C

Building Kernel-Mode Drivers

The *BUILD.EXE* utility supplied with the Windows NT DDK is the Microsoft-provided (and supported) program to assist developers in compiling and linking Windows NT kernel-mode drivers.* *BUILD* automatically establishes file dependencies, isolates target-dependent (e.g., header file and library include path information, compiler names and switches, linker switches) and platform-dependent information (x86, PowerPC, MIPS, Alpha platforms), and, thereby, provides a simple method for creating kernel-mode drivers and libraries and also user-mode applications (if you so desire).

BUILD expects you to tell it which files need to be compiled, the name and type information for the target driver or library to be generated, information identifying the target platform, and any special compilation and/or link options that you would like to specify. Once you provide this information, *BUILD* determines the appropriate file dependencies and uses the services of *NMAKE* and the installed compiler to generate your target driver, library, or application.

Inputs

As input, *BUILD* expects you to supply a text file named *SOURCES*, in the same directory as that containing the files to be compiled. The *SOURCES* file contains information that identifies your source files, the target to be built and other relevant information. *BUILD* also uses any environment variables that you may have specified, and any command-line options that you provide when you invoke *BUILD*. Finally, *BUILD* uses the platform-specific rules and other options specified

* Note once again that the Microsoft DDK does not include a compiler. You must purchase a compiler separately.

in the following files that are supplied with the DDK (in the *$BASEDIR\INC* directory*):

MAKEFILE.DEF

This is the primary control file used by *BUILD.* You should study the contents of this file to better understand how options specified by you can affect the manner in which your target driver or library is created.

WARNING The *MAKEFILE.DEF* file is poorly documented and extremely convoluted. It is a prime example of how not to implement a makefile. Unfortunately for us, it is also the only source available if we wish to understand some of what happens when *BUILD* is invoked.

MAKEFILE.PLT

This file contains target-platform-specific information. The target platform is either specified by you as a command-line option to *BUILD* or determined via an environment variable.t This file is included by *MAKEFILE.DEF.*

I386MK.INC / ALPHAMK.INC / MIPSMK.INC / PPCMK.INC

This file contains target-platform-specific build controls and is also included by *MAKEFILE.DEF.*

Output

BUILD generates the target driver, library, or application you specified in the *SOURCES* file. The *BUILD.LOG* (containing the list of commands invoked by *BUILD* and any compiler- or linker-generated statements), *BUILD. WJRN* (containing warnings generated during the build process), and *BUILD.EKR* (containing errors that prevent the successful completion of the build process) files are generated as by-products of the build process.

Two types of Windows NT drivers can be built:

Free build

This is the nondebug version of your driver. This is also the version you will typically ship to your customers. This version is normally compiled with full optimization enabled, and I would advise that you strip the free version of all symbolic information before shipping it.

---

* The *BASEDIK* environment variable is automatieally set up for you by the installation utility during the DDK installation process and its value is set to the base directory path specification where you installed the DDK.

t The *BUILD_DEFAULT_TARCETS*environment variable is typically initialized to the target platform type value (e.g., -386).

The free build environment does not define the DBG environment variable; therefore, any conditional debug code that you include in your driver can be automatically filtered out during the compilation process as shown here:

```
—if DBG
    // Include the debug code here. This code is automatically
    // filtered out for the free/retail build.
—endif // DBG
```

Contrary to what you might expect, the free version of your driver does contain symbolic information. To remove this symbolic information from the free build, execute the following sequence of steps:

— Execute the command *DUMPBIN /HEADERS y our -driver -name . sys / MORE* on the binary.

— Note the value associated with the *"image base"* in the OPTIONAL HEADER VALUES section. This value should typically be 10000 (hex).

— Execute the command *REBASE -b InitialBaseValue -x Symbol-Dir-Name y our -driver -name . sys.*

— A file by the name of *your-driver-name . dbg* will be created in the specified symbol directory, and the free binary will no longer contain any symbolic information.

Checked build

This is the debug version of your driver that you will typically build and use during development. Assertions in the driver code, debug print statements, debug breakpoints, and symbolic information are all compiled into the checked binary, and optimization is disabled for the debug build.

You should never ship the checked build to your customers or attempt to execute the checked build without having a debugger attached to the system. Otherwise, you may experience hangs and/or system crashes when assertions or breakpoints are hit.

To build the free, or retail, version of your driver, use the free build environment, which is set up automatically when you invoke the Free Build Command Window. Similarly, you should use the Checked Build Command Window to build the checked version of your target driver.

Building Your Driver

1. In your driver source directory, create a file called *MAKEFILE* containing the following:

```
# DO NOT EDIT THIS FILE! ! !  Edit .\sources. if you want to add a new
# source file to this component.  This file merely indirects to the
```

```
# real make filethat is shared by all the driver components of the
# Windows NT DDK

!INCLUDE $(NTMAKEENV)\makefile.def
```

2. Also in your driver source directory, create a *SOURCES* file, specifying the source files to be built, the target type and name, and any other additional command-line switch values you wish to have passed-on to the compiler and linker. The *\DDK\DOC\SOURCES.TPLfi\e* is a template that you should study and use when attempting to create your own *SOURCES* file. Below is the *SOURCES* file I used in compiling the sample file system driver:

```
# - Execute the "build" command to make the sample FSD driver

# The TARGETNAME variable is defined by the developer.  It is the name
# of the target (component) that is being built by this makefile.  It
# should NOT include any path or file extension information.

TARGETNAME=sfsd

# The TARGETPATH and TARGETTYPE variables are defined by the developer.
# The first specifies where the target is to be built.  The second
# specifies the type of target (either PROGRAM, DYNLINK, LIBRARY,
# UMAPPL_NOLIB, or BOOTPGM).  UMAPPL_NOLIB is used when you're only
# building user-mode apps and don't need to build a library.

TARGETPATH=obj

TARGETTYPE=DRIVER

# The INCLUDES variable specifies any include paths that are specific
# to this source directory.  Separate multiple directory paths with
# singlesemicolons.  Relative path specifications are okay.  The
# INCLUDES variable is not required.  Specifying an empty INCLUDES
# variable(i.e., INCLUDES= ) indicates no include paths are to be
# searched.
#
# NOTE: The "fsdk\inc" refers to the Microsoft-supplied File Systems
#       Developers Kit.

INCLUDES=. .\inc;\ddk-40\inc;\fsdk\inc-40;

# The SOURCES variable is defined by the developer.  It is a list of
# all the source files for this component.  Each source file should be
# on a separate line, using the line continuation character.  This
# will minimize merge conflicts if two developers are adding source
# files to the same component.  The SOURCES variable is required.  If
# there are no platform common sourcefiles, an empty SOURCES variable
# should be used, (i.e., SOURCES= )

# Source files common to multiple platforms

SOURCES=sfsdinit.c      \
        registry.c      \
```

```
create.c        \
misc.c          \
cleanup.c       \
close.c         \
read.c          \
write.c         \
fileinfo.c      \
flush.c         \
volinfo.c       \
dircntrl.c      \
fscntrl.c       \
devcntrl.c      \
shutdown.c      \
Ickcntrl.c      \
security.c      \
extattr.c       \
fastio.c
```

```
# Next specify any additional options for the compiler.
# Define the appropriate CPU type (and insert defines
# in the appropriate header file) to get the right
# values for "uint8," "uint16," etc. typedefs.

C_DEFINES=   -DUNICODE  -D_CPU_X86_

# The type of product being built - NT = kernel mode
UMTYPE=nt
```

3. Since I specified that the *obj* subdirectory should contain the target driver, I have created the *obj*, *obj\i386*, *obj\i386\cbecked*, and *obj\i386\free* directories on my computer, for creating the x86 version of the driver. Similarly, you should create the appropriate directories depending upon the target directory you specify and the target platform that you are compiling for.*
4. Execute *BUILD* to create your target driver, library, or application.

For More Information

The DDK documentation provides a guide to compiling and linking your kernel-mode drivers. Consult this documentation, the *SOURCES* files provided on the diskette accompanying this book and the *SOURCES* files that are supplied with sample source code provided with the DDK for more information.

* If you fail to do so, *BUILD* will automatically create all of the subdirectories mentioned here, except the *checked* and *free* sub-directories, which you will have to create yourself (manually).