_win7\_x86\i386

2> /Ic:\winddk\7100.0\internal\WDKSamples\inc

2> /IC:\WinDDK\7100.0\inc\api

2> /IC:\WinDDK\7100.0\inc\api

2> /IC:\WinDDK\7100.0\inc\crt

2> /D\_X86\_=1

2> /Di386=1

2> /DSTD\_CALL

2> /DCONDITION\_HANDLING=1

2> /DNT\_UP=1

2> /DNT\_INST=0

2> /DWIN32=100

2> /D\_NT1X\_=100

2> /DWINNT=1

2> /D\_WIN32\_WINNT=0x0601

2> /DWINVER=0x0601

2> /D\_WIN32\_IE=0x0800

2> /DWIN32\_LEAN\_AND\_MEAN=1

2> /DDEVL=1

2> /D\_\_BUILDMACHINE\_\_=WinDDK

2> /DFPO=0

2> /DNDEBUG

2> /D\_DLL=1

2> /D\_MT=1

2> -DUNICODE

2> -D\_UNICODE

2> /DNTDDI\_VERSION=0x06010000

2> /D\_ATL\_STATIC\_REGISTRY

2> /c

2> /Zc:wchar\_t-

2> /Zl

2> /Zp8

2> /Gy

2> /Gm-

2> /W4

2> /WX

2> /WX

2> /Gz

2> /hotpatch

2> /EHsc

2> /GR

2> /GF

2> /GS

2> /Zi

2> /Oxs

2> /Oy-

2> /Zi

2> /Fdc:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\

2> /DKMDF\_MAJOR\_VERSION\_STRING=01

2> /DKMDF\_MINOR\_VERSION\_STRING=009

2> /wd4603

2> /wd4627

2> /typedil-

2> /FIC:\WinDDK\7100.0\inc\api\warning.h

2> /Yustdafx.h

2> /Fpc:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\stdafx.pch

2> .\saurondll.cpp .\sauron.cpp

2>saurondll.cpp

2>sauron.cpp

2>Generating Code...

2> C:\WinDDK\7100.0\Bin\x86\oacr\oacrlink /lib /out:c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\SAURON.lib @c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\lib.rsp

// Generate import library for firefly\sauron at pass 1

2>Microsoft (R) Library Manager Version 9.00.30729.207

2>Copyright (C) Microsoft Corporation. All rights reserved.

2>/IGNORE:4198,4010,4037,4039,4065,4070,4078,4087,4089,4221

2>/WX

2>/nodefaultlib

2>/machine:ix86

2>/def:SauronDll.def

2>c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\saurondll.obj

2>c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\sauron.obj

2>c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\stdafx.obj

2> Creating library

c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\SAURON.lib and object

c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\SAURON.exp

// Compiling firefly\app at pass 2

1>Compiling and Linking c:\winddk\7100.0\src\hid\firefly\app \*\*\*\*\*\*\*\*\*\*\*\*\*

1>'nmake.exe /nologo BUILDMSG=Stop. –i BUILD\_PASS=PASS2 LINKONLY=1

NOPASS0=1 MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\app'

1>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\app

// Linking firefly\driver at pass 2

2>Linking for c:\winddk\7100.0\src\hid\firefly\driver \*\*\*\*\*\*\*\*\*\*\*\*\*

2>'nmake.exe /nologo BUILDMSG=Stop. –i BUILD\_PASS=PASS2 LINKONLY=1

NOPASS0=1 MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\driver'

2>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\driver

1>BUILDMSG: \_NT\_TARGET\_VERSION SET TO WINXP

2> C:\WinDDK\7100.0\Bin\x86\oacr\oacrlink

/out:c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.sys /machine:ix86

@c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\lnk.rsp

1> C:\WinDDK\7100.0\Bin\x86\oacr\oacrcl @c:\winddk\7100.0\src\hid\firefly\app\objfre\_win7\_x86\i386\cl.rsp

2>Microsoft (R) Incremental Linker Version 9.00.30729.207

2>Copyright (C) Microsoft Corporation. All rights reserved.

2>/MERGE:\_PAGE=PAGE

2>/MERGE:\_TEXT=.text

2>/SECTION:INIT,d

2>/OPT:REF

2>/OPT:ICF

2>/IGNORE:4198,4010,4037,4039,4065,4070,4078,4087,4089,4221

2>/INCREMENTAL:NO

2>/release

2>/NODEFAULTLIB

2>/WX

2>/debug

2>/debugtype:cv,fixup,pdata

2>/version:6.1

2>/osversion:6.1

2>/functionpadmin:5

2>/safeseh

2>/pdbcompress

2>/STACK:0x40000,0x1000

2>/driver

2>/base:0x10000

2>/subsystem:native,6.01

2>/entry:FxDriverEntry@8

2>/out:c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.sys

2>c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\firefly.res

2>c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\driver.obj

2>c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\device.obj

2>c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\wmi.obj

2>c:\winddk\7100.0\src\hid\firefly\driver\objfre\_win7\_x86\i386\vfeature.obj

2>C:\WinDDK\7100.0\lib\win7\i386\BufferOverflowK.lib

2>C:\WinDDK\7100.0\lib\win7\i386\ntoskrnl.lib

2>C:\WinDDK\7100.0\lib\win7\i386\hal.lib

2>C:\WinDDK\7100.0\lib\win7\i386\wmilib.lib

2>C:\WinDDK\7100.0\lib\wdf\kmdf\i386\1.9\WdfLdr.lib

2>C:\WinDDK\7100.0\lib\wdf\kmdf\i386\1.9\WdfDriverEntry.lib

2>C:\WinDDK\7100.0\lib\win7\i386\hidparse.lib

// Continue compiling firefly\app at pass 2

1>Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.207 for 80x86

1>Copyright (C) Microsoft Corporation. All rights reserved.

1>cl /Fo"c:\winddk\7100.0\src\hid\firefly\app\objfre\_win7\_x86\i386/"

1> /FC

1> /Ii386

1> /I.

1> /I..\shared

1> /Ic:\winddk\7100.0\src\hid\firefly\app\objfre\_win7\_x86\i386

1> /Ic:\winddk\7100.0\WDKSamples\inc

1> /Ic:\winddk\7100.0\WDKSamples\inc\objfre\_win7\_x86\i386

1> /Ic:\winddk\7100.0\internal\WDKSamples\inc

1> /IC:\WinDDK\7100.0\inc\api

1> /IC:\WinDDK\7100.0\inc\api

1> /IC:\WinDDK\7100.0\inc\crt

1> /D\_X86\_=1

1> /Di386=1

1> /DSTD\_CALL

1> /DCONDITION\_HANDLING=1

1> /DNT\_UP=1

1> /DNT\_INST=0

1> /DWIN32=100

1> /D\_NT1X\_=100

1> /DWINNT=1

1> /D\_WIN32\_WINNT=0x0501

1> /DWINVER=0x0501

1> /D\_WIN32\_IE=0x0603

1> /DWIN32\_LEAN\_AND\_MEAN=1

1> /DDEVL=1

1> /D\_\_BUILDMACHINE\_\_=WinDDK

1> /DFPO=0

1> /DNDEBUG

1> /D\_DLL=1

1> /D\_MT=1

1> -DUNICODE

1> -D\_UNICODE

1> /DNTDDI\_VERSION=0x05010200

1> /DPSAPI\_VERSION=1

1> /c

1> /Zc:wchar\_t-

1> /Zl

1> /Zp8

1> /Gy

1> /Gm-

1> /W4

1> /WX

1> /WX

1> /Gz

1> /hotpatch

1> /EHs-c-

1> /GR-

1> /GF

1> /GS

1> /Zi

1> /Oxs

1> /Oy-

1> /Zi

1> /Fdc:\winddk\7100.0\src\hid\firefly\app\objfre\_win7\_x86\i386\

1> /DKMDF\_MAJOR\_VERSION\_STRING=01

1> /DKMDF\_MINOR\_VERSION\_STRING=009

1> /wd4603

1> /wd4627

1> /typedil-

1> /FIC:\WinDDK\7100.0\inc\api\warning.h

1> .\firefly.cpp

1>firefly.cpp

// Linking firefly\sauron at pass 2

2>Linking for c:\winddk\7100.0\src\hid\firefly\sauron \*\*\*\*\*\*\*\*\*\*\*\*\*

2>'nmake.exe /nologo BUILDMSG=Stop. –i BUILD\_PASS=PASS2 LINKONLY=1

NOPASS0=1 MAKEDLL=1 MAKEDIR\_RELATIVE\_TO\_BASEDIR=src\hid\firefly\sauron'

2>BUILDMSG: Processing c:\winddk\7100.0\src\hid\firefly\sauron

// Linking firefly\app at pass 2

1> C:\WinDDK\7100.0\Bin\x86\oacr\oacrlink

/out:c:\winddk\7100.0\src\hid\firefly\app\objfre\_win7\_x86\i386\flicker.exe /machine:ix86

@c:\winddk\7100.0\src\hid\firefly\app\objfre\_win7\_x86\i386\lnk.rsp1>Microsoft (R) Incremental Linker Version 9.00.30729.207

1>Copyright (C) Microsoft Corporation. All rights reserved.

1>/MERGE:\_PAGE=PAGE

1>/MERGE:\_TEXT=.text

1>/SECTION:INIT,d

1>/OPT:REF

1>/OPT:ICF

1>/IGNORE:4198,4010,4037,4039,4065,4070,4078,4087,4089,4221

1>/INCREMENTAL:NO

1>/release

1>/NODEFAULTLIB

1>/WX

1>/debug

1>/debugtype:cv,fixup,pdata

1>/version:6.1

1>/osversion:6.1

1>/functionpadmin:5

1>/safeseh

1>/nxcompat

1>/merge:.rdata=.text

1>/pdbcompress

1>/STACK:0x40000,0x2000

1>/tsaware

1>/dynamicbase

1>/subsystem:console,5.01

1>/base:0x400000

1>/entry:mainCRTStartup

1>c:\winddk\7100.0\src\hid\firefly\app\objfre\_win7\_x86\i386\firefly.obj

1>c:\winddk\7100.0\lib\wxp\i386\advapi32.lib

1>c:\winddk\7100.0\lib\wxp\i386\kernel32.lib

1>c:\winddk\7100.0\lib\wxp\i386\msvcrt\_winxp.obj

1>C:\WinDDK\7100.0\lib\crt\i386\msvcrt.lib

1>c:\winddk\7100.0\src\hid\firefly\app\..\lib\objfre\_win7\_x86\i386\luminous.lib

1>c:\winddk\7100.0\lib\wxp\i386\ole32.lib

1>c:\winddk\7100.0\lib\wxp\i386\oleaut32.lib

1>c:\winddk\7100.0\lib\wxp\i386\wbemuuid.lib

1>c:\winddk\7100.0\lib\wxp\i386\sehupd.lib

// Resource compilation and linking for firefly\sauron at pass 2

2> rc.exe -l 409 -DSXS\_PROCESSOR\_ARCHITECTURE="""x86""" -DSXS\_TARGET="""SAURON.DLL""" -DSYSTEM\_COMPATIBLE\_ASSEMBLY\_NAME="""Microsoft.Windows.SystemCompatible""" -DLSYSTEM\_COMPATIBLE\_ASSEMBLY\_NAME=L"""Microsoft.Windows.SystemCompatible""" -DSXS\_ASSEMBLY\_VERSION="""""" -n /r /fo c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\saurondll.res /D\_X86\_=1 /Di386=1 /DSTD\_CALL /DCONDITION\_HANDLING=1 /DNT\_UP=1 /DNT\_INST=0 /DWIN32=100 /D\_NT1X\_=100 /DWINNT=1 /D\_WIN32\_WINNT=0x0601 /DWINVER=0x0601 /D\_WIN32\_IE=0x0800 /DWIN32\_LEAN\_AND\_MEAN=1 /DDEVL=1 /D\_\_BUILDMACHINE\_\_=WinDDK /DFPO=0 /DNDEBUG /D\_DLL=1 /D\_MT=1 -DUNICODE -D\_UNICODE /DNTDDI\_VERSION=0x06010000 /D\_ATL\_STATIC\_REGISTRY /IC:\WinDDK\7100.0\inc\atl71 /I..\shared /Ic:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\WDKSamples\inc /Ic:\winddk\7100.0\WDKSamples\inc\objfre\_win7\_x86\i386 /Ic:\winddk\7100.0\internal\WDKSamples\inc /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\api /IC:\WinDDK\7100.0\inc\crt .\saurondll.rc

2>Microsoft (R) Windows (R) Resource Compiler Version 6.1.6908.0

2>Copyright (C) Microsoft Corporation. All rights reserved.

2> C:\WinDDK\7100.0\Bin\x86\oacr\oacrlink

/out:c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\SAURON.DLL /machine:ix86

@c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\lnk.rsp2>Microsoft (R) Incremental Linker Version 9.00.30729.207

2>Copyright (C) Microsoft Corporation. All rights reserved.

2>/MERGE:\_PAGE=PAGE

2>/MERGE:\_TEXT=.text

2>/SECTION:INIT,d

2>/OPT:REF

2>/OPT:ICF

2>/IGNORE:4198,4010,4037,4039,4065,4070,4078,4087,4089,4221

2>/INCREMENTAL:NO

2>/release

2>/NODEFAULTLIB

2>/WX

2>/debug

2>/debugtype:cv,fixup,pdata

2>/version:6.1

2>/osversion:6.1

2>/functionpadmin:5

2>/safeseh

2>/nxcompat

2>/merge:.rdata=.text

2>/pdbcompress

2>/dynamicbase

2>/STACK:0x40000,0x1000

2>/dll

2>/base:@C:\WinDDK\7100.0\bin\coffbase.txt,SAURON

2>/subsystem:windows,6.01

2>/entry:\_DllMainCRTStartup@12

2>c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\stdafx.obj

2>c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\SAURON.exp

2>c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\saurondll.res

2>c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\saurondll.obj

2>c:\winddk\7100.0\src\hid\firefly\sauron\objfre\_win7\_x86\i386\sauron.obj

2>C:\WinDDK\7100.0\lib\atl\i386\atls.lib

2>C:\WinDDK\7100.0\lib\atl\i386\atlthunk.lib

2>C:\WinDDK\7100.0\lib\crt\i386\msvcrt.lib

2>C:\WinDDK\7100.0\lib\win7\i386\advapi32.lib

2>C:\WinDDK\7100.0\lib\win7\i386\comctl32.lib

2>C:\WinDDK\7100.0\lib\win7\i386\kernel32.lib

2>C:\WinDDK\7100.0\lib\win7\i386\ole32.lib

2>C:\WinDDK\7100.0\lib\win7\i386\oleaut32.lib

2>C:\WinDDK\7100.0\lib\win7\i386\user32.lib

2>C:\WinDDK\7100.0\lib\win7\i386\gdi32.lib

2>C:\WinDDK\7100.0\lib\win7\i386\uuid.lib

2>C:\WinDDK\7100.0\lib\win7\i386\wbemuuid.lib

2>c:\winddk\7100.0\src\hid\firefly\sauron\..\lib\objfre\_win7\_x86\i386\luminous.lib