

# Installing, Using and Testing Microsoft Windows Driver Kit (WDK)

What do we have in this session?

## **Introduction**

### **Installing the WDK**

### **The Windows Debuggers**

### **Installing Symbols Package**

### **The WinDbg**

### **Starting WinDbg**

### **Verifying WDK Installation**

### **Steps on Using WDK and Building Windows Driver from Code Sample**

### **Building a Windows Driver Sample**

### **Drivers and Services**

### **Installing/Registering a Driver**

### **Starting a Driver**

### **Viewing Driver Output**

### **Unloading a Driver**

### **Print Devices: Generic Text-Only Driver Sample**

### **Useful WDK references:**

## **Introduction**

The Windows machine specification used in this session are:

1. Operating System : Windows XP Pro SP2
2. RAM : 2 GB DDR2
3. HDD : 160++GB
4. Display : 128 MB ATI PCI Express
5. Processor : Intel Core 2 Duo 4400 2.00 GHz

The DDK has been superseded by the Windows Driver Kit (WDK). While the DDK can be downloaded openly, you may need to register for free and participate in the respective Microsoft community to download WDK. Microsoft said that the WDK should be used for the following reasons:

1. Use the Windows Vista build environments in the WDK to build drivers that use new features or functionality available only in Windows Vista.
2. Use the Windows Server 2003 build environments in the WDK to build drivers that use new features or functionality available only in Windows Server 2003.
3. Use the Windows XP build environments in the WDK to build drivers that do not use new functionality available only in Windows Vista or only in Windows Server 2003 and that are targeted for either Windows XP or Windows Server

2003 and Windows XP. The Windows XP build environments in the WDK contain minor updates to the Windows DDK that shipped with Windows XP SP1 and with Windows XP.

You should use the Windows 2000 build environments in the WDK to build drivers designed to run on Windows Vista, Windows Server 2003, Windows XP, or Windows 2000. The Windows 2000 build environment in the WDK includes updated headers and libraries for Windows 2000 Service Pack 4 (SP4).

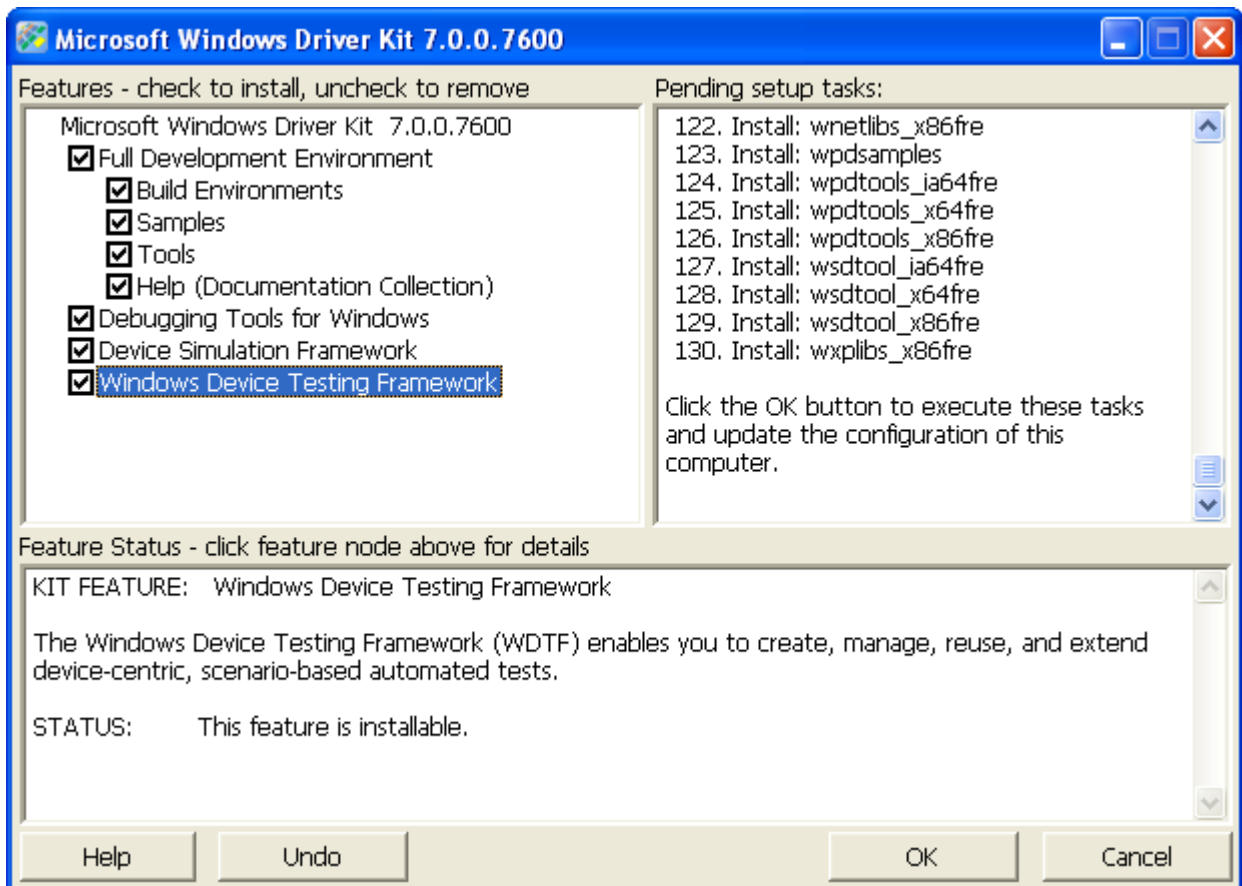
### **Installing the WDK**

Firstly, go to the [WDK site](#) to download the ISO. If you download the ISO, you need to burn it on the CD to make it useable. If needed, you can rename the .ISO extension to .IMG. If you want to get an idea how to install/extract the .IMG/.ISO files, go to [this link](#). Newer DVD/CD-ROM can read directly the ISO, so no need for us to burn it and you can skip the ISO burning steps.

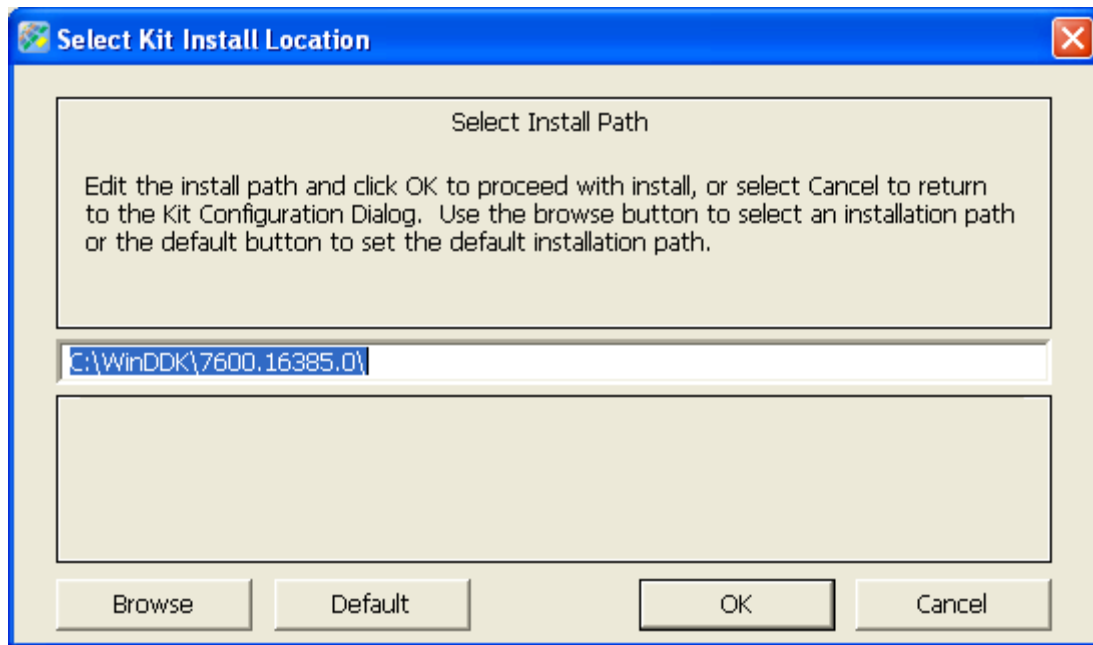
Then we are ready to install the WDK. Insert the previously burned ISO CD into the DVD/CD-ROM drive and the autorun should be launched welcoming you with the following splash screen. Click the splash screen.



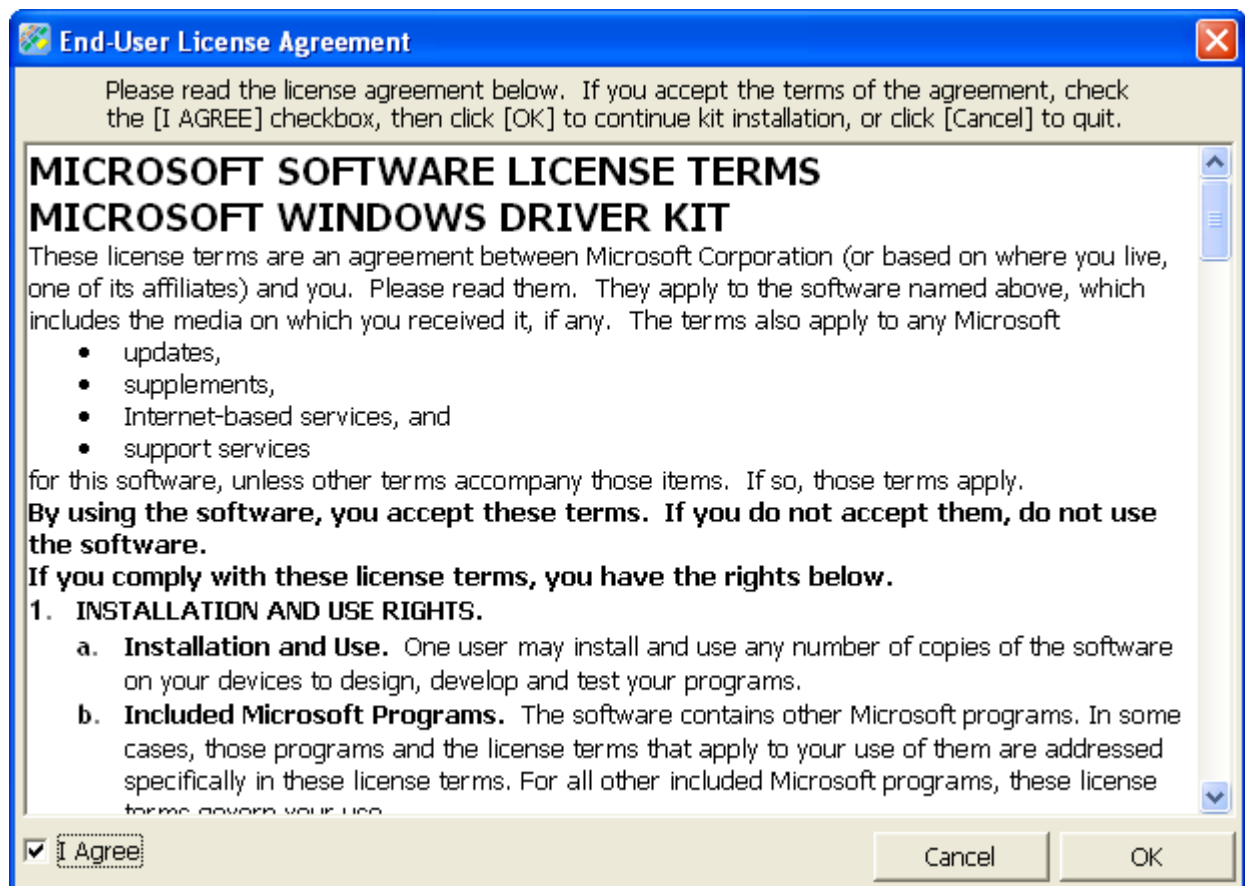
Select the features that need to be installed. In this case we select all the features. When you select any of the features, the description will be in the bottom window. Click the OK button to continue.



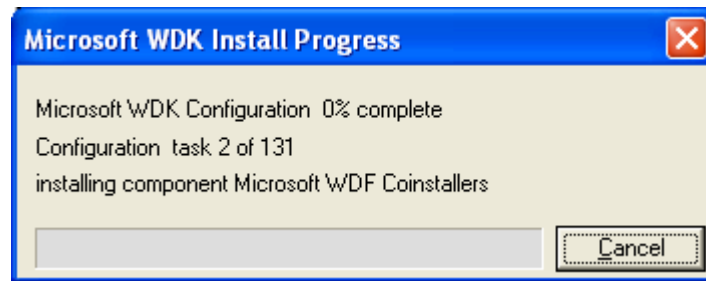
Next, the default path was given to us. Edit or use the Browse button if you need to modify the installation path. In this case we just accept the given path. Click the OK button.



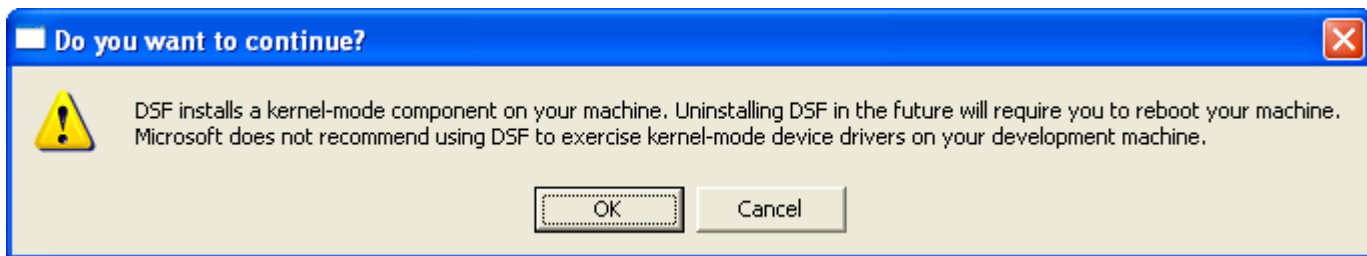
Accept the EULA and click the OK button.



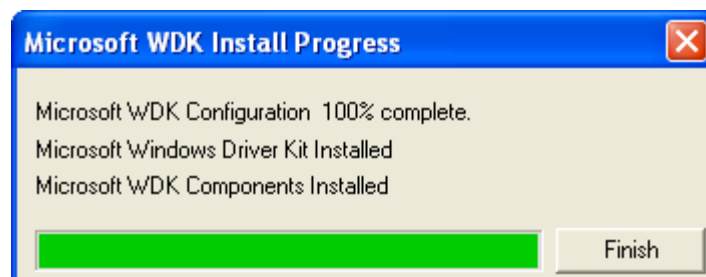
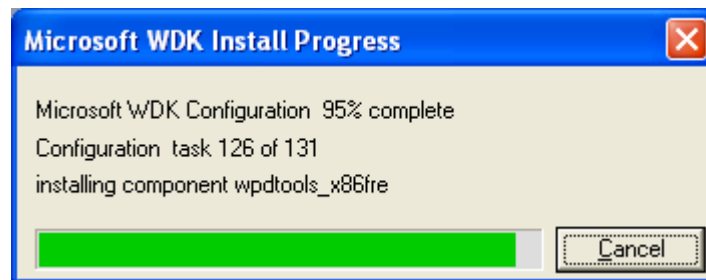
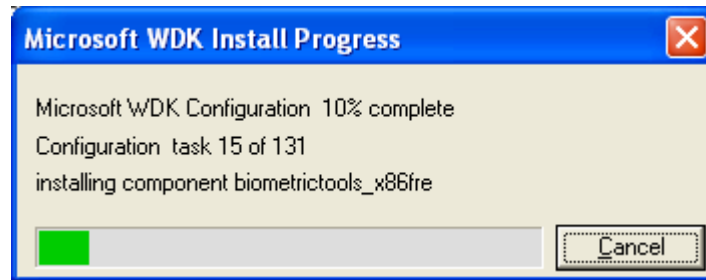
The installation progress window will start. You can cancel the installation any time.



Just accept the Device Simulation Framework (DSF) warning page.

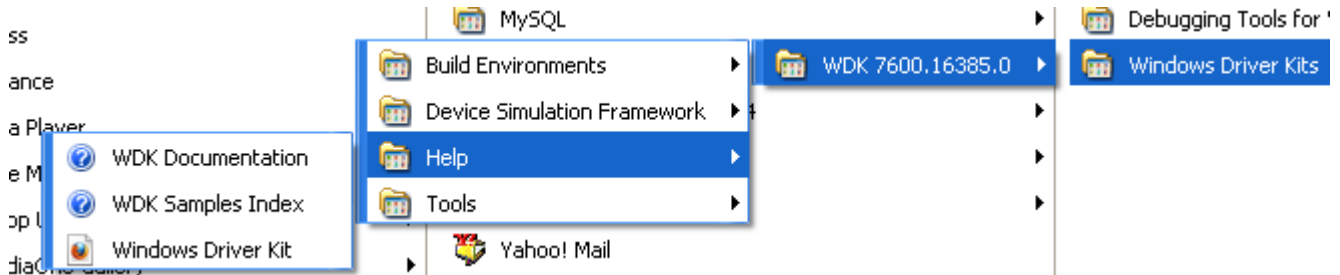


The installation should begin. Just wait until it finish.



If there is no error, the installation should be completed successfully at this stage. Any error should be welcomed with Windows error message and you should take appropriate action for re-installation.

Then you can verify the installation through the Start menu as shown below.



## The Windows Debuggers

Debugging Tools for Windows features WinDbg, a powerful debugger with a graphical interface that can debug both user-mode and kernel-mode code. Debugging Tools for Windows also includes:

1. KD - Command-line kernel debugger.
2. NTSD - Command-line user-mode debugger.
3. CDB - Command-line user-mode debugger (variant of NTSD).

and many additional tools as listed in the following Table.

Tool	Description
Logger (Logger.exe and Logexts.dll)	A tool and an extension DLL that record the function calls and other actions of a program.
LogViewer (Logviewer.exe)	A tool that displays the logs created by Logger.
ADPlus (Autodump+, Adplus.vbs)	A console-based Microsoft Visual Basic script that can automatically create memory dump files and log files with debug output from one or more processes.
DbgRpc (Dbgrpc.exe)	A tool used to display Microsoft Remote Procedure Call (RPC) state information.
KDBGCtrl (Kernel Debugging Control, Kdbgctrl.exe)	A tool that controls and configures the kernel debugging connection.
SrcSrv (Srcsrv.dll)	A source server that can be used to deliver source files while debugging.
SymSrv (Symsrv.dll)	A symbol server that the debugger can use to connect to a symbol store.
SymStore (Symstore.exe)	A tool used to create a symbol store.
SymProxy	A tool used to create a single HTTP symbol server on your network that all your debuggers can point to. This has the benefit of pointing to

	multiple symbol servers (both internal and external) with a single symbol path, handling all authentication, and increasing performance via symbol caching.
AgeStore (Agestore.exe)	A tool that removes old entries in the downstream store of a symbol server or a source server.
DBH (Dbh.exe)	A tool that displays information about the contents of a symbol file.
PDBCopY (Pdbcopy.exe)	A tool that removes private symbol information from a symbol file, and controls which public symbols are included in the file.
DumpChk (Dump File Checking Utility, Dumpchk.exe)	A tool used to validate a memory dump file.
DbgSrv (Dbgsrv.exe)	A process server used for remote debugging.
KdSrv (Kdsrv.exe)	A KD connection server used for remote debugging.
DbEngPrx (Dbengprx.exe)	A repeater (small proxy server) used for remote debugging.
The Remote tool (Remote.exe)	A remoting tool that can be used to remotely control any console program, including KD, CDB, and NTSD.
GFlags (Global Flags Editor, Gflags.exe)	A tool used to control registry keys and other settings.
The Kill tool (Kill.exe)	A tool used to terminate a process.
The Breakin tool (Breakin.exe)	A tool used to cause a user-mode break to occur in a process.
The List tool (List.exe)	File List Utility
TList (Tlist.exe)	Task List Viewer
RTLlist (Rtlist.exe)	Remote Task List Viewer. A tool used to list running processes via a DbgSrv process server.
UMDH (User-Mode Dump Heap utility, Umdh.exe)	A tool used to analyze heap allocations.
USBView (Universal Serial Bus Viewer, Usbview.exe)	A tool used to display the USB devices connected to a computer.

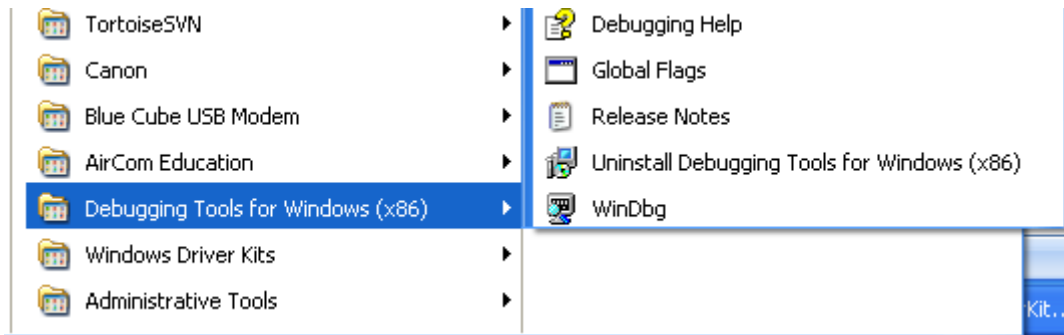
The documentation in Debugging Tools for Windows (Debugging Help) describes the use of these debuggers and includes tips for user-mode and kernel-mode debugging. Debugging Tools for Windows is available in three different versions:

1. A 32-bit version.
2. A native Intel Itanium version, and
3. A native x64 version.

The 32-bit version is appropriate for most users. If you are planning on debugging a user-mode application on an Itanium-based processor, you should install the Itanium version

of the debuggers. If you are planning on debugging a user-mode application on an x64 processor, you should select the x64 version of the debuggers. The Itanium and x64 debuggers can be installed only on 64-bit versions of Windows. These debugging tools require approximately 25 MB of hard disk space.

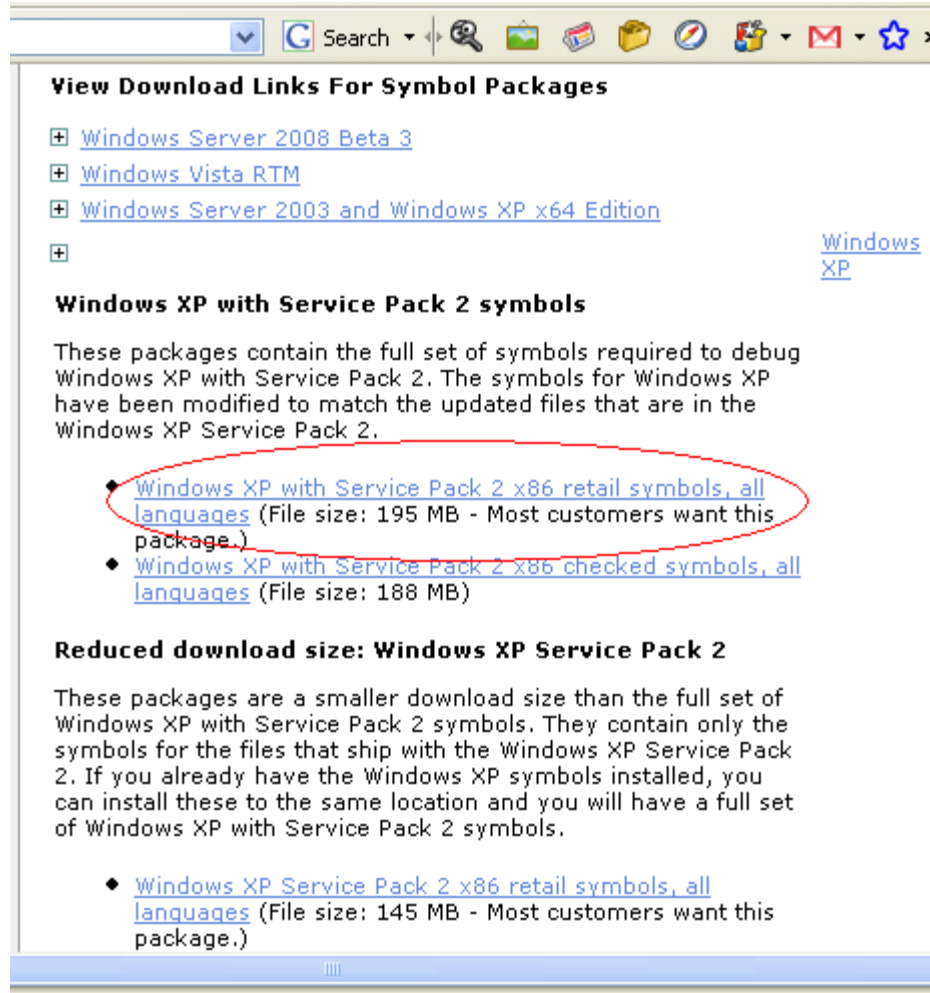
To obtain the most current version of Debugging Tools for Windows, visit the [Microsoft Debugging Tools](http://Microsoft.com/DebuggingTools) web site. It seems that the Debugging Tools for Windows (x86) was installed together during the WDK installation.



## Installing The Symbols Package

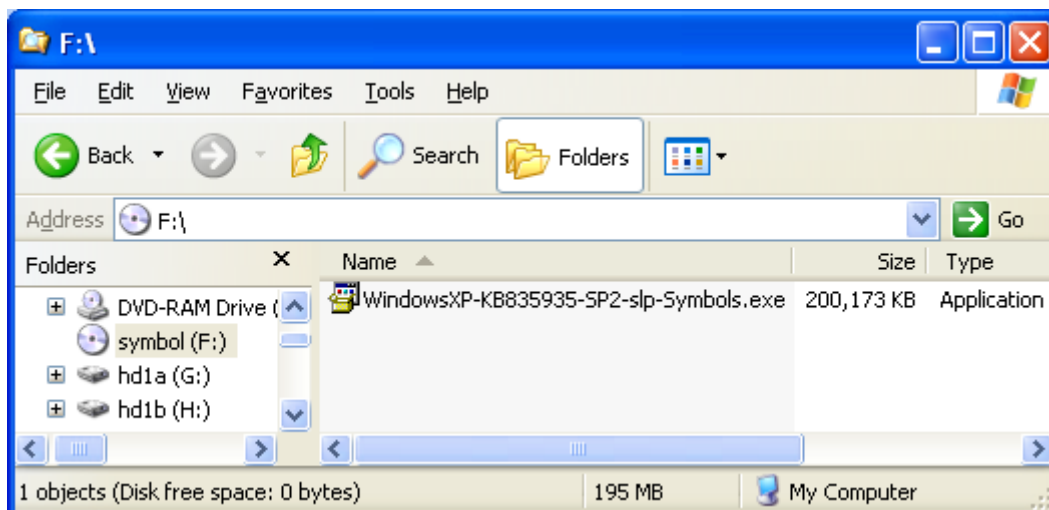
As said by Microsoft, the new linker strips all the debug information from the SYS file and moves the data into a PDB file. The PDB file should be copied to the symbols directory for debugging. Copying the SYS file will not provide debugging information. If you want the entire set of symbols for Windows Vista, Windows Server 2008, Windows XP, or Windows 2000, then you can [download a symbol package and install it on your computer](#).



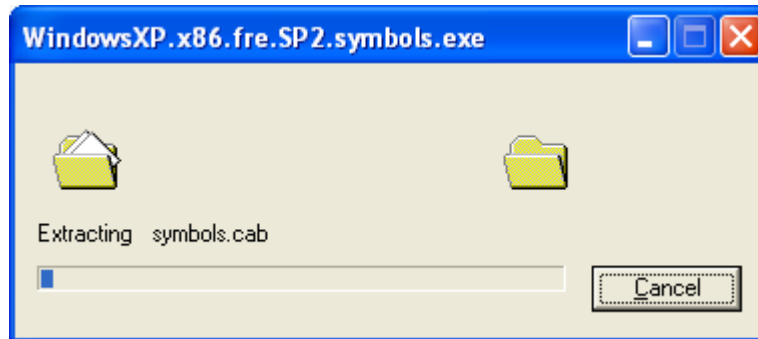


Selecting Windows XP with Service Pack 2 retail symbols for downloading

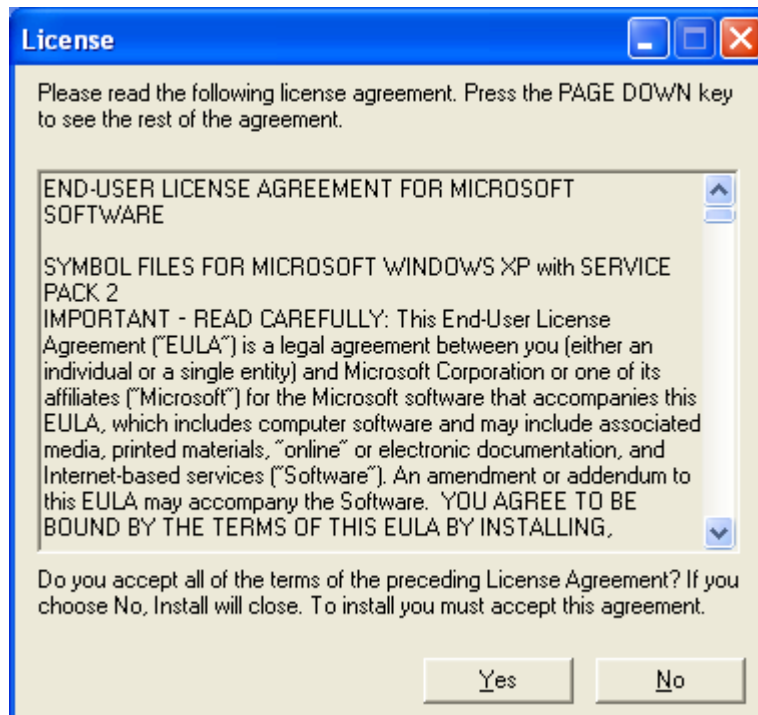
In our case, because we can't be online all the time, we have downloaded a package for Windows XP SP2 symbols package and run the executable file (~ 200MB).



The Windows XP with Service Pack 2 retail symbols self-extraction file

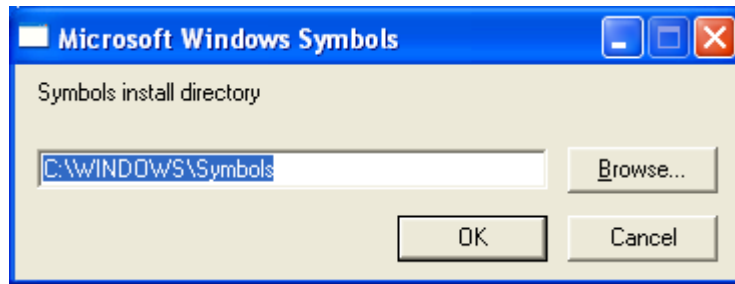


The Windows XP with Service Pack 2 retail symbols file extraction

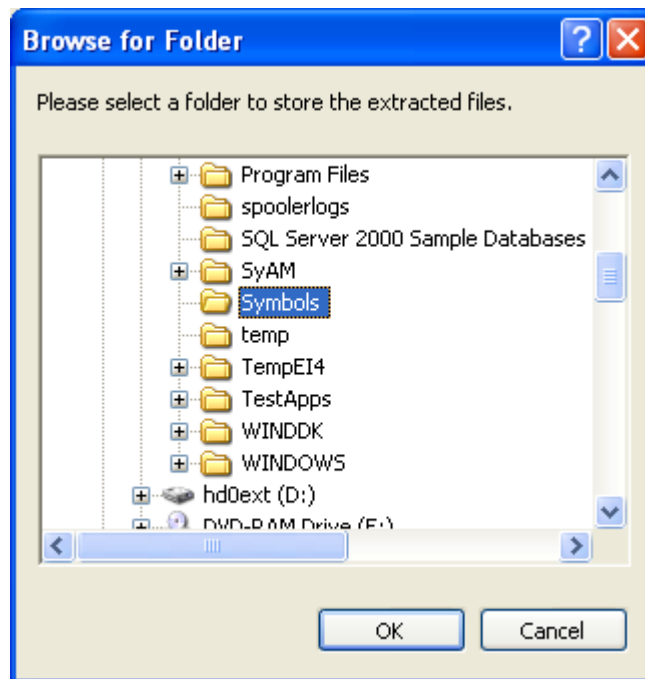


The Windows XP with Service Pack 2 retail symbols License agreement

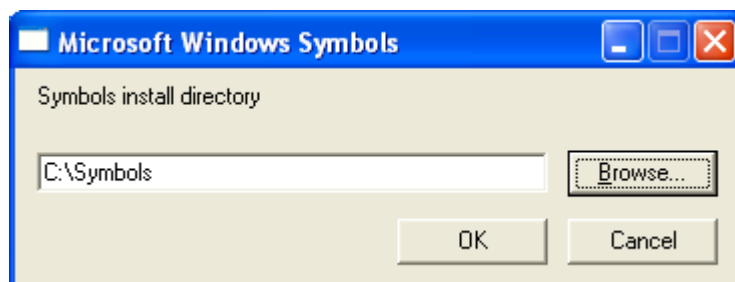
The default installation path was given for us. If you want to change the path, please use the Browse button.



The Windows XP with Service Pack 2 retail symbols default installation path

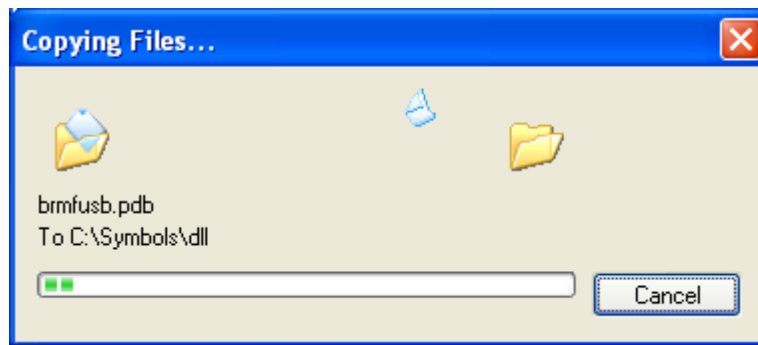


The Windows XP with Service Pack 2 retail symbols custom directory installation selection



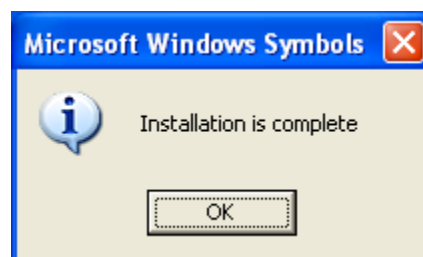
The Windows XP with Service Pack 2 retail symbols package installation path

The symbol package installation should start shortly.

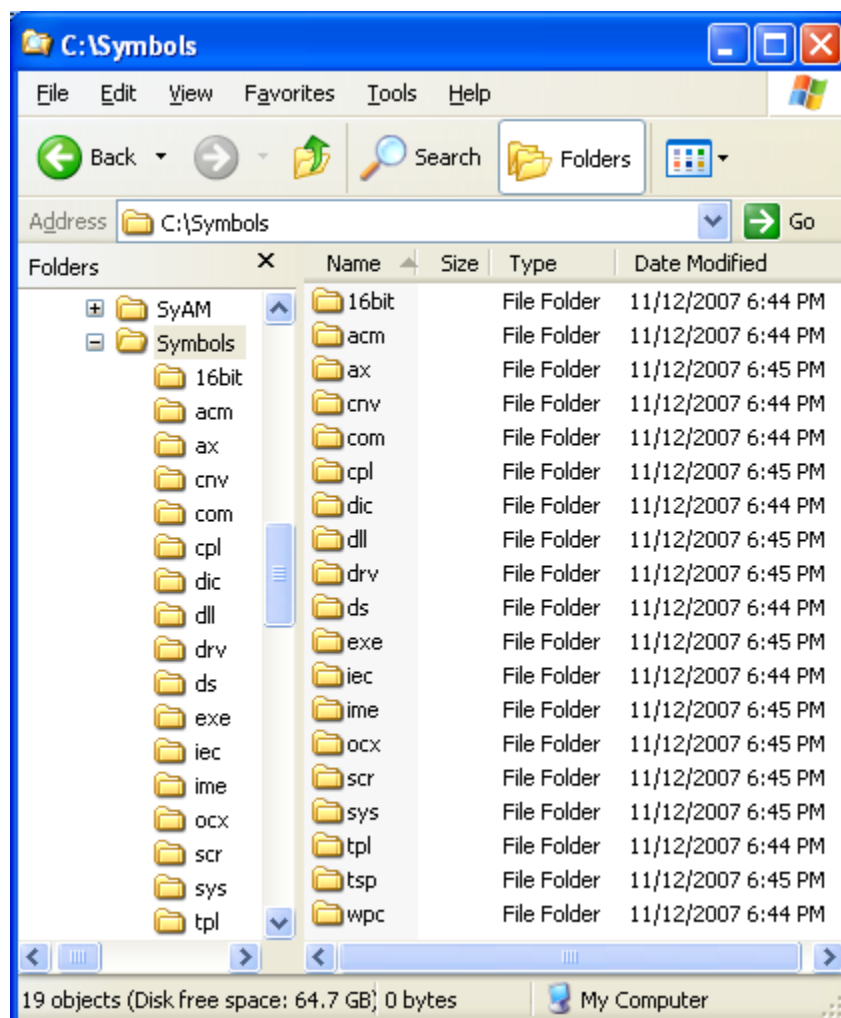


The Windows XP with Service Pack 2 retail symbols files copying in action

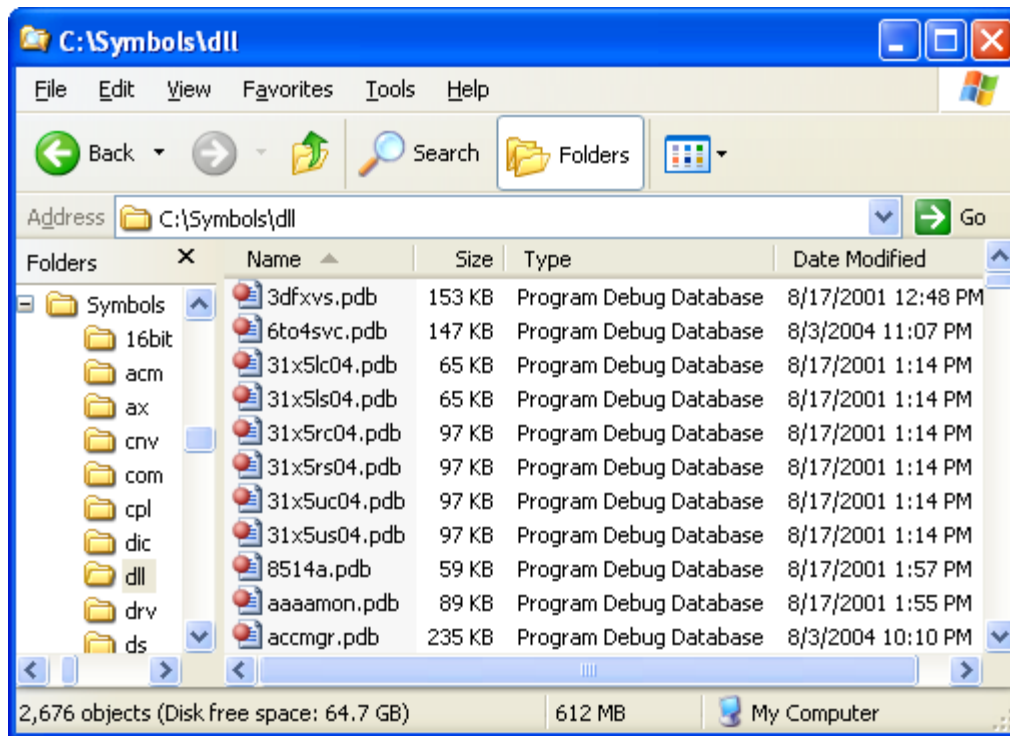
If there is no error, you should be welcomed by the following Windows message.



The Windows XP with Service Pack 2 retail symbols package installation is complete



The Windows XP with Service Pack 2 retail symbols installed directories



The Windows XP with Service Pack 2 retail symbols PDB files

## The WinDbg

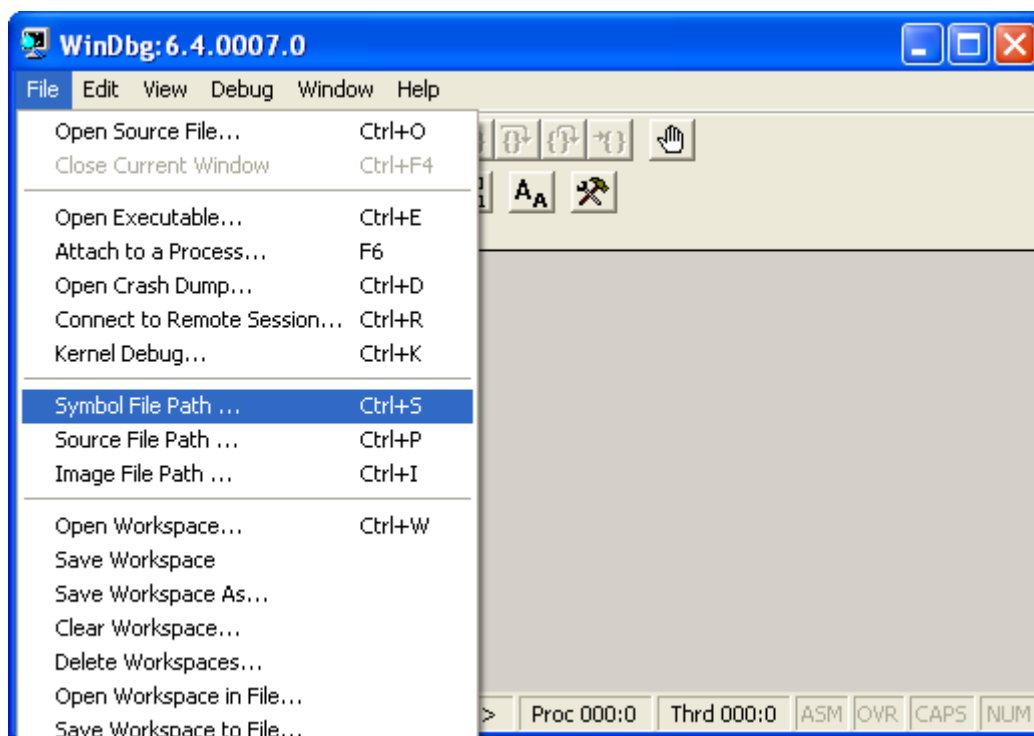
After executing windbg, you need to set the path to the symbol files. This path points to the directories with the pdb files of your drivers. You can have different directories by separating them with a semicolon (;). You also need to point to the corresponding PDB files of all the windows components, if you want the call stacks that you'll see to include the functions from the components that are developed by Microsoft.

However, the problem in this case is that the windows PDB files change between service packs, hotfixes, etc. Fortunately, Microsoft has configured a symbol server, which can be used to download the needed files on-demand. This means that you just set the symbol path to the symbol server and windbg downloads only the PDB files that it needs. In order to do this, you need to add an entry to the windbg symbol path that's equal to:

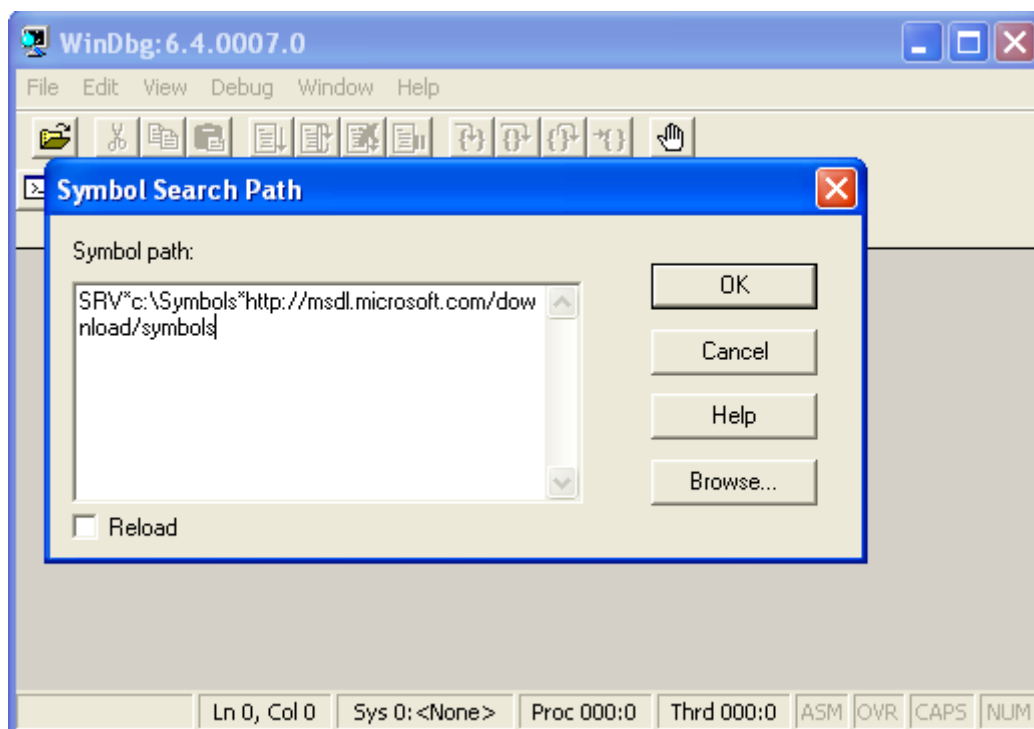
```
SRV*DownstreamStore*http://msdl.microsoft.com/download/symbols
```

In the above line, `http://msdl.microsoft.com/download/symbols` is the symbol server (you don't need to modify this), and `DownstreamStore` is the path, where you want the pdb files to be downloaded. This needs to be substituted by a local directory, e.g. `C:\Symbols`, so the complete entry would be:

```
SRV*c:\Symbols*http://msdl.microsoft.com/download/symbols
```



Setting the Symbol File Path



Setting the PDB Symbol File Path for the on demand download

Finally, if you have additional pdb files for the drivers that you are developing in directories `C:\mydrivers1`, `C:\mydrivers1\misc` and `C:\mydrivers2`, the complete symbol path would be:

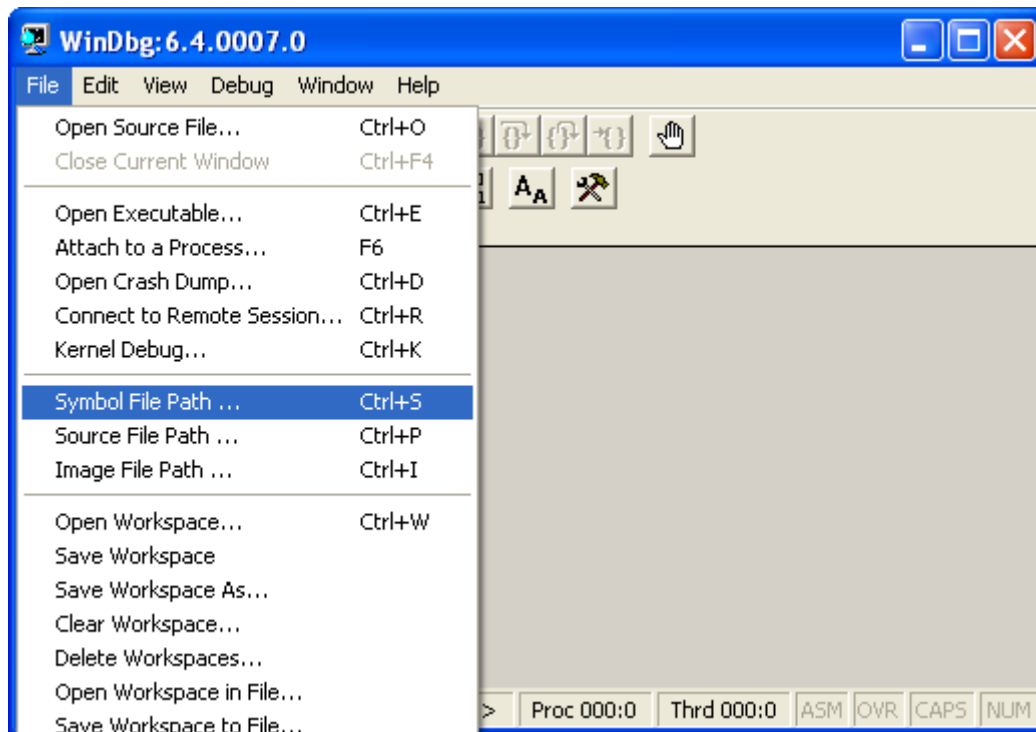
```
SRV*c:\websymbols*http://msdl.microsoft.com/download/symbols;c:\drivers1;c:\drivers1\misc;d:\drivers2
```

Also, as it can be seen from the above example, the directories in the path aren't recursive, so if you have PDB files both in `C:\mydrivers1` and `C:\mydrivers1\misc`, then you need to include both of them, since the format doesn't imply the latter.

In order to set this line, you need to open windbg, go to File → Symbol File Path and paste the line in the text area.

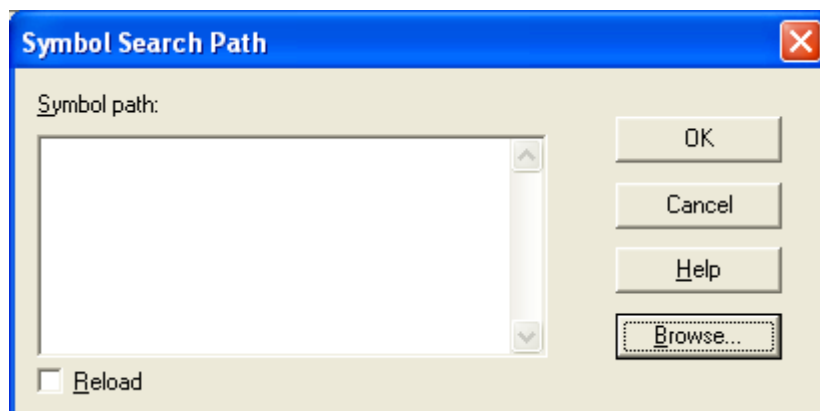
In our case the PDB files already installed under the `C:\Symbols` directory, so we just point to the directory in the Symbol File Path.

Launch the Windbg program.

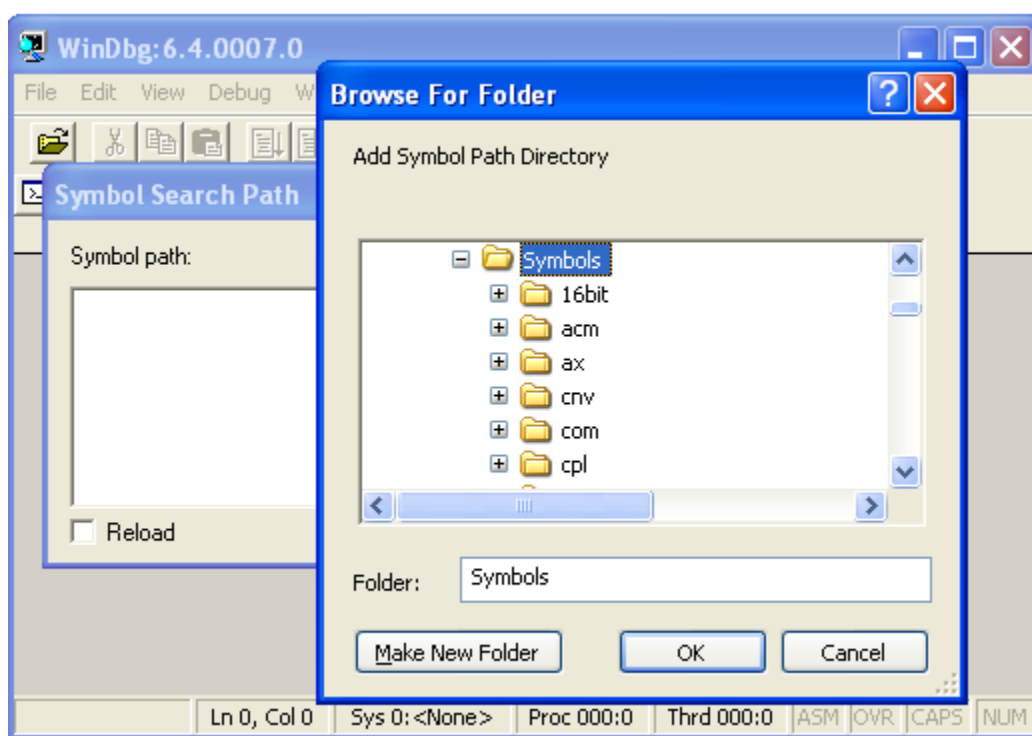


Invoking the Symbol File Path setting page

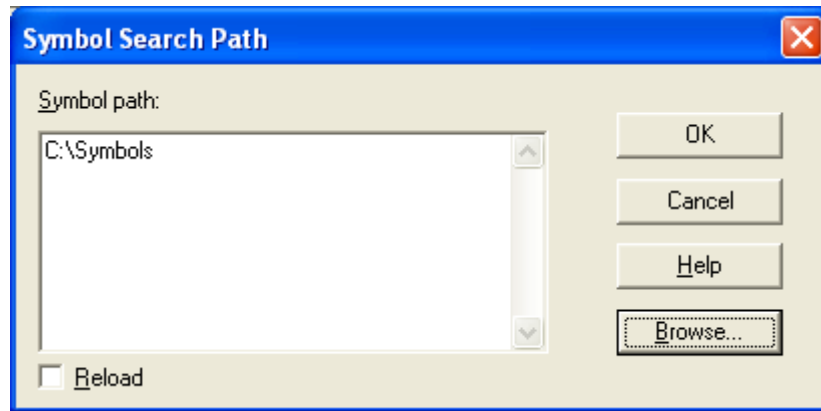




The Symbol File Path setting page

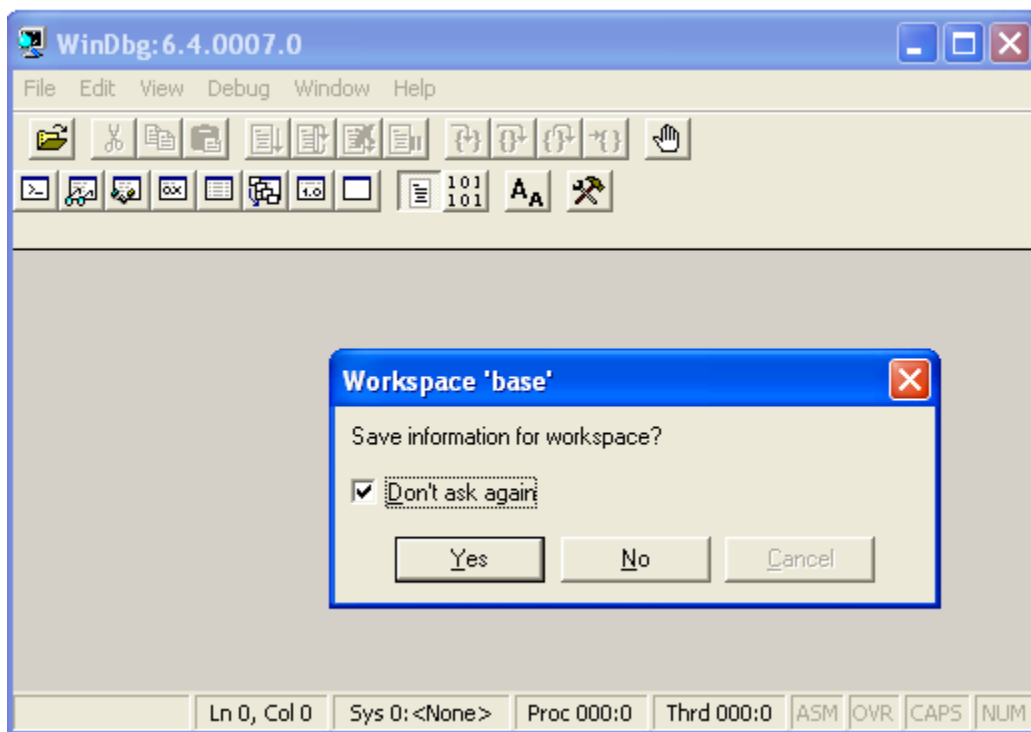


Selecting the `C:\Symbols` as the Symbol File Path



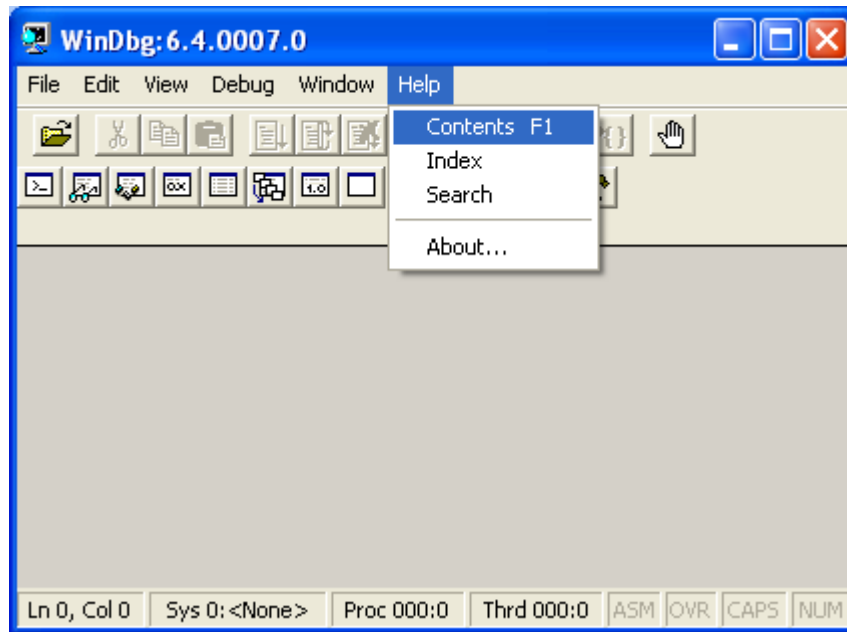
The Symbol File Path has been set to the local `C:\Symbols` directory

Don't forget to save the changes that have been done.

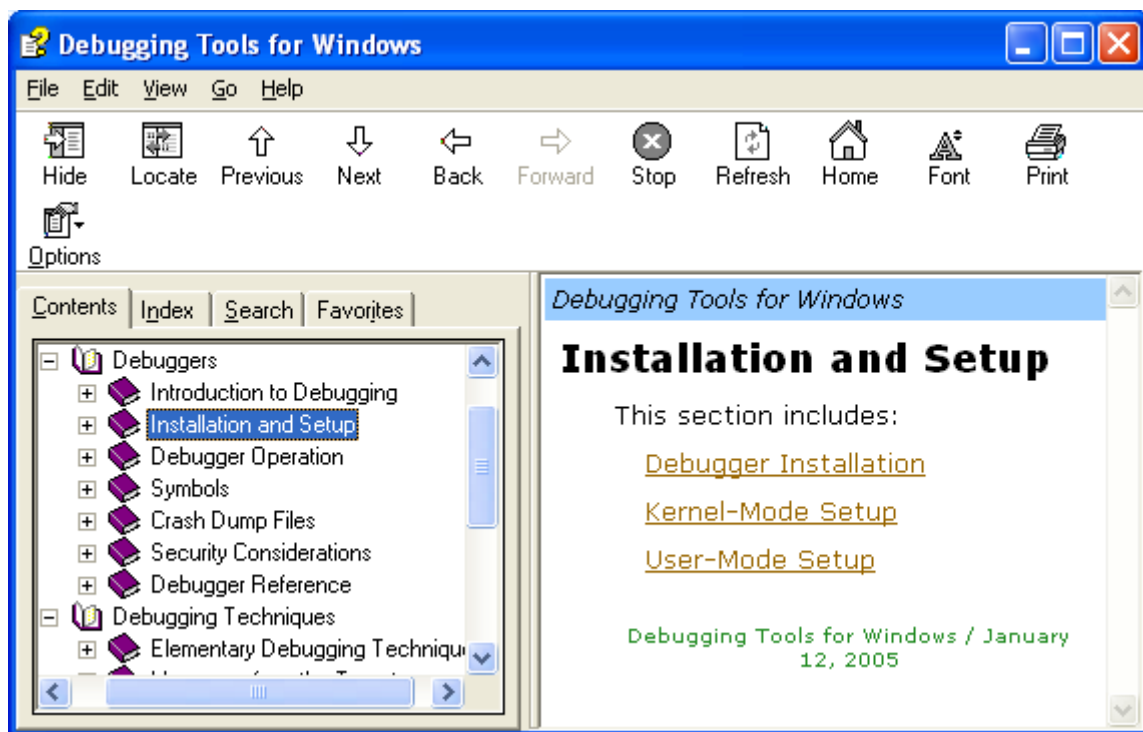


Saving the WinDbg changed workspace

More complete information should be found in the Help.



Invoking the WinDbg Help



The WinDbg Help

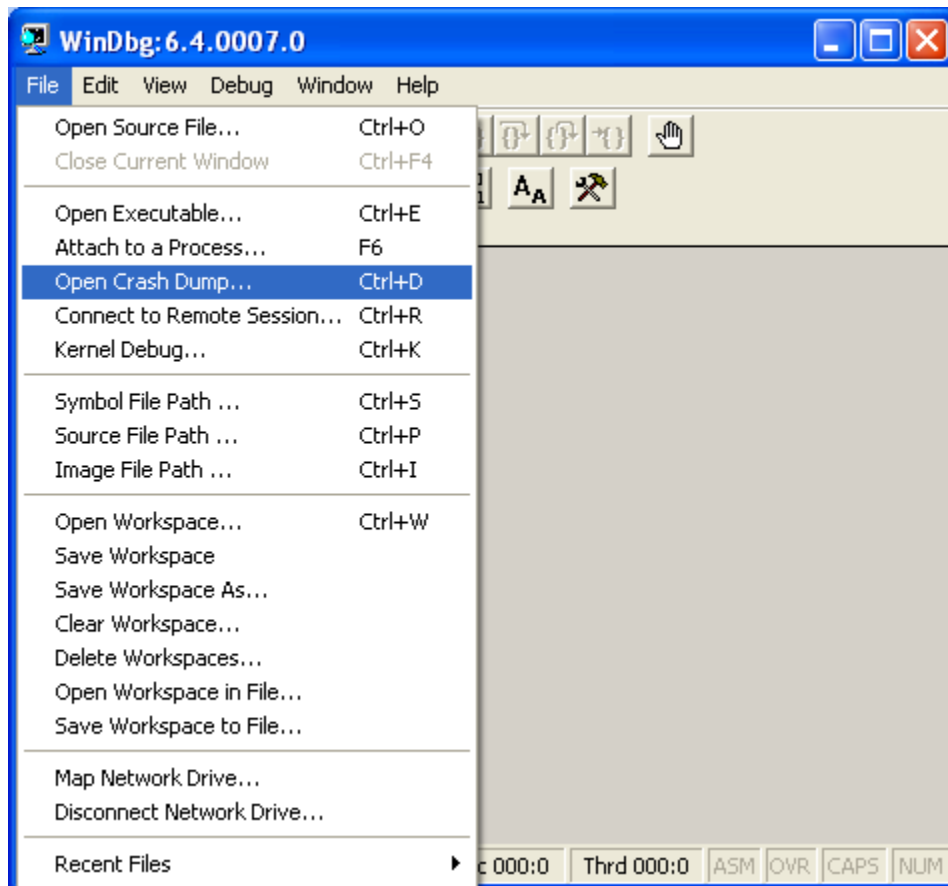
Kernel-mode memory dump files can be analyzed by WinDbg. The processor or Windows version that the dump file was created on does not need to match the platform on which KD is being run.

## Starting WinDbg

To analyze a dump file, start WinDbg with the `-z` command-line option:

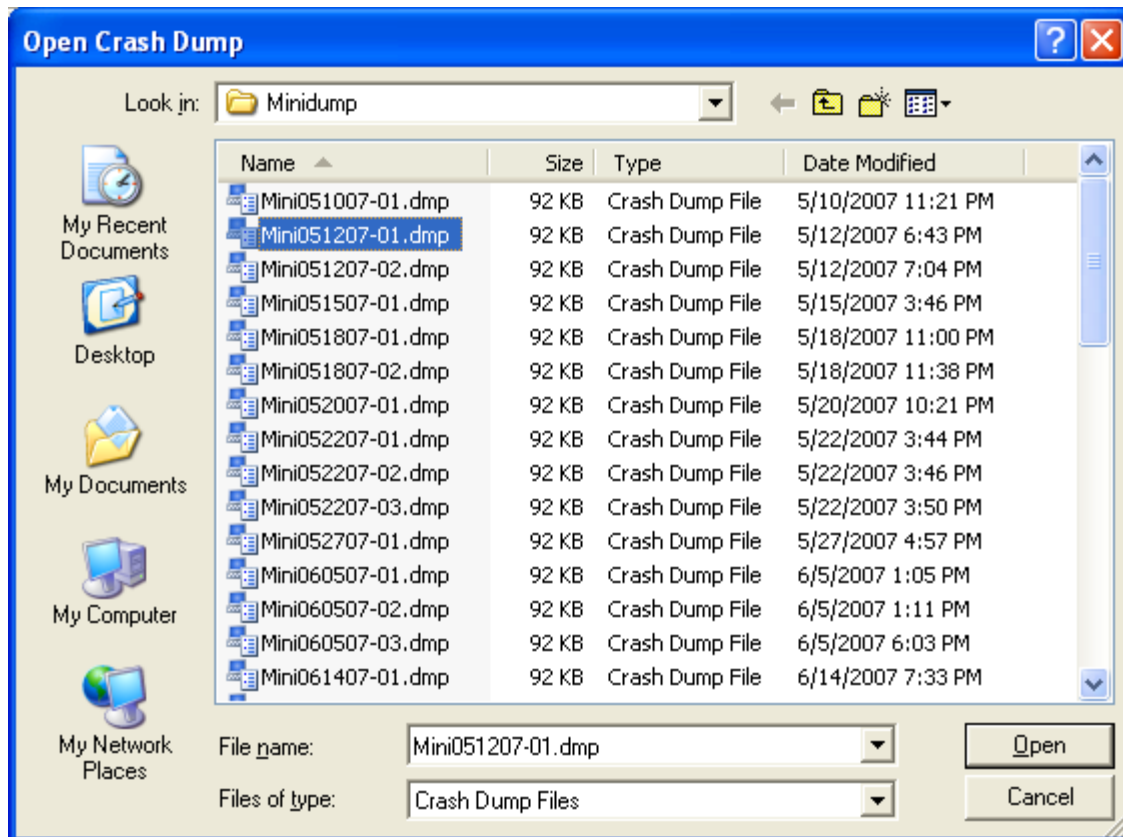
```
windbg -y SymbolPath -i ImagePath -z DumpFileName
```

The `-v` option (verbose mode) is also useful. If WinDbg is already running and is in dormant mode, you can open a crash dump by selecting the `File → Open Crash Dump` menu command or pressing the `CTRL+D` shortcut key.



### Opening the Windows Crash Dump file

When the `Open Crash Dump` dialog box appears, enter the full path and name of the crash dump file in the `File name` text box, or use the dialog box to select the proper path and file name. When the proper file has been chosen, click `Open`.



Selecting the Windows mini Crash Dump file

```

Command - Dump C:\WINDOWS\Minidump\Mini031009-02.dmp - WinDbg
Loading User Symbols
Loading unloaded module list
...
*** WARNING: Unable to verify timestamp for hal.dll
*****
*
*                               Bugcheck Analysis
*
*****

Use !analyze -v to get detailed debugging information.

BugCheck 1000000A, {ffdf, 2, 1, 806e4a8e}

Unable to load image KS-959.sys, Win32 error 0n2
*** WARNING: Unable to verify timestamp for KS-959.sys
*** ERROR: Module load completed but symbols could not be loaded for
Probably caused by : KS-959.sys ( KS_959+b34 )

Followup: MachineOwner
-----

0: kd> !analyze -v
*****
*
*                               Bugcheck Analysis
*
*****

0: kd>

```

You can also open a dump file after the debugger is running by using the `.opendump` (Open Dump File) command, followed with `g` (Go). Dump files generally end with the extension `.dmp` or `.mdmp`. You can use network shares or Universal Naming Convention (UNC) file names for the memory dump file. Well, it will take a long story to provide examples on how to debug either the user mode or kernel mode and why not you try the following links by Windows device driver developer for more information.

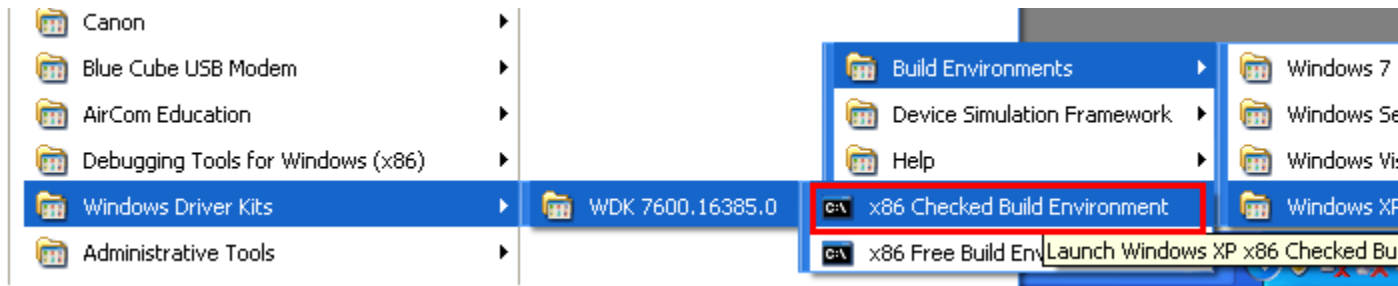
1. [A Crash Dump analysis tutorials.](#)
2. [Windbg basic tutorials.](#)
3. [Tips on how to analyze strange Crash Dumps and uninstall the Windows hidden drivers.](#)

## Verifying WDK Installation

To verify that you have properly installed the Windows Driver Kit (WDK), complete the following steps:

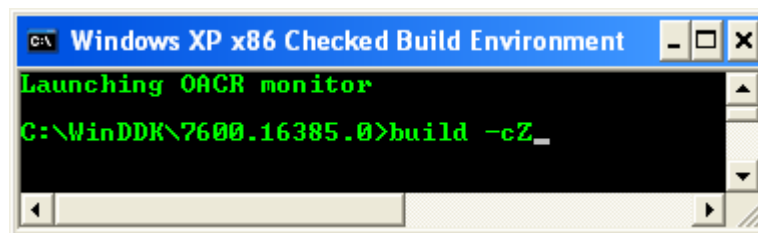
1. Start a Windows XP Checked Build Environment Command Prompt window:

- a. Click the **Start** button, and then click **All Programs**.
- b. Point to **Windows Driver Kits**, point to your selected **WDK XXXX**, point to **Build Environments**, and then point to **Windows XP** (or your chosen Windows OS).
- c. Click **Windows XP Checked Build Environment** to create the build environment for the chosen Windows platform.

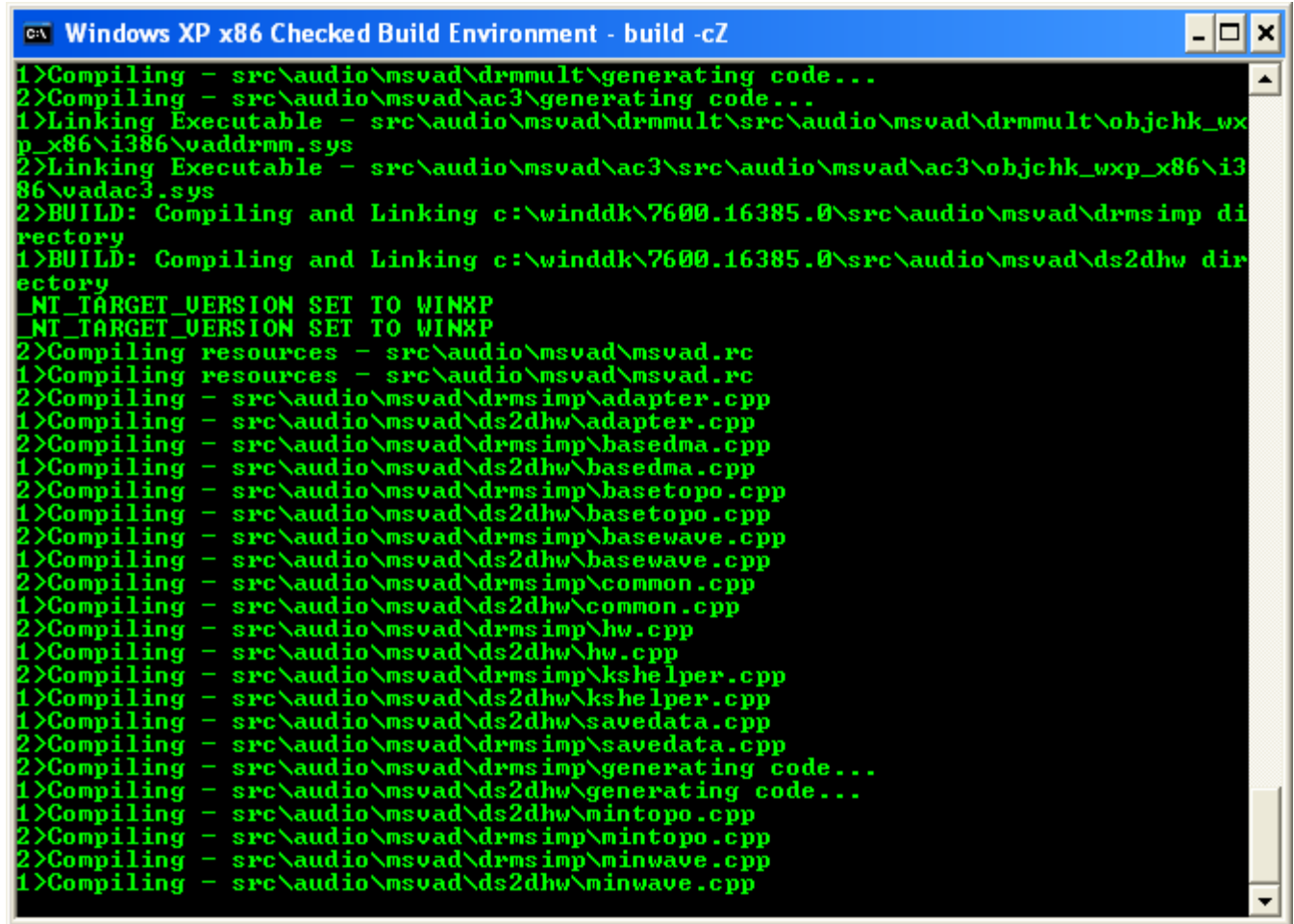


### Running the Checked Build Environment for Windows XP

2. At the command prompt, type **build -cZ** to compile and link the complete set of installed sources. This command can take more than 30 minutes to complete, depending on which WDK components are installed and the system on which the build is being performed.



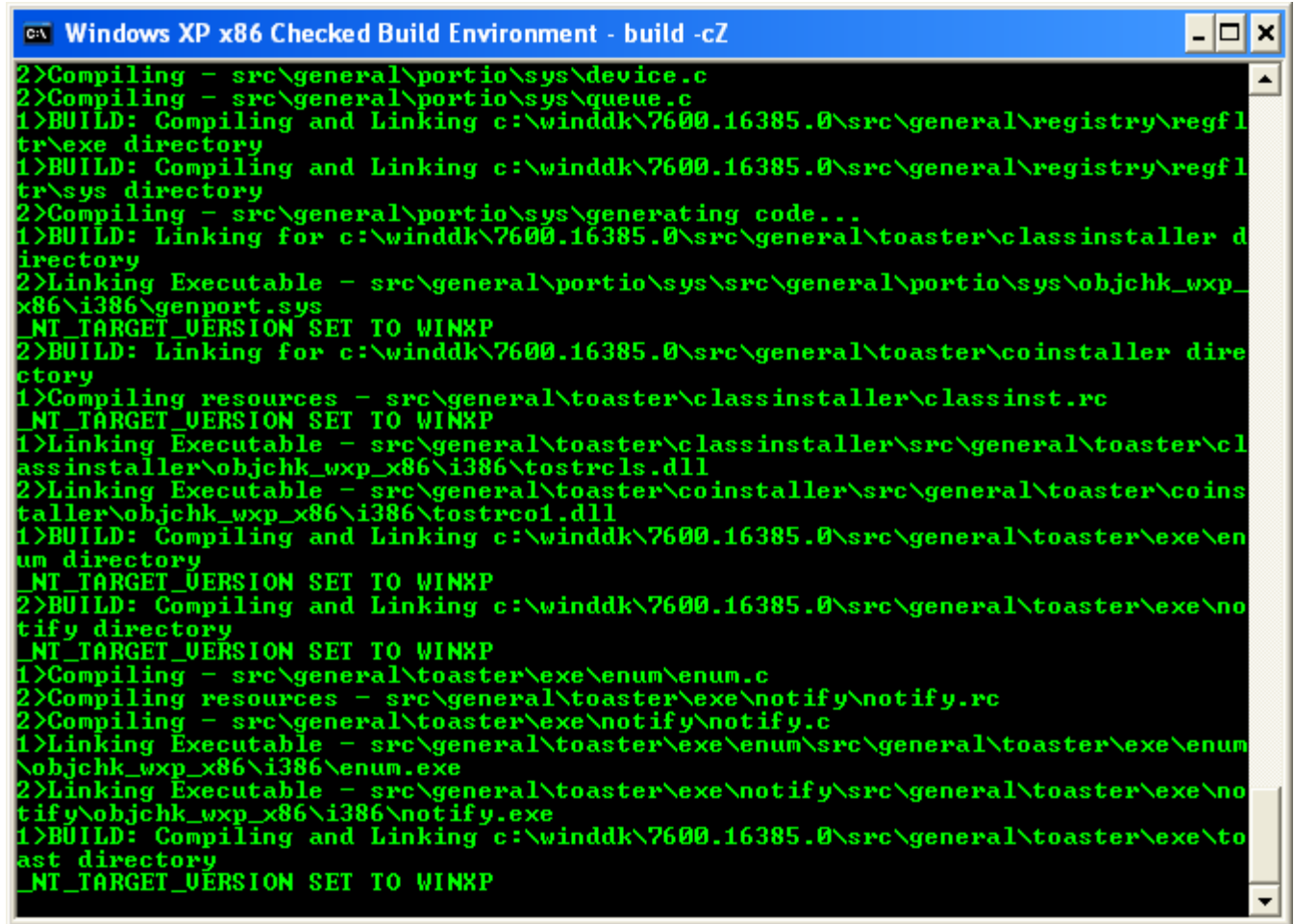
Running the **build** command



```
C:\ Windows XP x86 Checked Build Environment - build -cZ
1>Compiling - src\audio\msvad\drmmult\generating code...
2>Compiling - src\audio\msvad\ac3\generating code...
1>Linking Executable - src\audio\msvad\drmmult\src\audio\msvad\drmmult\objchk_wx
p_x86\i386\vaddrmm.sys
2>Linking Executable - src\audio\msvad\ac3\src\audio\msvad\ac3\objchk_wxp_x86\i3
86\vadac3.sys
2>BUILD: Compiling and Linking c:\winddk\7600.16385.0\src\audio\msvad\drmsimp di
rectory
1>BUILD: Compiling and Linking c:\winddk\7600.16385.0\src\audio\msvad\ds2dhw dir
ectory
_NT_TARGET_VERSION SET TO WINXP
_NT_TARGET_VERSION SET TO WINXP
2>Compiling resources - src\audio\msvad\msvad.rc
1>Compiling resources - src\audio\msvad\msvad.rc
2>Compiling - src\audio\msvad\drmsimp\adapter.cpp
1>Compiling - src\audio\msvad\ds2dhw\adapter.cpp
2>Compiling - src\audio\msvad\drmsimp\basedma.cpp
1>Compiling - src\audio\msvad\ds2dhw\basedma.cpp
2>Compiling - src\audio\msvad\drmsimp\basetopo.cpp
1>Compiling - src\audio\msvad\ds2dhw\basetopo.cpp
2>Compiling - src\audio\msvad\drmsimp\basewave.cpp
1>Compiling - src\audio\msvad\ds2dhw\basewave.cpp
2>Compiling - src\audio\msvad\drmsimp\common.cpp
1>Compiling - src\audio\msvad\ds2dhw\common.cpp
2>Compiling - src\audio\msvad\drmsimp\hw.cpp
1>Compiling - src\audio\msvad\ds2dhw\hw.cpp
2>Compiling - src\audio\msvad\drmsimp\kshelper.cpp
1>Compiling - src\audio\msvad\ds2dhw\kshelper.cpp
1>Compiling - src\audio\msvad\ds2dhw\savedata.cpp
2>Compiling - src\audio\msvad\drmsimp\savedata.cpp
2>Compiling - src\audio\msvad\drmsimp\generating code...
1>Compiling - src\audio\msvad\ds2dhw\generating code...
1>Compiling - src\audio\msvad\ds2dhw\mintopo.cpp
2>Compiling - src\audio\msvad\drmsimp\mintopo.cpp
2>Compiling - src\audio\msvad\drmsimp\minwave.cpp
1>Compiling - src\audio\msvad\ds2dhw\minwave.cpp
```

The Checked Build Environment for Windows XP in action





```
C:\ Windows XP x86 Checked Build Environment - build -cZ
2>Compiling - src\general\portio\sys\device.c
2>Compiling - src\general\portio\sys\queue.c
1>BUILD: Compiling and Linking c:\winddk\7600.16385.0\src\general\registry\regfltr\exe directory
1>BUILD: Compiling and Linking c:\winddk\7600.16385.0\src\general\registry\regfltr\sys directory
2>Compiling - src\general\portio\sys\generating code...
1>BUILD: Linking for c:\winddk\7600.16385.0\src\general\toaster\classinstaller directory
2>Linking Executable - src\general\portio\sys\src\general\portio\sys\objchk_wxp_x86\i386\genport.sys
_NT_TARGET_VERSION SET TO WINXP
2>BUILD: Linking for c:\winddk\7600.16385.0\src\general\toaster\coinstaller directory
1>Compiling resources - src\general\toaster\classinstaller\classinst.rc
_NT_TARGET_VERSION SET TO WINXP
1>Linking Executable - src\general\toaster\classinstaller\src\general\toaster\classinstaller\objchk_wxp_x86\i386\tostrcls.dll
2>Linking Executable - src\general\toaster\coinstaller\src\general\toaster\coinstaller\objchk_wxp_x86\i386\tostrcol.dll
1>BUILD: Compiling and Linking c:\winddk\7600.16385.0\src\general\toaster\exe\enum directory
_NT_TARGET_VERSION SET TO WINXP
2>BUILD: Compiling and Linking c:\winddk\7600.16385.0\src\general\toaster\exe\notify directory
_NT_TARGET_VERSION SET TO WINXP
1>Compiling - src\general\toaster\exe\enum\enum.c
2>Compiling resources - src\general\toaster\exe\notify\notify.rc
2>Compiling - src\general\toaster\exe\notify\notify.c
1>Linking Executable - src\general\toaster\exe\enum\src\general\toaster\exe\enum\objchk_wxp_x86\i386\enum.exe
2>Linking Executable - src\general\toaster\exe\notify\src\general\toaster\exe\notify\objchk_wxp_x86\i386\notify.exe
1>BUILD: Compiling and Linking c:\winddk\7600.16385.0\src\general\toaster\exe\toast directory
_NT_TARGET_VERSION SET TO WINXP
```

More action for the Checked Build Environment for Windows XP

3. The build command should complete with no errors and/or warnings displayed as shown below.

```

C:\ Windows XP x86 Checked Build Environment
2>Linking Executable - src\wmi\wmiacpi\src\wmi\wmiacpi\objchk_wxp_x86\i386\acpiim
of.dll
2>BUILD: Linking for c:\winddk\7600.16385.0\src\wmi\wmisamp directory
_NT_TARGET_VERSION SET TO WINXP
1>BUILD: Linking for c:\winddk\7600.16385.0\src\wpd\wpdbasichardwaredriver direc
tory
_NT_TARGET_VERSION SET TO WINXP
2>Linking Executable - src\wmi\wmisamp\src\wmi\wmisamp\objchk_wxp_x86\i386\wmisa
mp.sys
1>Compiling resources - src\wpd\wpdbasichardwaredriver\wpdbasichardwaredriver.rc
2>BUILD: Linking for c:\winddk\7600.16385.0\src\wpd\wpdhelloworlddriver director
y
1>Linking Executable - src\wpd\wpdbasichardwaredriver\src\wpd\wpdbasichardwaredr
iver\objchk_wxp_x86\i386\wpdbasichardwaredriver.dll
_NT_TARGET_VERSION SET TO WINXP
2>Compiling resources - src\wpd\wpdhelloworlddriver\wpdhelloworlddriver.rc
2>Linking Executable - src\wpd\wpdhelloworlddriver\src\wpd\wpdhelloworlddriver\o
bjchk_wxp_x86\i386\wpdhelloworlddriver.dll
1>BUILD: Linking for c:\winddk\7600.16385.0\src\wpd\wpdmultitransportdriver direc
tory
2>BUILD: Linking for c:\winddk\7600.16385.0\src\wpd\wpdservicesampledriver direc
tory
1>BUILD: Linking for c:\winddk\7600.16385.0\src\wpd\wpdwudfsampledriver director
y
_NT_TARGET_VERSION SET TO WINXP
1>Compiling resources - src\wpd\wpdwudfsampledriver\wpdwudfsampledriver.rc
1>Linking Executable - src\wpd\wpdwudfsampledriver\src\wpd\wpdwudfsampledriver\o
bjchk_wxp_x86\i386\wpdwudfsampledriver.dll
BUILD: Finish time: Sun Jan 03 12:09:41 2010
BUILD: Done

1,588 files compiled
77 libraries built
222 executables built
C:\WinDDK\7600.16385.0>

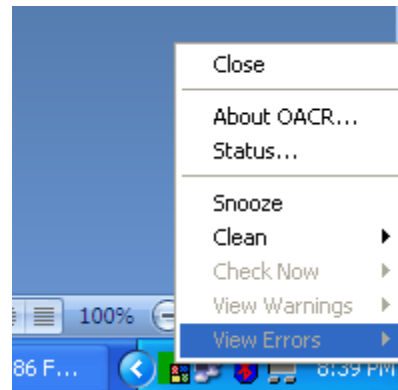
```

The Checked Build Environment for Windows XP is completed without any error.

By invoking the PREFast, you can see the warnings and errors in details (if any).



The warnings and error can be seen by invoking the menus by right-clicking the icon tray.



**OACR Warnings - WDKSamples:x86chk**

**PREfast** 9.0.21022.90713 **Message List** Filter

#	Description	Warning	Source Path	Source Location	In Function
1	Bitwise operation on logica...	6291	c:\winddk\760...	busenum.c(73)	DriverEntry
2	Bitwise operation on logica...	6291	c:\winddk\760...	busenum.c(177)	Bus_CreateO
3	Bitwise operation on logica...	6291	c:\winddk\760...	busenum.c(182)	Bus_CreateO
4	Bitwise operation on logica...	6291	c:\winddk\760...	busenum.c(276)	Bus_IoCtl
5	Bitwise operation on logica...	6291	c:\winddk\760...	busenum.c(290)	Bus_IoCtl
6	Bitwise operation on logica...	6291	c:\winddk\760...	busenum.c(303)	Bus_IoCtl
7	Bitwise operation on logica...	6291	c:\winddk\760...	busenum.c(344)	Bus_DriverU
8	Bitwise operation on logica...	6291	c:\winddk\760...	buspdo.c(258)	Bus_PDO_Pr
9	Bitwise operation on logica...	6291	c:\winddk\760...	buspdo.c(268)	Bus_PDO_Pr
10	Bitwise operation on logica...	6291	c:\winddk\760...	buspdo.c(457)	Bus_PDO_Qu
11	Bitwise operation on logica...	6291	c:\winddk\760...	buspdo.c(673)	Bus_PDO_Qu
12	Bitwise operation on logica...	6291	c:\winddk\760...	buspdo.c(806)	Bus_PDO_Qu
13	Bitwise operation on logica...	6291	c:\winddk\760...	buspdo.c(818)	Bus_PDO_Qu
14	Bitwise operation on logica...	6291	c:\winddk\760...	buspdo.c(1273)	Bus_GetCrisp
15	Bitwise operation on logica...	6291	c:\winddk\760...	buspdo.c(1304)	Bus_SetCrisp
16	Bitwise operation on logica...	6291	c:\winddk\760...	buspdo.c(1331)	Bus_IsSafet
17	Bitwise operation on logica...	6291	c:\winddk\760...	pnpc.c(82)	Bus_AddDev
18	Bitwise operation on logica...	6291	c:\winddk\760...	pnpc.c(171)	Bus_AddDev
19	Bitwise operation on logica...	6291	c:\winddk\760...	pnpc.c(207)	Bus_AddDev
20	Bitwise operation on logica...	6291	c:\winddk\760...	pnpc.c(216)	Bus_AddDev
21	Bitwise operation on logica...	6291	c:\winddk\760...	pnpc.c(230)	Bus_AddDev
22	Bitwise operation on logica...	6291	c:\winddk\760...	pnpc.c(235)	Bus_AddDev
23	Bitwise operation on logica...	6291	c:\winddk\760...	pnpc.c(307)	Bus PnP

The next tutorial, will try to demonstrate the steps on how to build the Windows driver using a given sample code.

## Steps on Using WDK and Building Windows Driver from Code Sample

Firstly create an empty directory to store your driver project in C drive. Use a simple path with no spaces in it, just because it is easier to navigate from the command-prompt. For example:

C:\MYDRIVERS\HELLO\

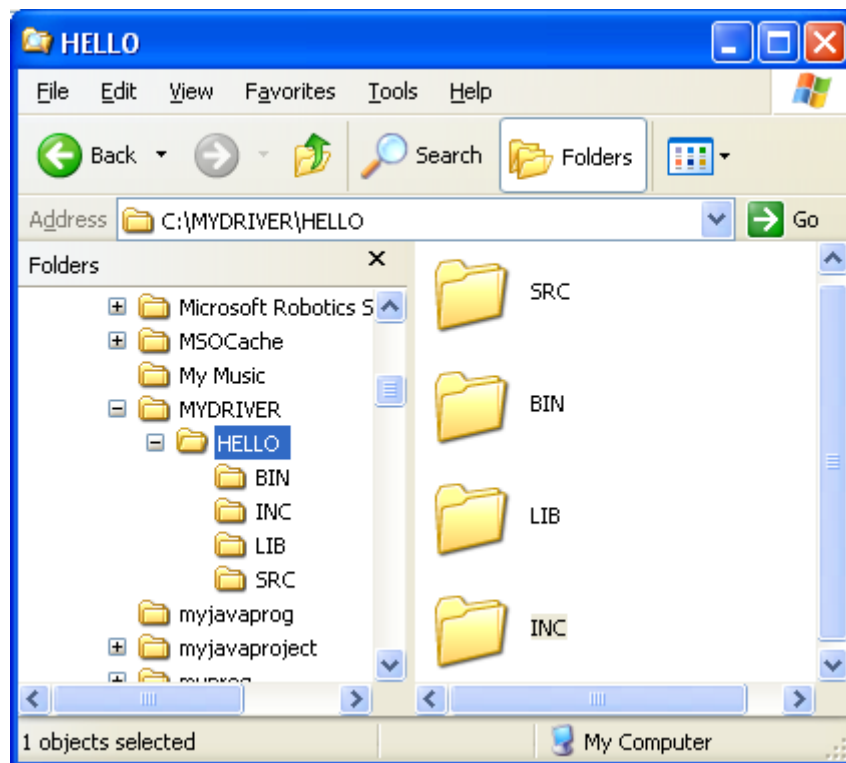
In the HELLO folder create the following subfolders for organizing our project.

SRC – for source code files

BIN – for binary files

LIB – for custom made or extra library files

INC – for include files



Create an empty text file in the SRC directory and call it "hello.c", then type the following code into it.

```
#include <ntddk.h>
```

```
NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING
RegistryPath)
{
    DbgPrint("Hello WDK World!\n");
    return STATUS_SUCCESS;
}
```

The DriverEntry() is a function required by the WDK - it is similar in concept to DllMain() because it is called when your driver first loads - however your driver remains loaded after you return back to the kernel.

The two parameters are worth mentioning. DriverObject is a pointer to a DRIVER\_OBJECT structure, a kernel data structure used to represent the loaded device driver. RegistryPath() is a Unicode string which holds the corresponding service entry location in the registry.

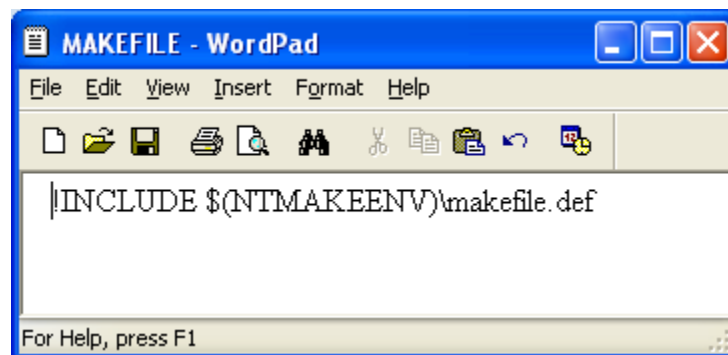
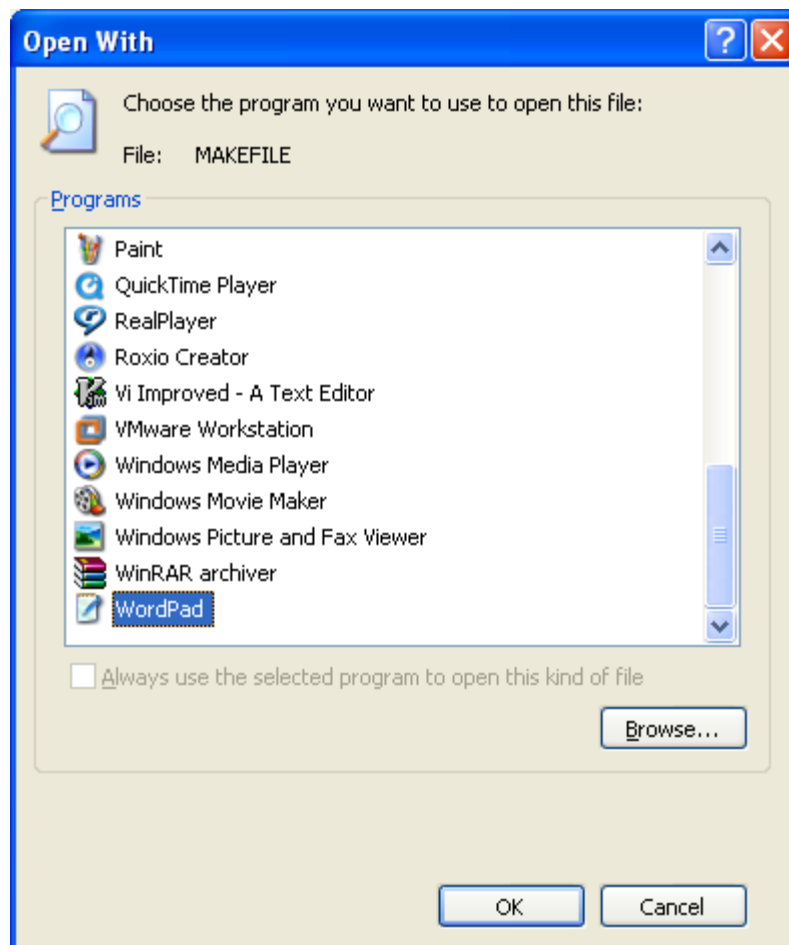
Unlike win32 programs which use BOOL return types, and TRUE and FALSE, kernel drivers use NTSTATUS return types. This is a 32bit value which can hold many different error codes and severity levels. The value STATUS\_SUCCESS (which is zero) is used to indicate a successful returns value. Other STATUS\_xxx codes can be found in the ntstatus.h header file. Note that, if we return anything other than STATUS\_SUCCESS from DriverEntry, our driver will fail to load

### **Building a Windows Driver Sample**

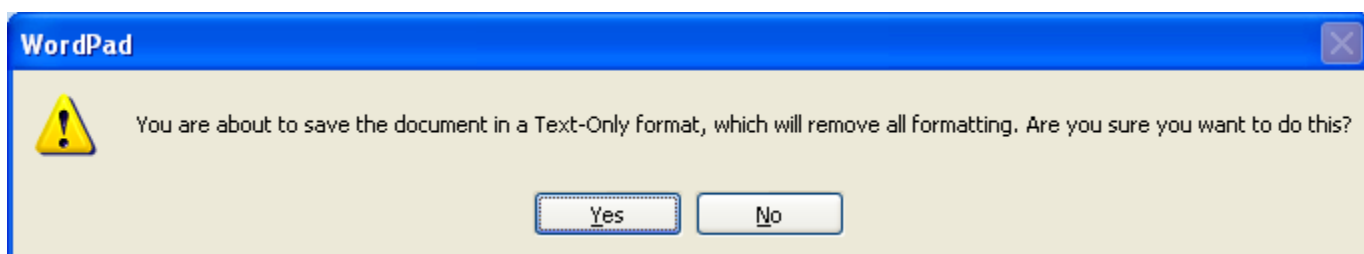
Assuming that you have created your project directory, you need to create two more files which the WDK build utility requires in order to build your project. The first file you need to create is called "MAKEFILE" - note that there is no extension on this filename. MAKEFILE should contain a single line (shown below):

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

and must never be edited in any way. Usually the WDK samples include such a file, so you can copy it straight from there. Use WordPad to edit the file.



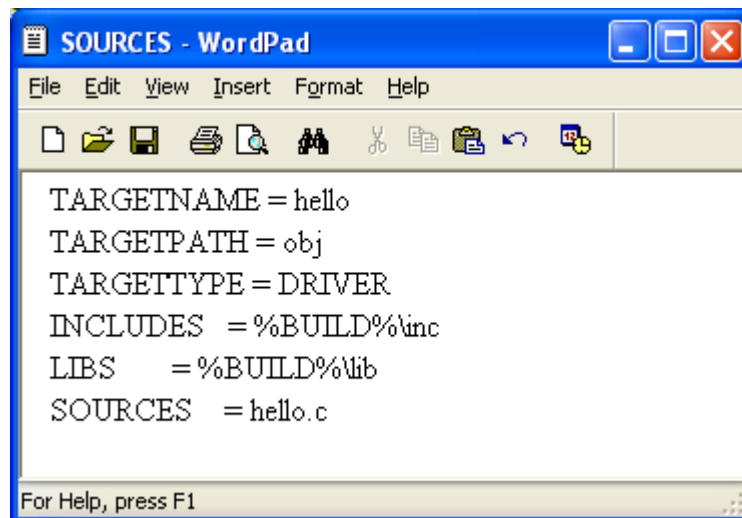
Don't forget to save the file and click Yes for the following warning splash screen if any.



The second file you need to create is your main project file, called "SOURCES", again with no file extension. The contents of this file direct the WDK build utility to your source code files.

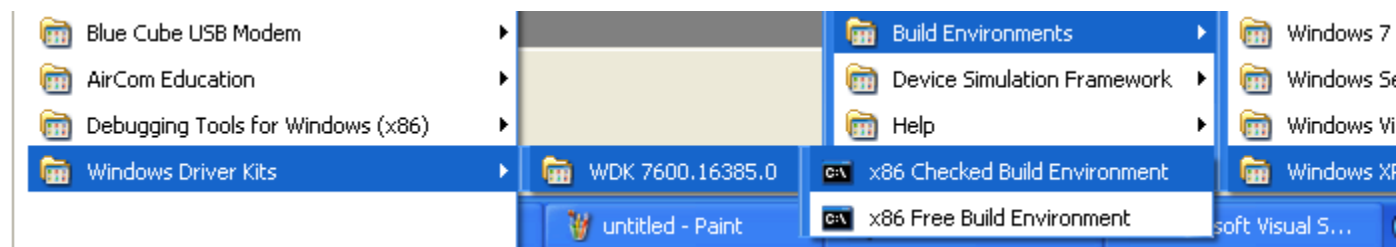
SOURCES contains two important pieces of information - the name of your driver, and a list of source-files. If you have more than one source-file, then they should be listed one-after-the-other on the same line, separated by a space. You have to be careful with filenames that contain spaces (i.e. use quotes), so it is easiest just to use simple names. Put the following codes in the SOURCES file and save it.

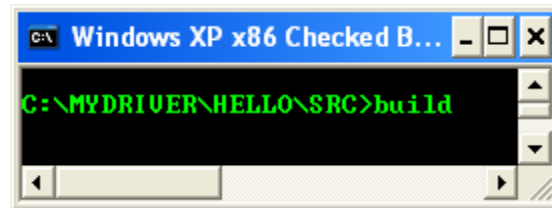
```
TARGETNAME = hello
TARGETPATH = obj
TARGETTYPE = DRIVER
INCLUDES    = %BUILD%\inc
LIBS        = %BUILD%\lib
SOURCES     = hello.c
```



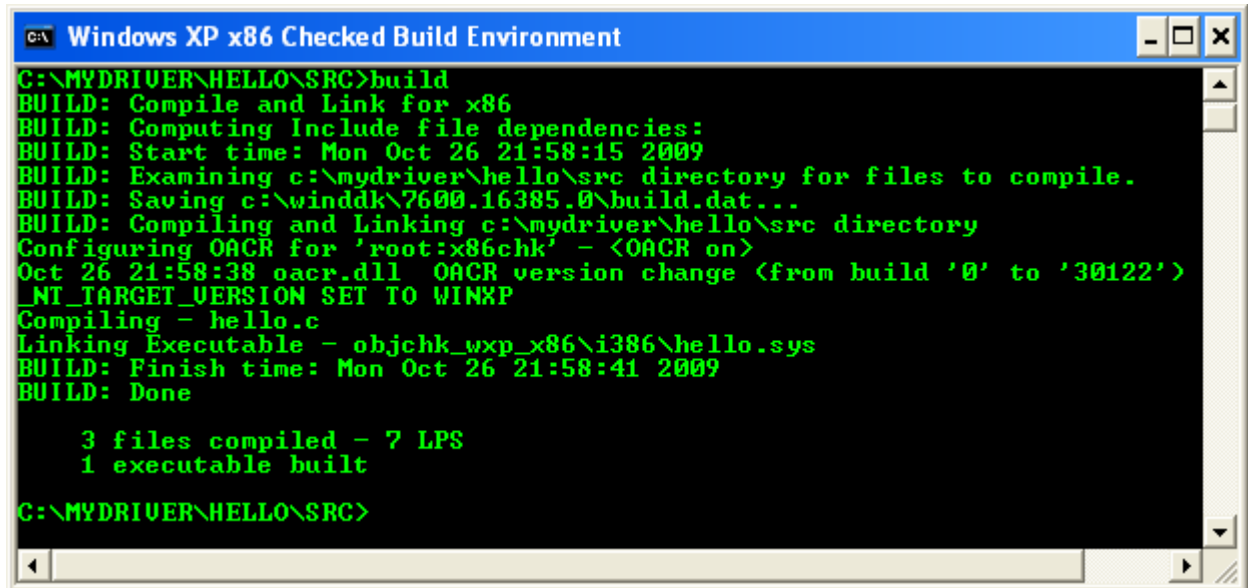
The SOURCES file template can be found at [SOURCES file template](#).

Building a driver is very simple once you have created a driver project. At this point you should have three files in your project directory - hello.c, SOURCES and MAKEFILE. Open up the x86 Checked Build Environment for Windows XP (the WDK command-prompt) and go to the project SRC folder and type "build".





```
C:\MYDRIVER\HELLO\SRC>build
```



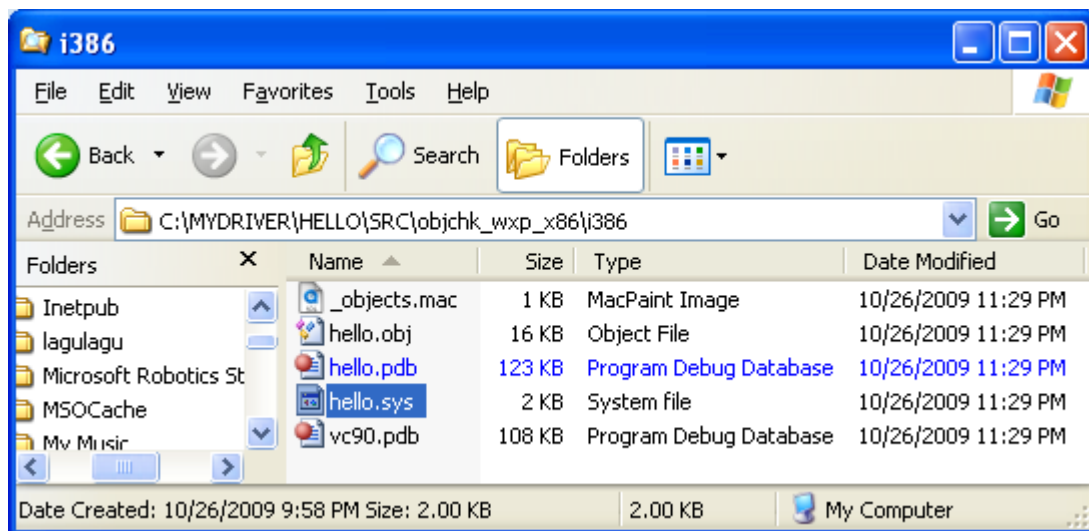
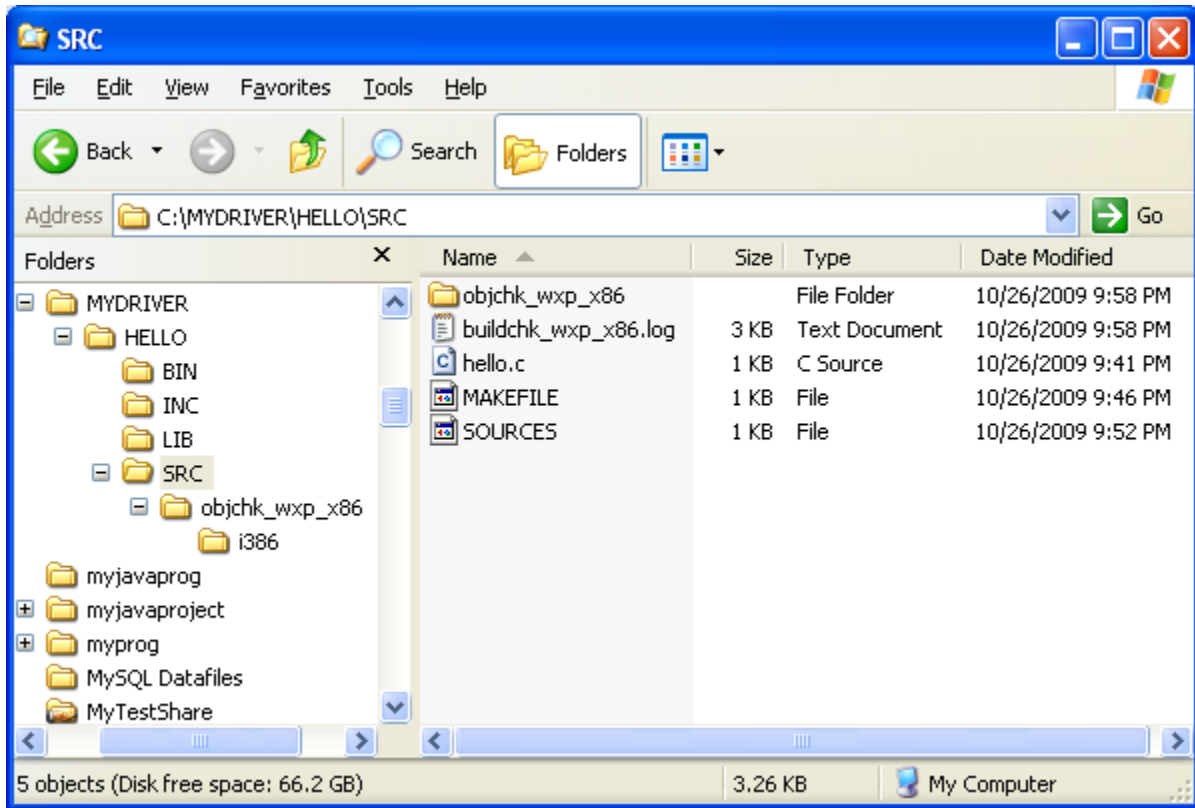
```
C:\MYDRIVER\HELLO\SRC>build
BUILD: Compile and Link for x86
BUILD: Computing Include file dependencies:
BUILD: Start time: Mon Oct 26 21:58:15 2009
BUILD: Examining c:\mydriver\hello\src directory for files to compile.
BUILD: Saving c:\winddk\7600.16385.0\build.dat...
BUILD: Compiling and Linking c:\mydriver\hello\src directory
Configuring OACR for 'root:x86chk' - <OACR on>
Oct 26 21:58:38 oacr.dll OACR version change <from build '0' to '30122'>
_NT_TARGET_VERSION SET TO WINXP
Compiling - hello.c
Linking Executable - objchk_wxp_x86\i386\hello.sys
BUILD: Finish time: Mon Oct 26 21:58:41 2009
BUILD: Done

    3 files compiled - 7 LPS
    1 executable built

C:\MYDRIVER\HELLO\SRC>
```

The following Figure shows the created folders and files after the building steps were completed successfully.





## Drivers and Services

Installing and executing a driver is quite a bit different than running a "normal" program from the command line. There is no concept of running a new process because a driver is a kernel module which lives permanently inside the system. Two steps are required to get a driver installed:

1. Registering the driver as a system service, and then

## 2. Starting the driver.

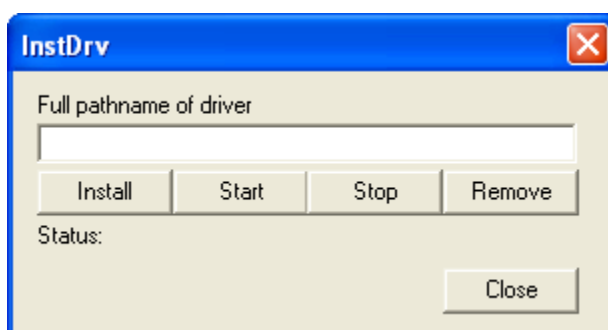
This is usually the area that confuses people who are not used to working with drivers. A driver is treated by Windows as a regular service which can be started and stopped just like any other service. The Windows component that controls all this is called the Service Control Manager (SCM) which you have learned in the other topic (Windows Service). This component exposes an API which can be used to register, unregister, and start & stop drivers and regular Win32 services.

Actually registering a driver and starting it can be a little confusing if you've not done it before. However for beginners, there are several tools that can be used to automate these processes. For example:

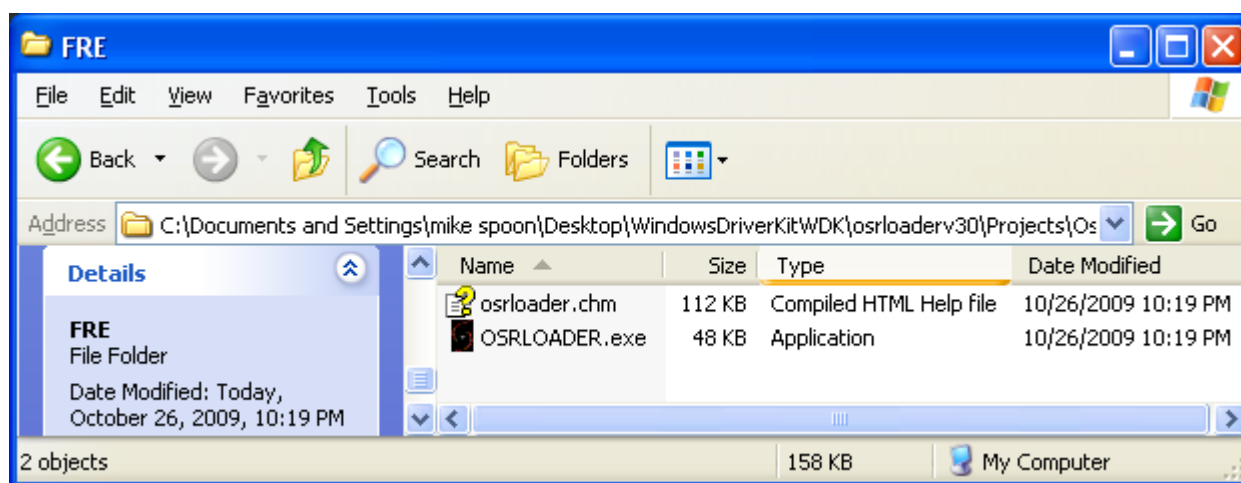
<http://www.osronline.com/> - osrloadervxx

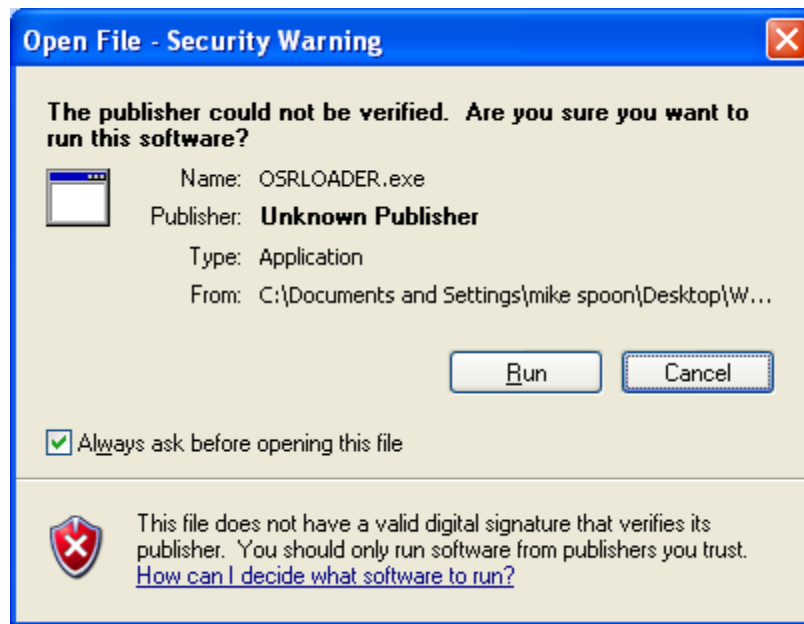
<http://www.rootkit.com/> - InstDrv, an advanced\_loader

This handy tool allows you to register, start, stop and unregister drivers, all from a single GUI. The following Figure shows InstDrv tool




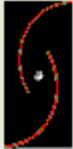
In this tutorial we will use osrloader version 3.0. Launch the executable.





Click the Browse button and set the path to the hello.sys file.


**OSR Driver Loader**
?
✕



Open Systems Resources, Inc.  
 105 Route 101A Suite 19  
 Amherst, NH 03031  
 Ph: (603) 595-6500  
 Fax: (603) 595-6503  
 Ver: V3.0 - Sept 6, 2007

Exit

Help

ServiceGroupOrder

Active Services

Registry Key: yourdrivename

Driver Path: 

Browse

Driver Version:

Driver Size:

Driver File Time:

Display Name:

Service Start:

Load Group: 

Group Load Order

Order In Group:  Type:  Error:

Depend On Group(s): 

AudioGroup

Base

Boot Bus Extender

Boot File System

OSR Driver Loader

Last Status:

MiniFilter Settings

Default Instance:

Altitude:

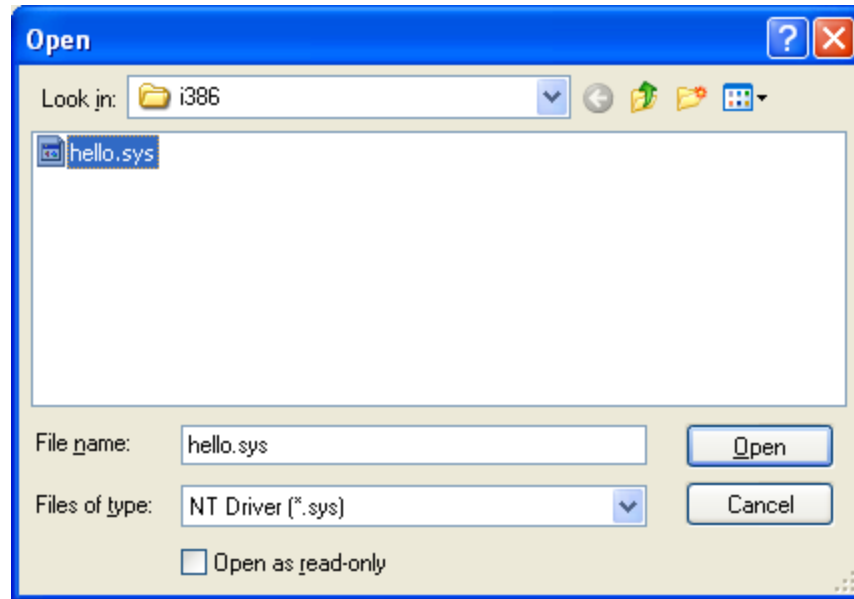
Flags:

Register Service

Unregister Service

Start Service

Stop Service

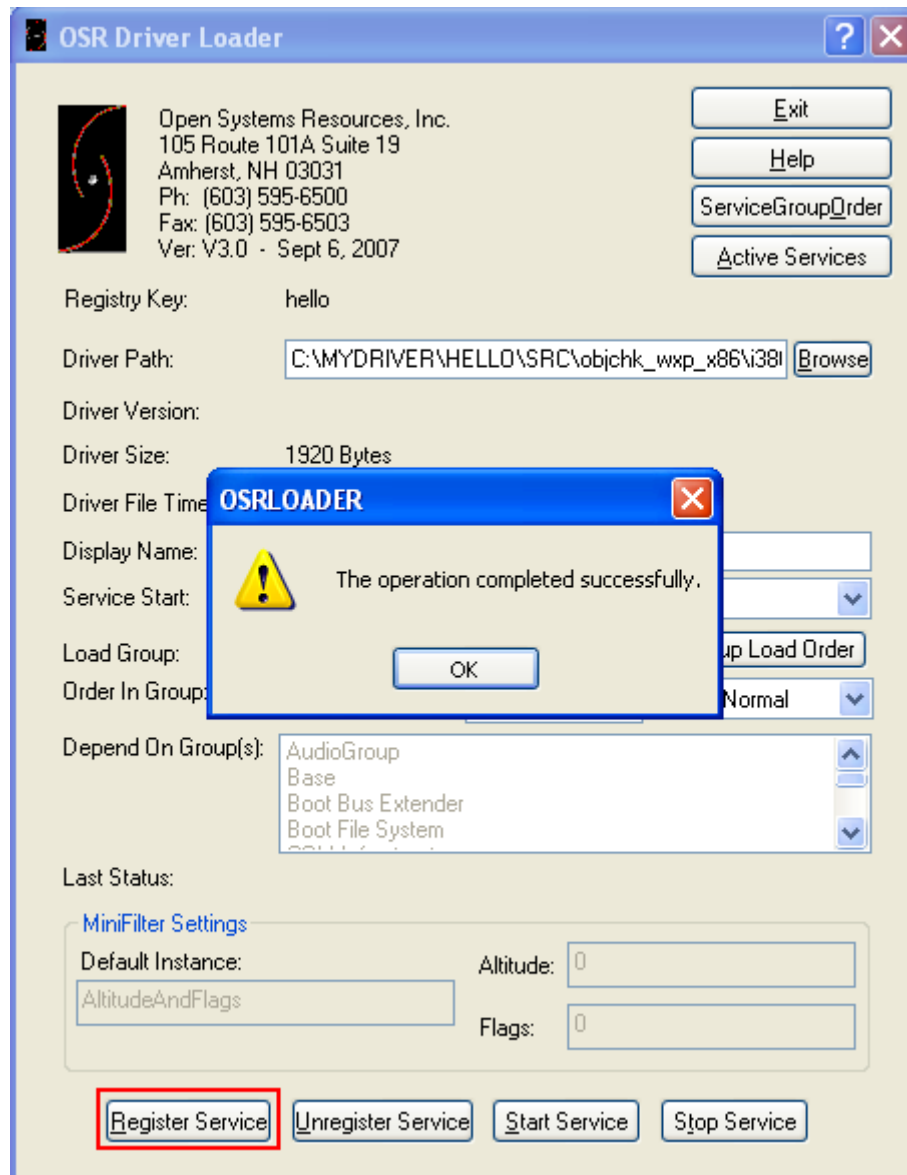


### Installing/Registering a Driver

There are two ways to register a driver as a system service. The first (and easiest) is to use the `CreateService()` API which is documented in MSDN. But basically all this does is create a few values in the Registry on your behalf. The second method is to manually create these values however this step is not recommended. In this tutorial we are using a tool that makes our tasks automated.

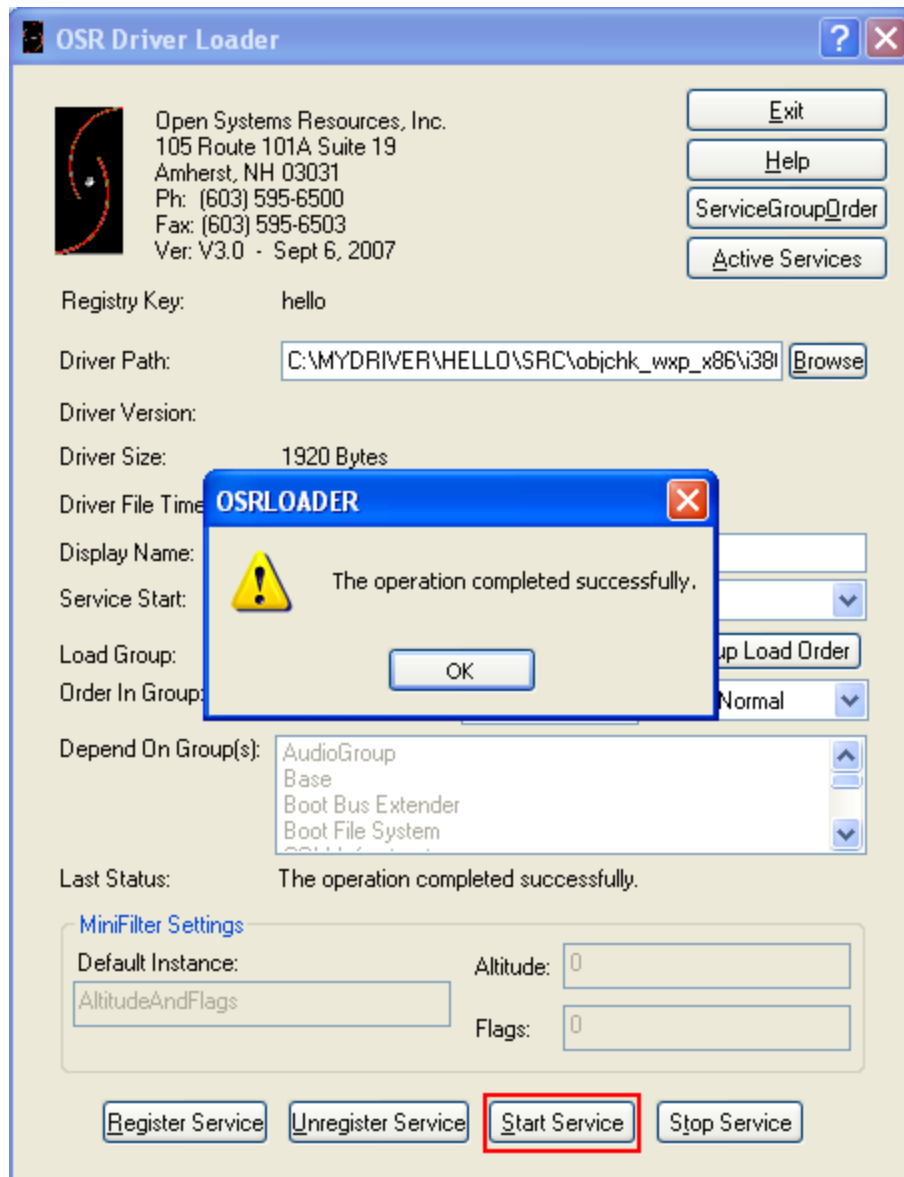
Note that a driver only needs to be registered once (it can only be registered once) - and can be started and stopped as many times as you like after this.

Click the Register Service button to register/install the driver.



## Starting a Driver

Once a driver has been registered as a system-service, it can be loaded (and unloaded) using the Service Control Manager (SCM). In this tutorial we will use osrloader by clicking the Start Service button.



You can start a driver programmatically using the StartService() API call, but it is far easier to goto the command-prompt and type:

```
net start hello
```

The following output will then be displayed:

*The hello service was started successfully.*

Nothing else will appear to happen though because drivers don't (and can't) output any data to the console. Note that at this point the driver has been loaded into kernel-space and your DriverEntr() function has been executed.

There are several types of the service start as shown below.

**OSR Driver Loader**

Open Systems Resources, Inc.  
105 Route 101A Suite 19  
Amherst, NH 03031  
Ph: (603) 595-6500  
Fax: (603) 595-6503  
Ver: V3.0 - Sept 6, 2007

Registry Key: hello

Driver Path: C:\MYDRIVER\HELLO\SRC\objchk\_wxp\_x86\i386\hello.sys Browse

Driver Version:

Driver Size: 1920 Bytes

Driver File Time: Monday, October 26, 2009 21:58:41

Display Name: hello

Service Start: Demand

Load Group: Automatic  
Boot  
Demand  
Disabled  
System

Order In Group:

Depend On Group(s): Base  
Boot Bus Extender  
Boot File System

Last Status: The operation completed successfully.

**MiniFilter Settings**

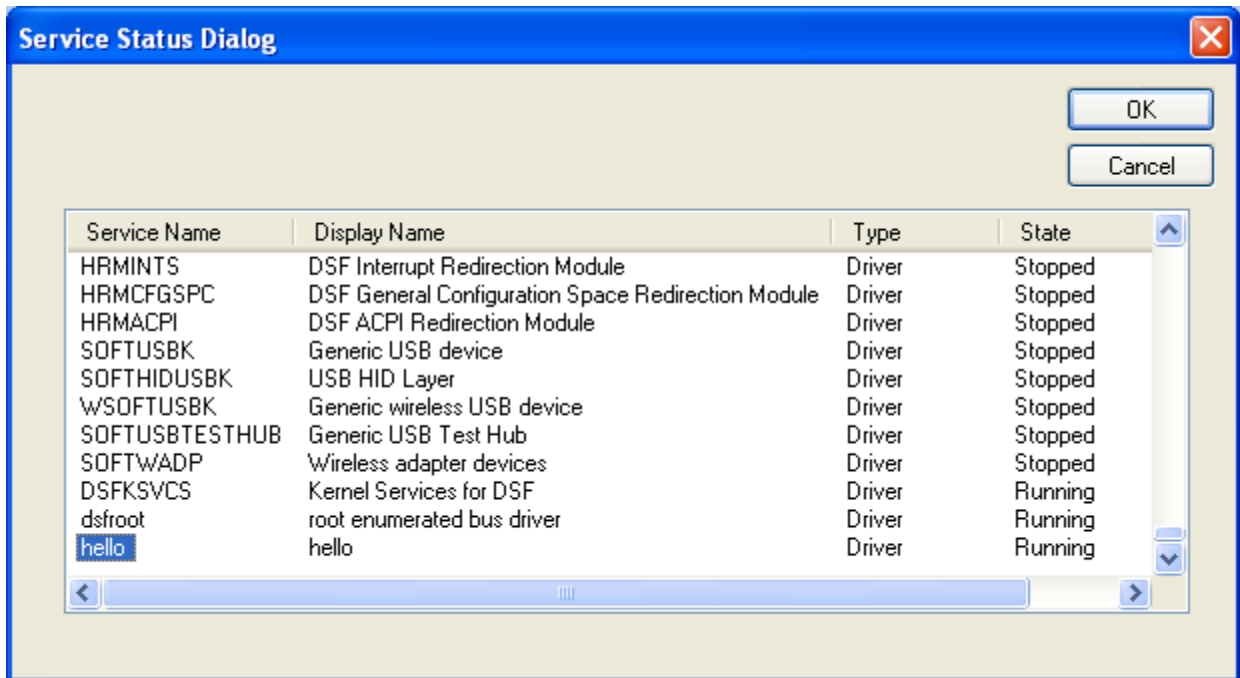
Default Instance: AltitudeAndFlags Altitude: 0

Flags: 0

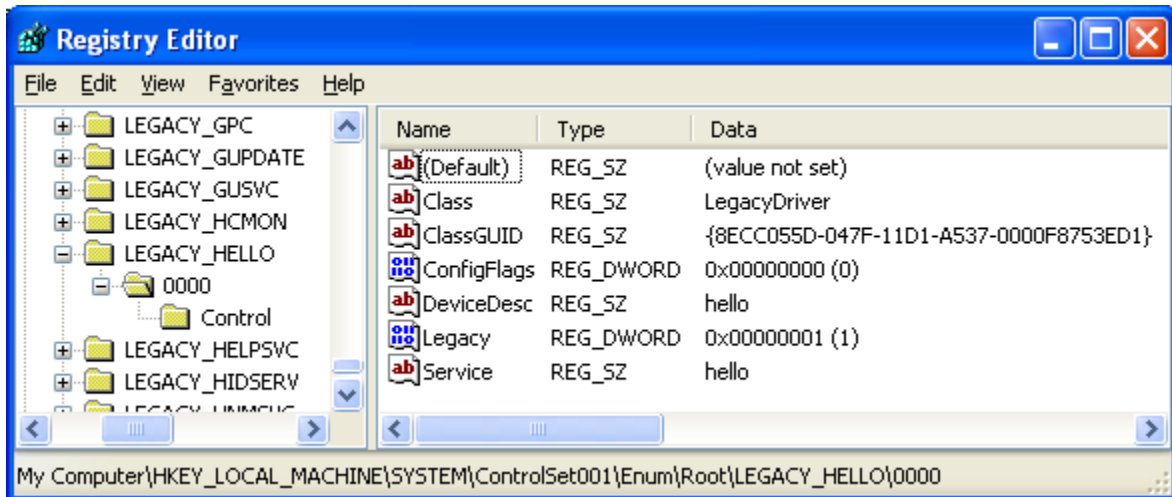
Register Service Unregister Service Start Service Stop Service

Click the Active Services button to verify that our driver has been loaded and running.

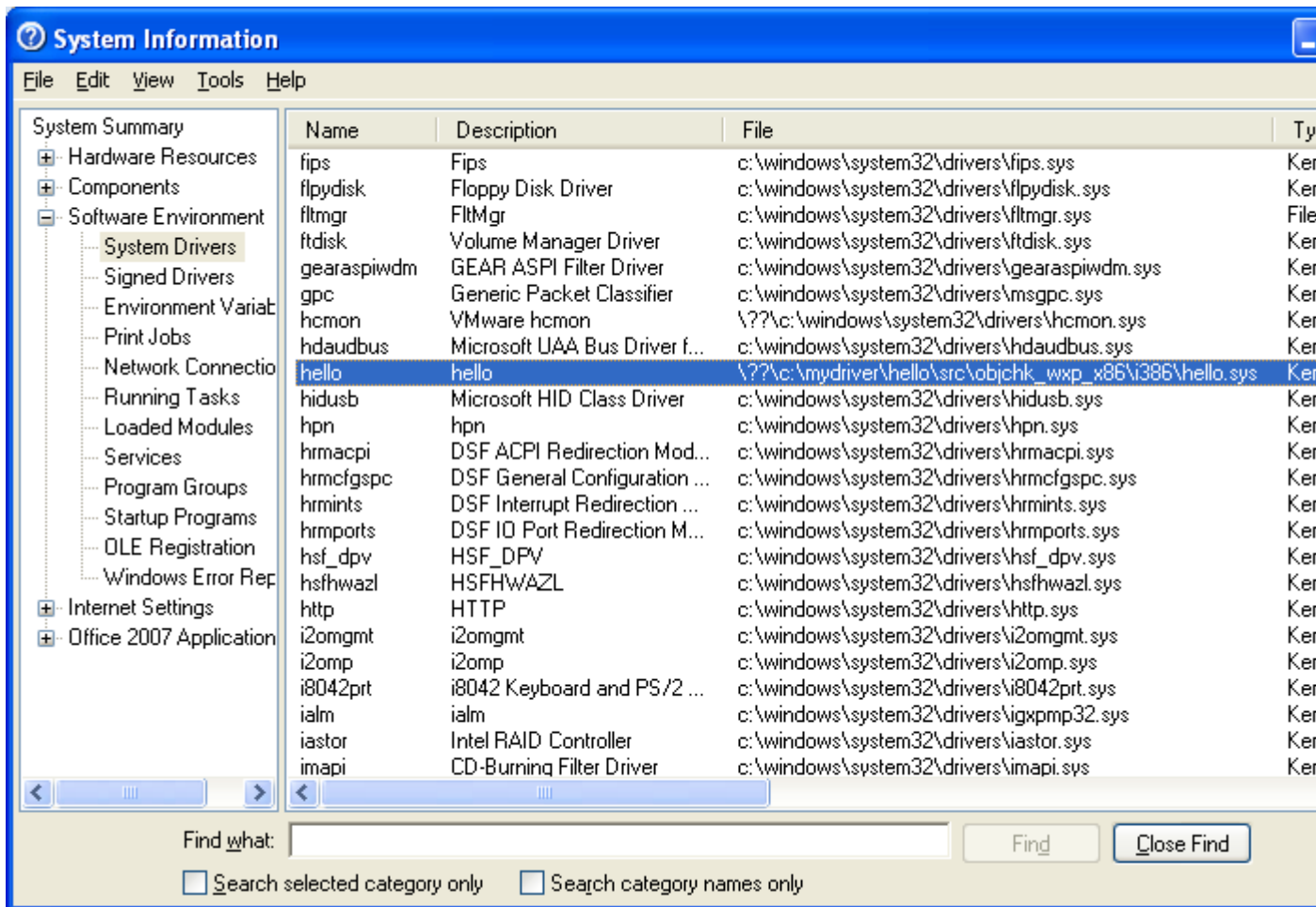




Open registry editor (regedt32) and search the hello word.



Next, click Start > all Programs > Accessories > System Tools > System Information



## Viewing Driver Output

For debugging purposes it is common for drivers to use the `DbgPrint()` API to emit debug messages. The syntax is given below.

```
ULONG DbgPrint(const char *fmt, ...);
```

`DbgPrint()` is very similar to the regular `printf()` routine, the difference being that it can only be called from kernel mode. The debug messages, instead of displaying on the screen, are sent to the kernel debugger (usually WinDbg), assuming that one is attached.

## Unloading a Driver

Stopping (and unloading) a driver is as simple as starting it using the net command:

```
net stop hello
```

However the driver we have built at this point will never unload, because we have omitted a tiny detail. The one thing we left out was the `DriverUnload()` routine, which the

service-control-manager calls when a driver is about to unload. This unload routine must be specified during DriverEntry() if our driver is to be unloadable, and a pointer to the routine stored in the DriverObject:

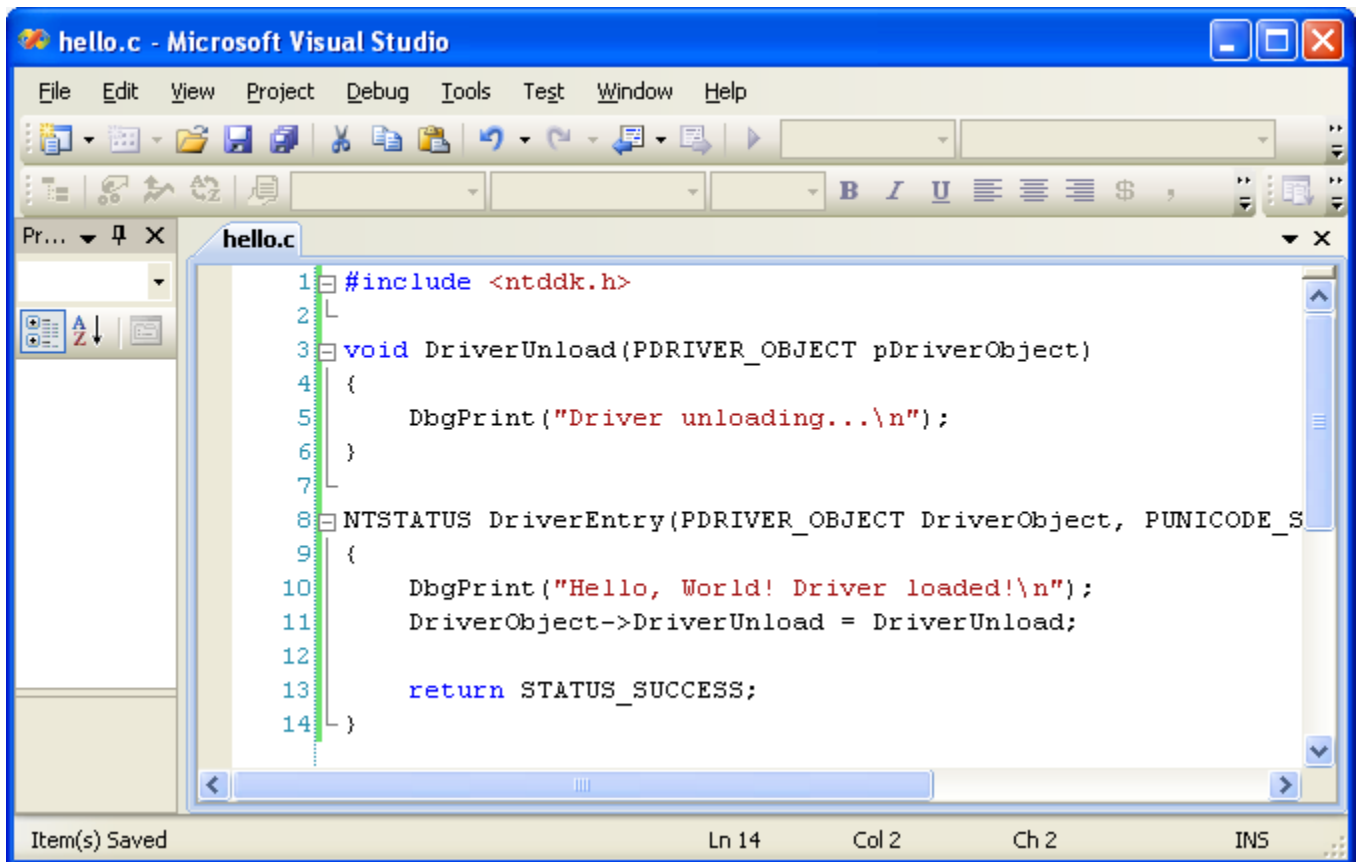
```
#include <ntddk.h>

void DriverUnload(PDRIVER_OBJECT pDriverObject)
{
    DbgPrint("Driver unloading...\n");
}

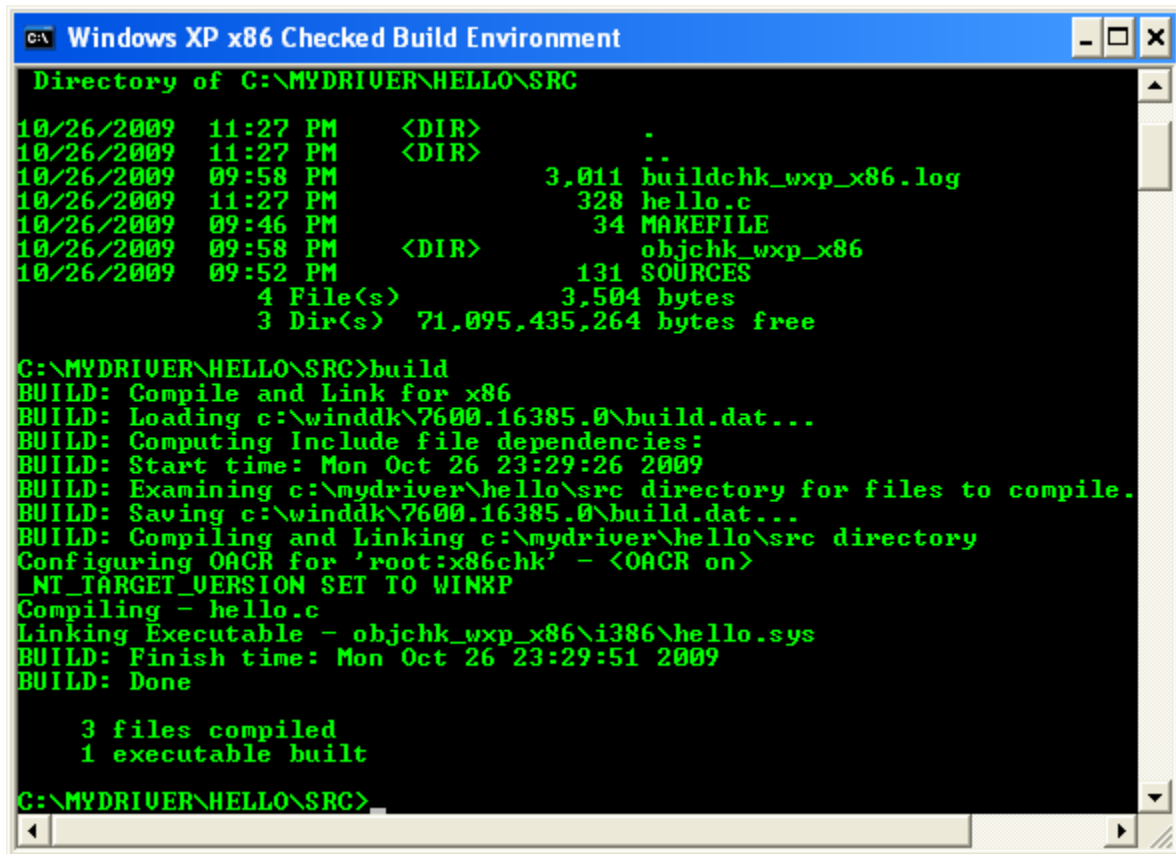
NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING
RegistryPath)
{
    DbgPrint("Hello, World! Driver loaded!\n");
    DriverObject->DriverUnload = DriverUnload;

    return STATUS_SUCCESS;
}
```

Modify your basic driver source code by adding the above code.



You will have to rebuild the driver.



```

C:\ Windows XP x86 Checked Build Environment

Directory of C:\MYDRIVER\HELLO\SRC

10/26/2009  11:27 PM    <DIR>          .
10/26/2009  11:27 PM    <DIR>          ..
10/26/2009  09:58 PM             3,011 buildchk_wxp_x86.log
10/26/2009  11:27 PM             328 hello.c
10/26/2009  09:46 PM              34 MAKEFILE
10/26/2009  09:58 PM    <DIR>          objchk_wxp_x86
10/26/2009  09:52 PM             131 SOURCES
               4 File(s)              3,504 bytes
               3 Dir(s)  71,095,435,264 bytes free

C:\MYDRIVER\HELLO\SRC>build
BUILD: Compile and Link for x86
BUILD: Loading c:\winddk\7600.16385.0\build.dat...
BUILD: Computing Include file dependencies:
BUILD: Start time: Mon Oct 26 23:29:26 2009
BUILD: Examining c:\mydriver\hello\src directory for files to compile.
BUILD: Saving c:\winddk\7600.16385.0\build.dat...
BUILD: Compiling and Linking c:\mydriver\hello\src directory
Configuring OACR for 'root:x86chk' - <OACR on>
_NT_TARGET_VERSION SET TO WINXP
Compiling - hello.c
Linking Executable - objchk_wxp_x86\i386\hello.sys
BUILD: Finish time: Mon Oct 26 23:29:51 2009
BUILD: Done

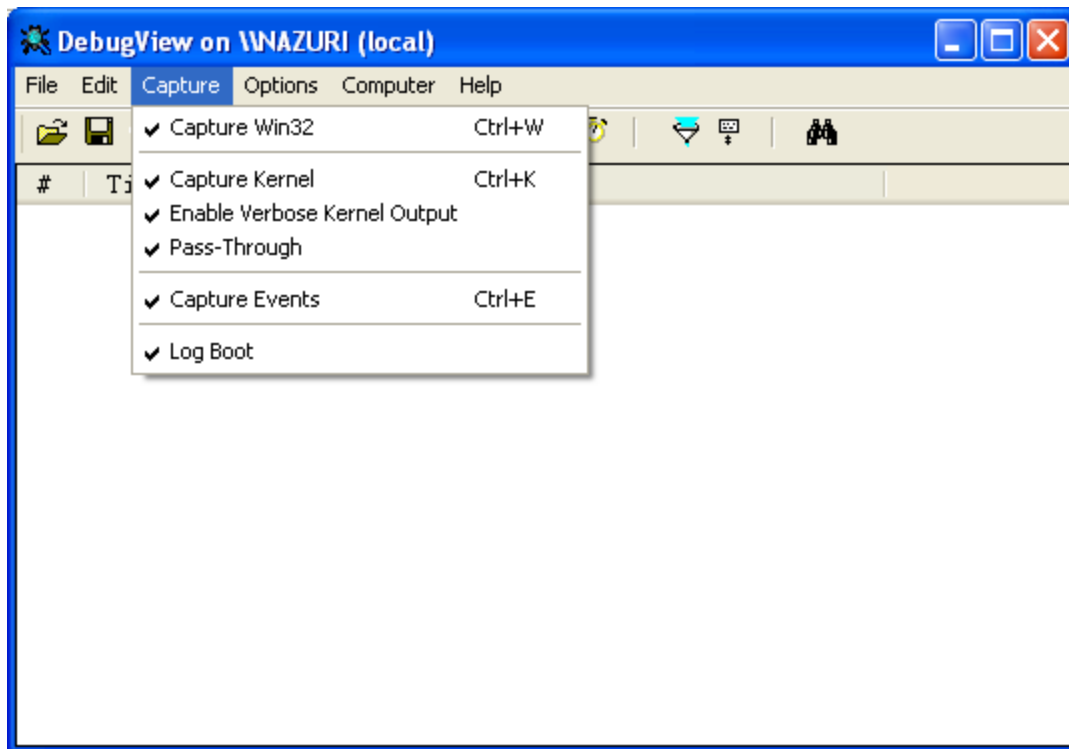
      3 files compiled
      1 executable built

C:\MYDRIVER\HELLO\SRC>
  
```

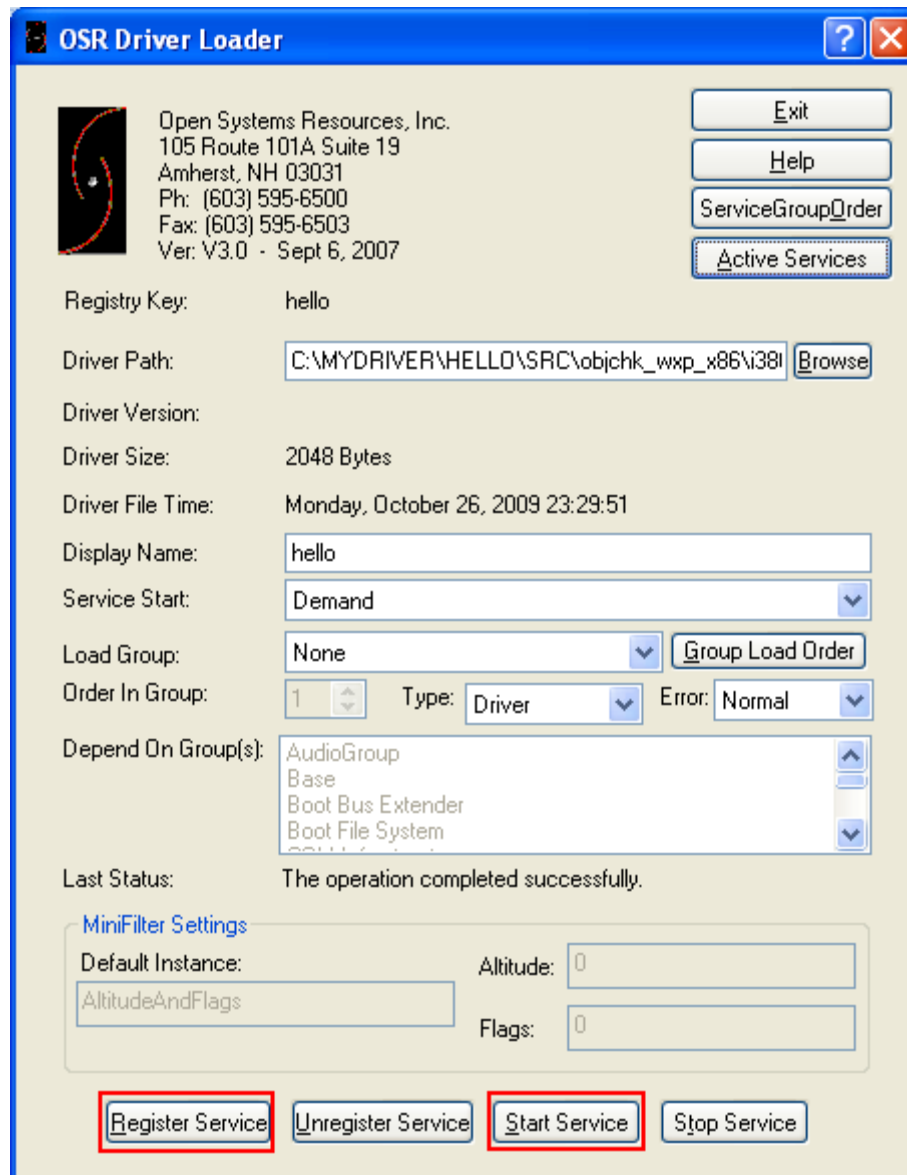
To clean-up all the 'mess' done, reboot your computer.

In our case to simplify the steps to view the verbose output of the driver based on using the DbgPrint(), we use DebugView tool. DebugView tool can be downloaded from [Microsoft technet](http://www.microsoft.com/technet).

Run this tool and enable the Capture settings as shown below.



Next, run the osrloader as done previously. Register and start the Hello driver.



**OSR Driver Loader**

Open Systems Resources, Inc.  
105 Route 101A Suite 19  
Amherst, NH 03031  
Ph: (603) 595-6500  
Fax: (603) 595-6503  
Ver: V3.0 - Sept 6, 2007

Registry Key: hello

Driver Path: C:\MYDRIVER\HELLO\SRC\objchk\_wxp\_x86\i386\hello.sys Browse

Driver Version:

Driver Size: 2048 Bytes

Driver File Time: Monday, October 26, 2009 23:29:51

Display Name: hello

Service Start: Demand

Load Group: None Group Load Order

Order In Group: 1 Type: Driver Error: Normal

Depend On Group(s): AudioGroup, Base, Boot Bus Extender, Boot File System

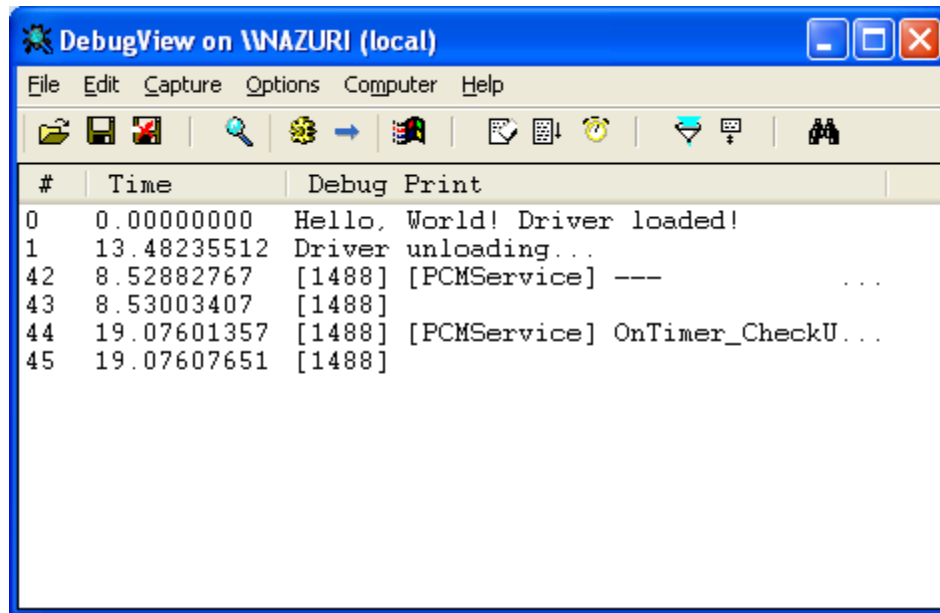
Last Status: The operation completed successfully.

**MiniFilter Settings**

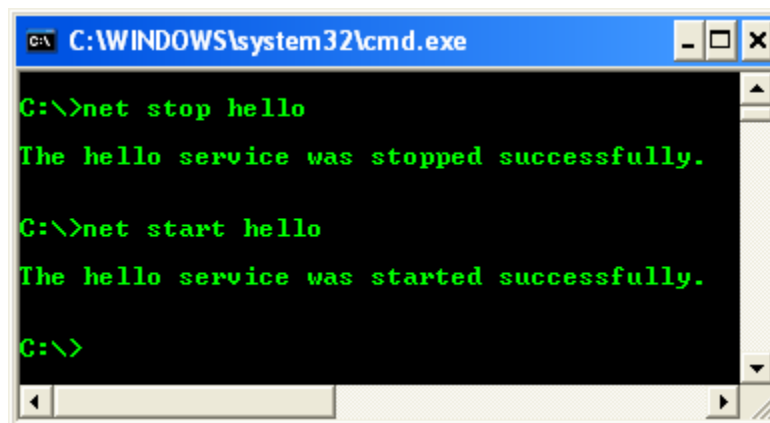
Default Instance: AltitudeAndFlags Altitude: 0 Flags: 0

Register Service Unregister Service Start Service Stop Service

Notice the DebugView messages.



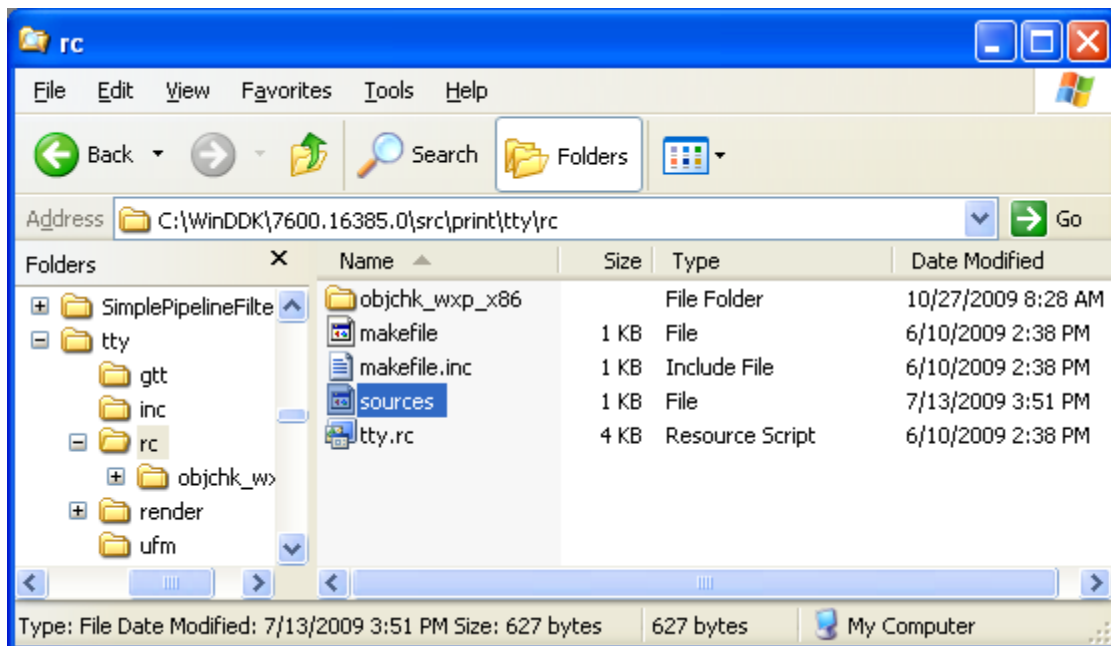
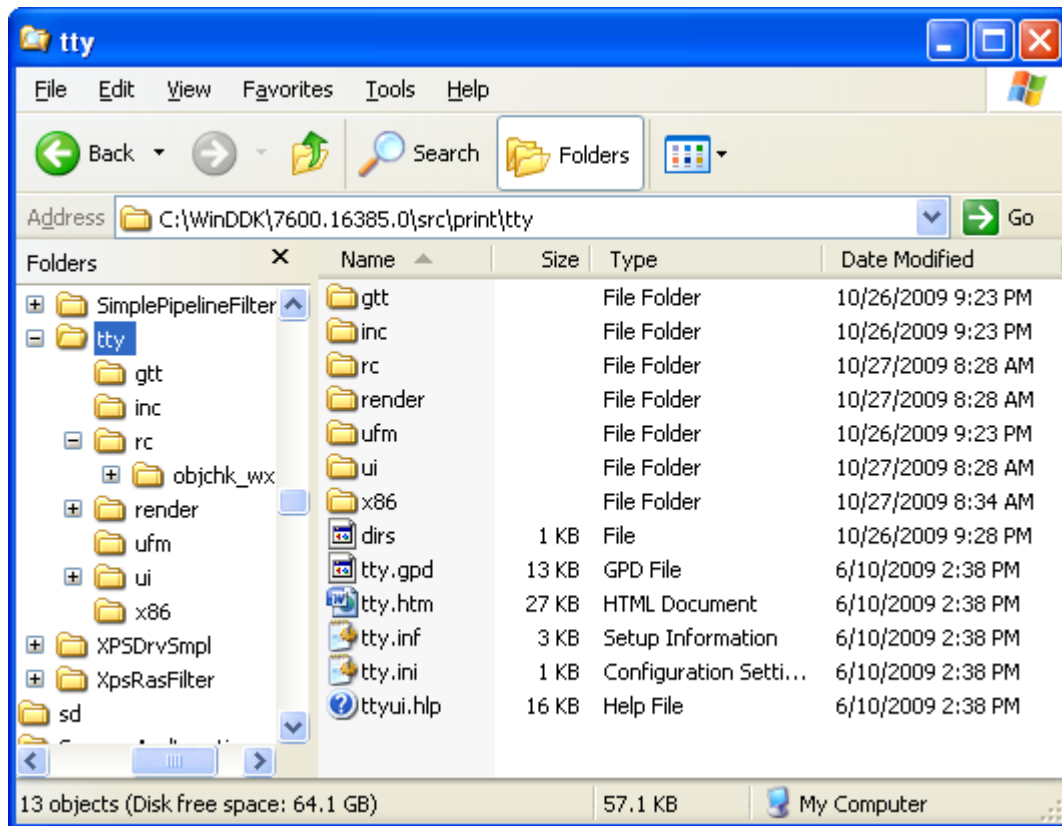
Then, you also are able to use "net start" and "net stop" as often as you like to start and stop the driver.



### Print Devices: Generic Text-Only Driver Sample

The sample codes for various Windows device drivers can be found under the Windows Driver Kit installation if the sample code module has been selected during the WDK installation. Those Windows driver sample source codes can be found under the SRC subfolder:

**C:\WinDDK\7600.16385.0\src**

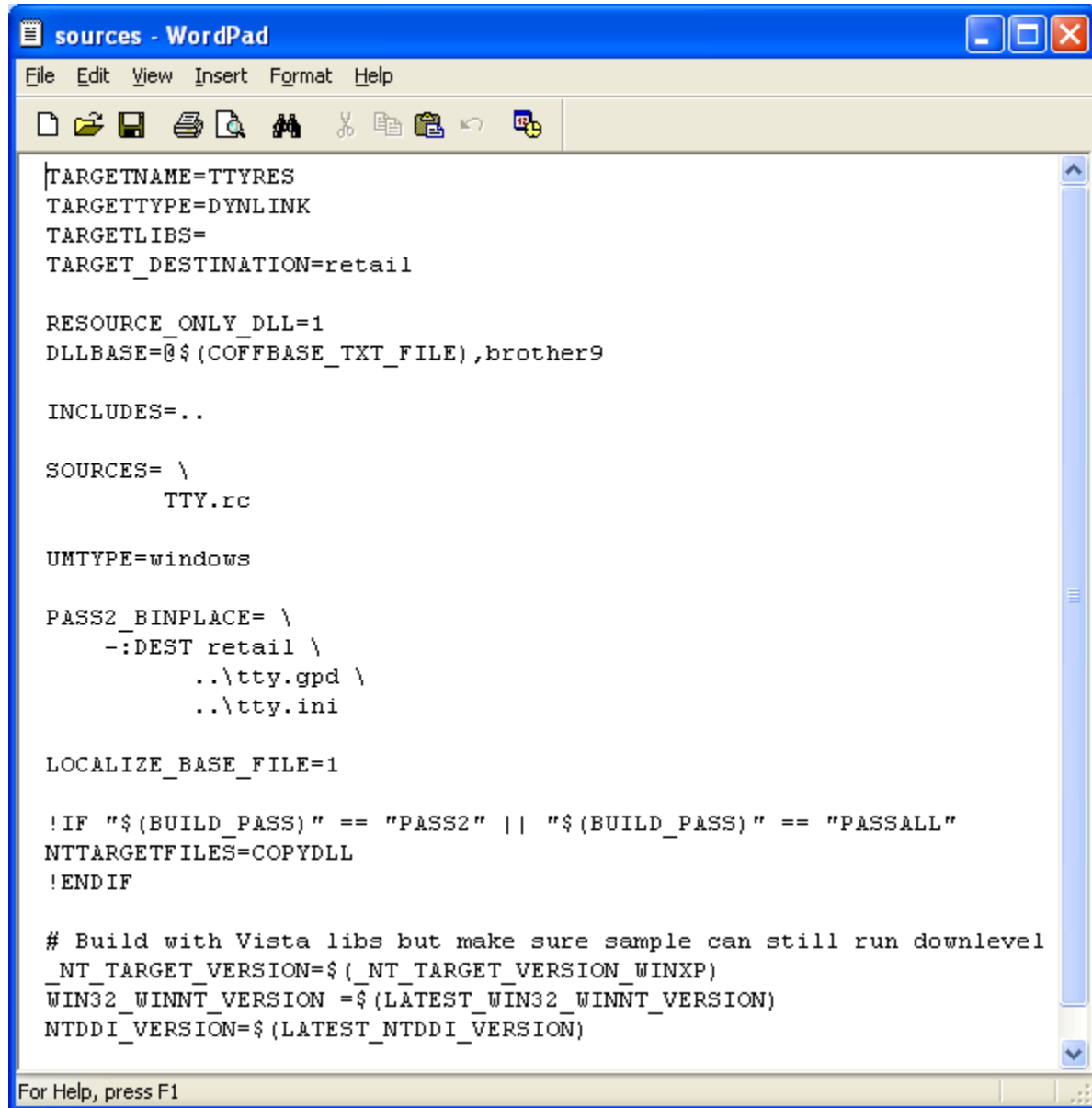


The online version can be found at [Windows Driver Code Sample](#). In this exercise we will try building a sample generic text-only printing driver. The Generic Text-Only Driver (TTY) sample demonstrates how to implement a print driver that is Unidrv-based, generic, and text-only.



The TTY driver is a generic, text-only driver. It prints only text, and it prints the text in the native font of the print device, regardless of any formatting in the original document. The TTY driver sample uses the same source files as in the in-box driver, which means you can expect the same output as the in-box generic text-only driver when you build and install this sample.

The following Figures show the [SOURCES](#), MAKEFILE and dirs files content.



```
TARGETNAME=TTYRES
TARGETTYPE=DYNLINK
TARGETLIBS=
TARGET_DESTINATION=retail

RESOURCE_ONLY_DLL=1
DLLBASE=@$(COFFBASE_TXT_FILE),brother9

INCLUDES=..

SOURCES= \
    TTY.rc

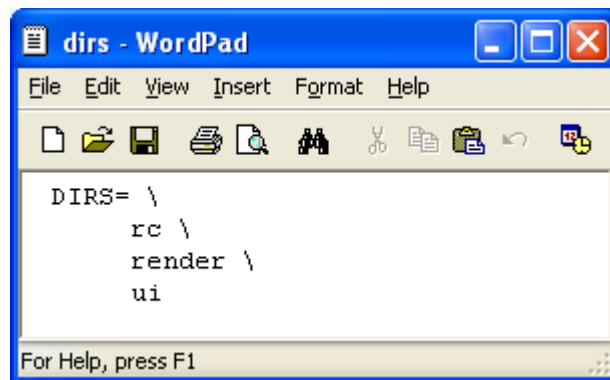
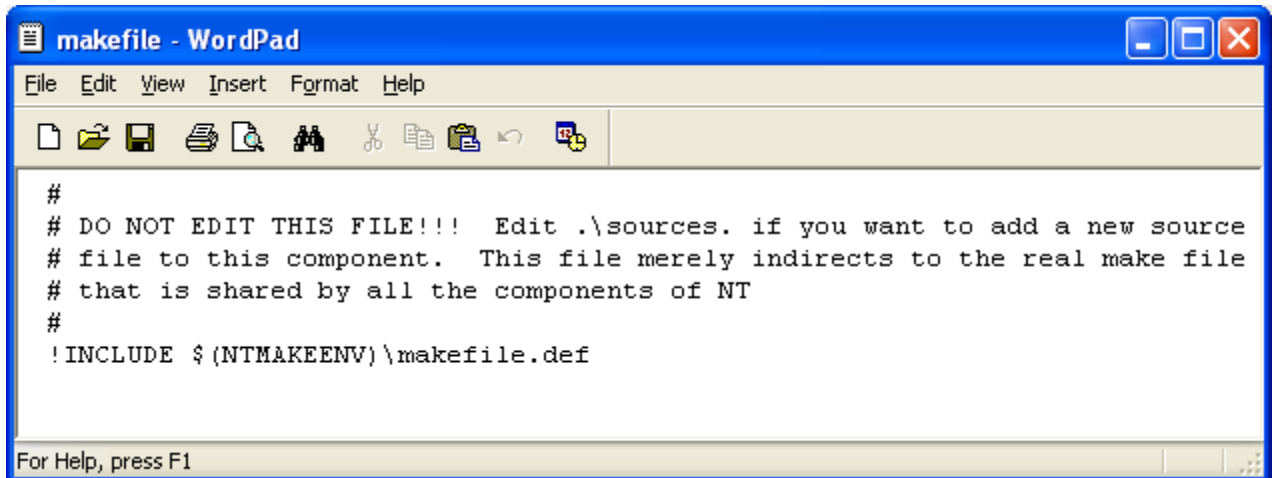
UMTYPE=windows

PASS2_BINPLACE= \
    -:DEST retail \
        ..\tty.gpd \
        ..\tty.ini

LOCALIZE_BASE_FILE=1

!IF "$(BUILD_PASS)" == "PASS2" || "$(BUILD_PASS)" == "PASSALL"
NTTARGETFILES=COPYDLL
!ENDIF

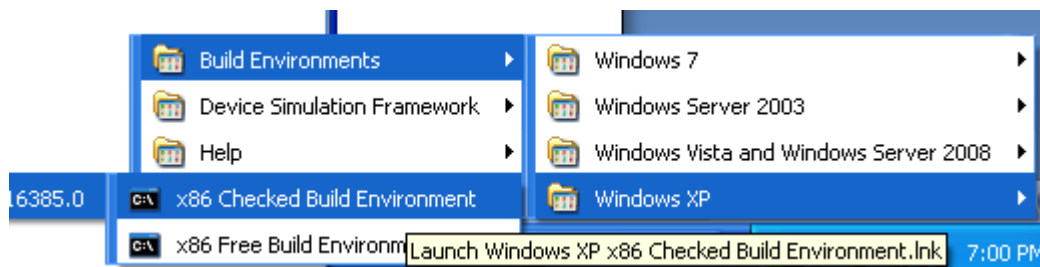
# Build with Vista libs but make sure sample can still run downlevel
_NT_TARGET_VERSION=$( _NT_TARGET_VERSION_WINXP)
WIN32_WINNT_VERSION =$(LATEST_WIN32_WINNT_VERSION)
NTDDI_VERSION=$(LATEST_NTDDI_VERSION)
```



To create a version with verbose debug output, add WPP\_DEBUG to the compile defines in the sources file.

To build the samples, run build -cZ from each directory (RC, UI, and Render). The DLLs are placed in the appropriate platform directory (that is, i386).

Open the Windows XP Check Build Environment.



Run the build -cZ command from RC, UI and Render directories.

```
C:\ Windows XP x86 Free Build Environment

C:\WinDDK\7600.16385.0\src\print\tty\rc>build -cZ
BUILD: Compile and Link for x86
BUILD: Start time: Sun Jan 03 19:53:00 2010
BUILD: Examining c:\winddk\7600.16385.0\src\print\tty\rc directory for files to
compile.
BUILD: Compiling and Linking c:\winddk\7600.16385.0\src\print\tty\rc directory
Configuring OACR for 'WDKSamples:x86fre' - <OACR on>
_NT_TARGET_VERSION SET TO WINXP
Compiling resources - tty.rc
Linking Executable - objfre_wxp_x86\i386\ttyres.dll
BUILD: Finish time: Sun Jan 03 19:53:08 2010
BUILD: Done

    1 file compiled
    1 executable built

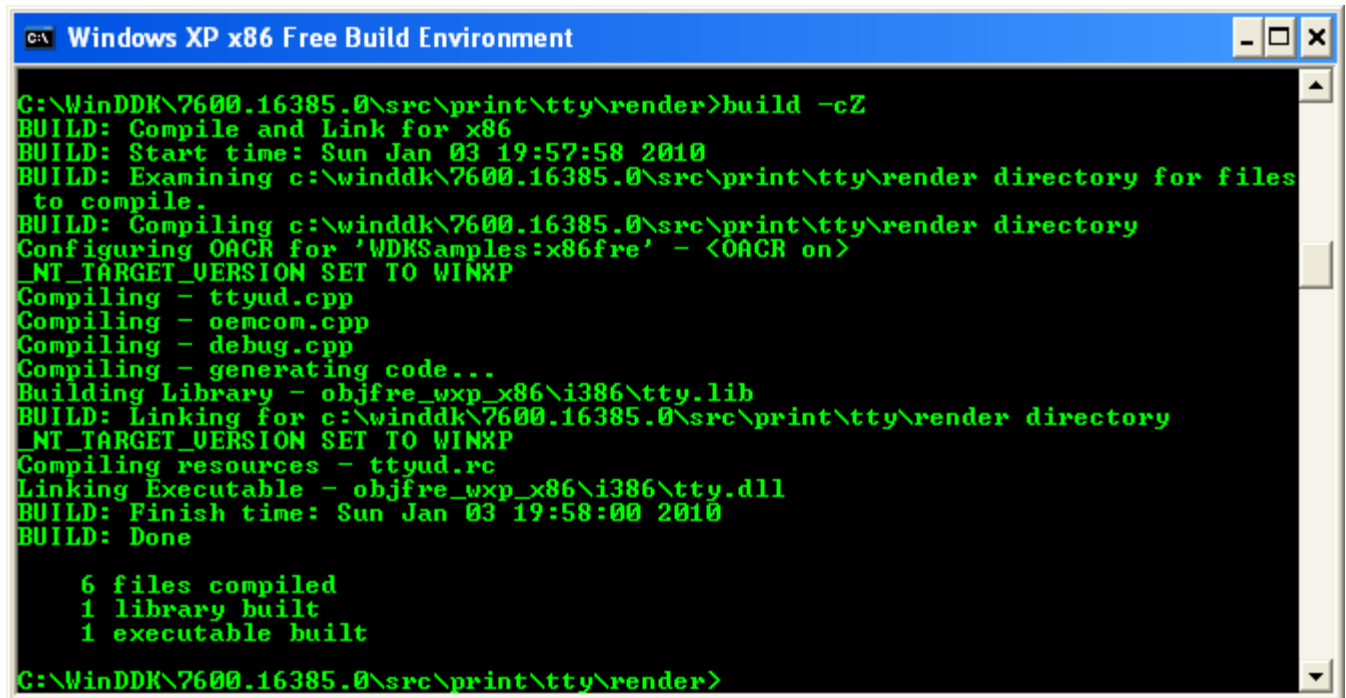
C:\WinDDK\7600.16385.0\src\print\tty\rc>_
```

```
C:\ Windows XP x86 Free Build Environment

C:\WinDDK\7600.16385.0\src\print\tty\ui>build -cZ
BUILD: Compile and Link for x86
BUILD: Start time: Sun Jan 03 19:56:54 2010
BUILD: Examining c:\winddk\7600.16385.0\src\print\tty\ui directory for files to
compile.
BUILD: Compiling c:\winddk\7600.16385.0\src\print\tty\ui directory
Configuring OACR for 'WDKSamples:x86fre' - <OACR on>
_NT_TARGET_VERSION SET TO WINXP
Compiling - ttyui.cpp
Compiling - oemcom.cpp
Compiling - debug.cpp
Compiling - generating code...
Building Library - objfre_wxp_x86\i386\ttyui.lib
BUILD: Linking for c:\winddk\7600.16385.0\src\print\tty\ui directory
_NT_TARGET_VERSION SET TO WINXP
Compiling resources - ttyui.rc
Linking Executable - objfre_wxp_x86\i386\ttyui.dll
BUILD: Finish time: Sun Jan 03 19:57:00 2010
BUILD: Done

    6 files compiled
    1 library built
    1 executable built

C:\WinDDK\7600.16385.0\src\print\tty\ui>
```



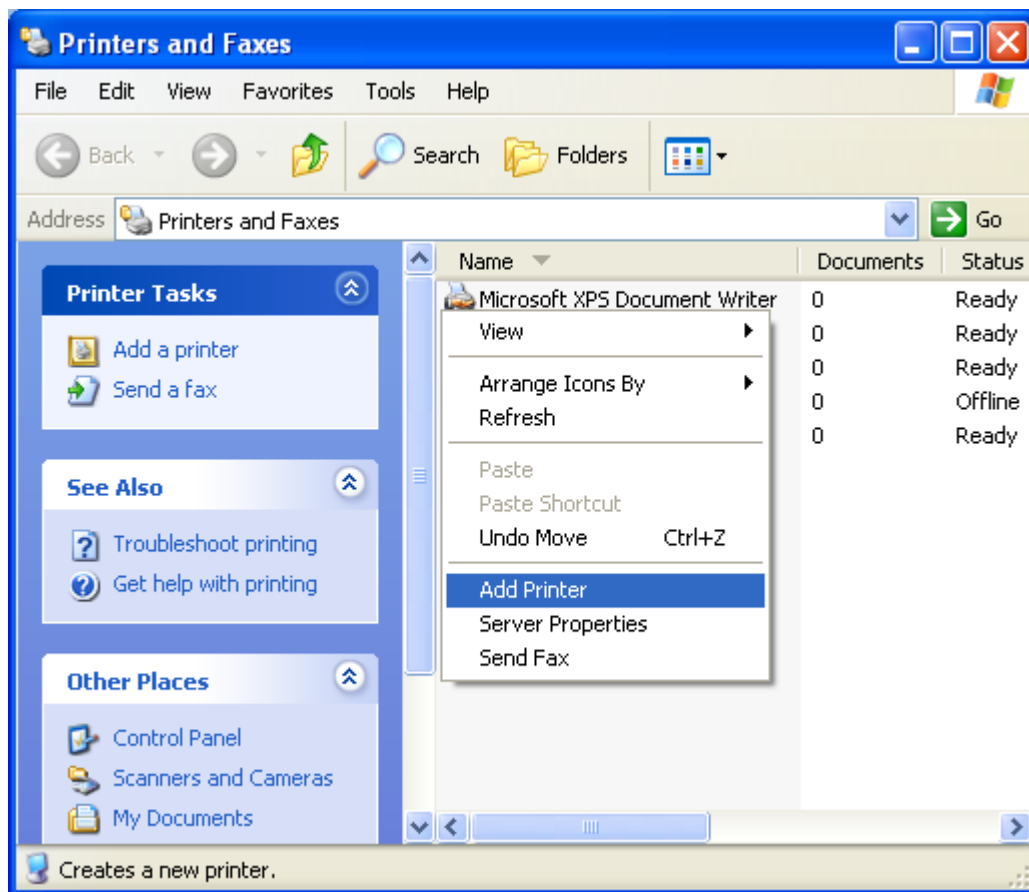
```
C:\ Windows XP x86 Free Build Environment

C:\WinDDK\7600.16385.0\src\print\tty\render>build -cZ
BUILD: Compile and Link for x86
BUILD: Start time: Sun Jan 03 19:57:58 2010
BUILD: Examining c:\winddk\7600.16385.0\src\print\tty\render directory for files
to compile.
BUILD: Compiling c:\winddk\7600.16385.0\src\print\tty\render directory
Configuring OACR for 'WDKSamples:x86fre' - <OACR on>
_NT_TARGET_VERSION SET TO WINXP
Compiling - ttyud.cpp
Compiling - oemcom.cpp
Compiling - debug.cpp
Compiling - generating code...
Building Library - objfre_wxp_x86\i386\tty.lib
BUILD: Linking for c:\winddk\7600.16385.0\src\print\tty\render directory
_NT_TARGET_VERSION SET TO WINXP
Compiling resources - ttyud.rc
Linking Executable - objfre_wxp_x86\i386\tty.dll
BUILD: Finish time: Sun Jan 03 19:58:00 2010
BUILD: Done

    6 files compiled
    1 library built
    1 executable built

C:\WinDDK\7600.16385.0\src\print\tty\render>
```


After building the samples, you can install the samples by using the Add Printer wizard. Select the local printer, click Have Disk, and point to the directory that contains the tty.inf file (this file already provided for us and may need some modification, for example the path of some of the file. We do nothing in this case). Microsoft Windows 2000 and later drivers do not need to be copied to the local directory that contains tty.inf. The following Figures show the steps on how to install the printer driver as we do for Windows.



### Add Printer Wizard

**Local or Network Printer**

The wizard needs to know which type of printer to set up.




Select the option that describes the printer you want to use:

☒ Local printer attached to this computer

☐ Automatically detect and install my Plug and Play printer

☐ A network printer, or a printer attached to another computer


 To set up a network printer that is not attached to a print server, use the "Local printer" option.

< Back   Next >   Cancel

### Add Printer Wizard

**Select a Printer Port**


Computers communicate with printers through ports.



Select the port you want your printer to use. If the port is not listed, you can create a new port.

☒ Use the following port: LPT1: (Recommended Printer Port)

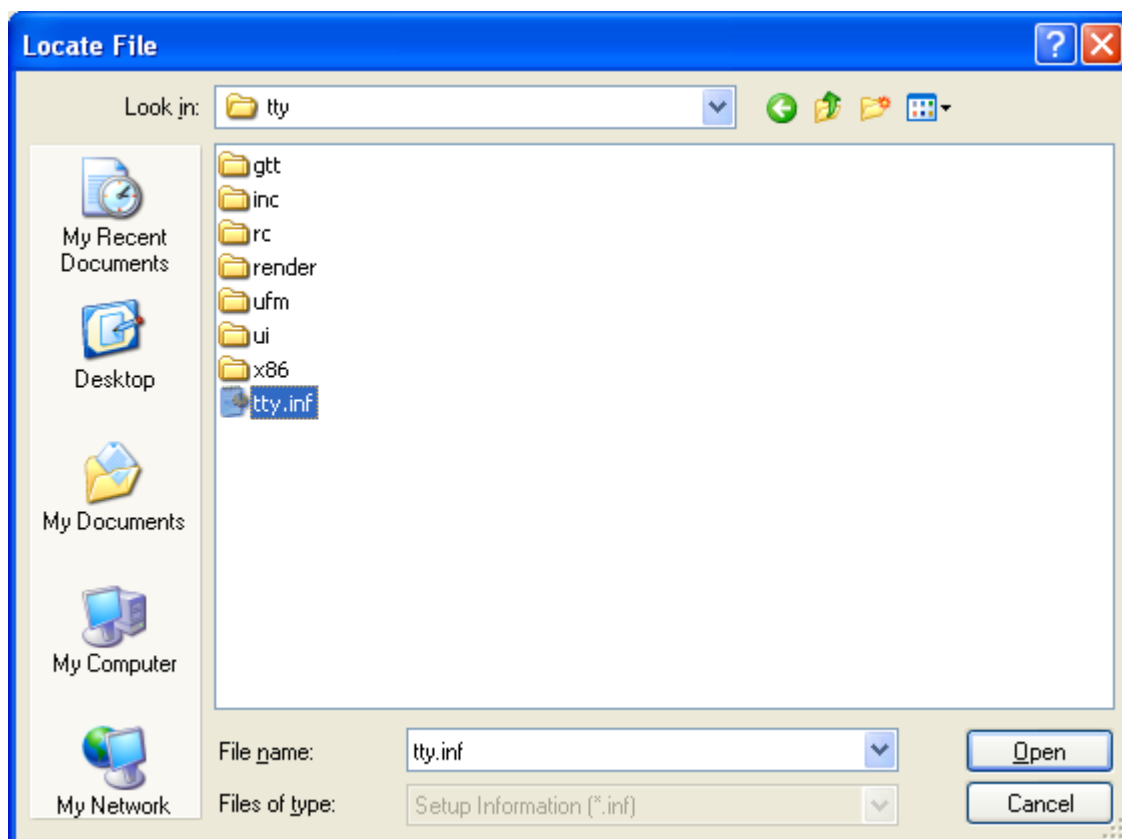
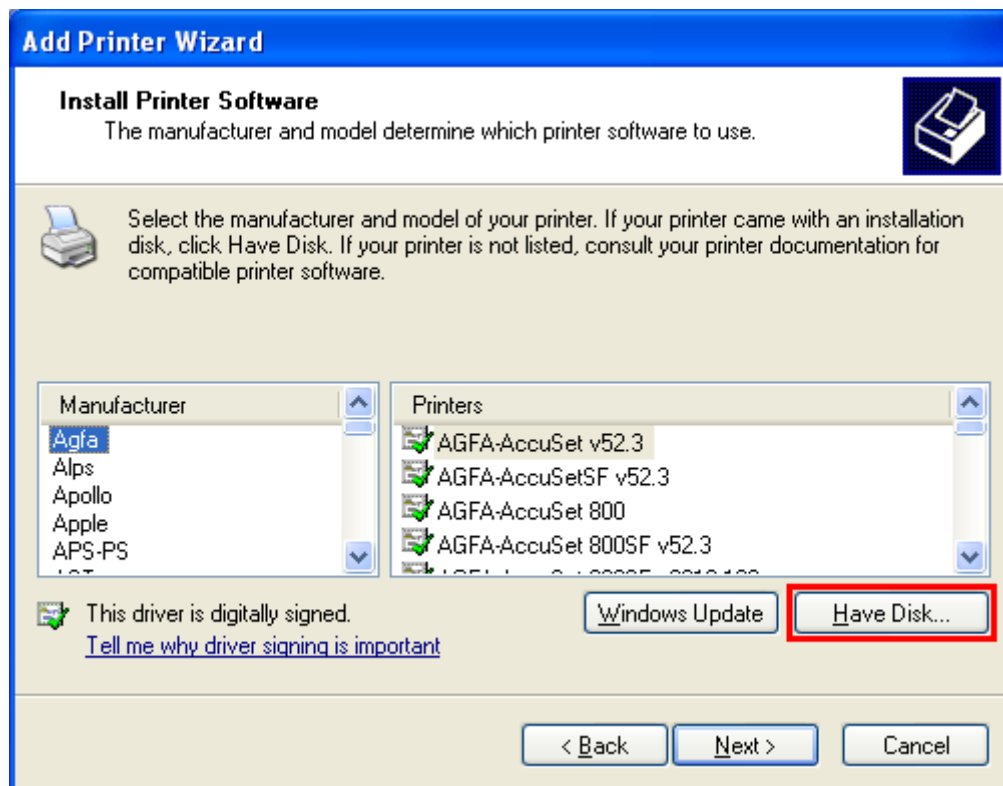
Note: Most computers use the LPT1: port to communicate with a local printer. The connector for this port should look something like this:

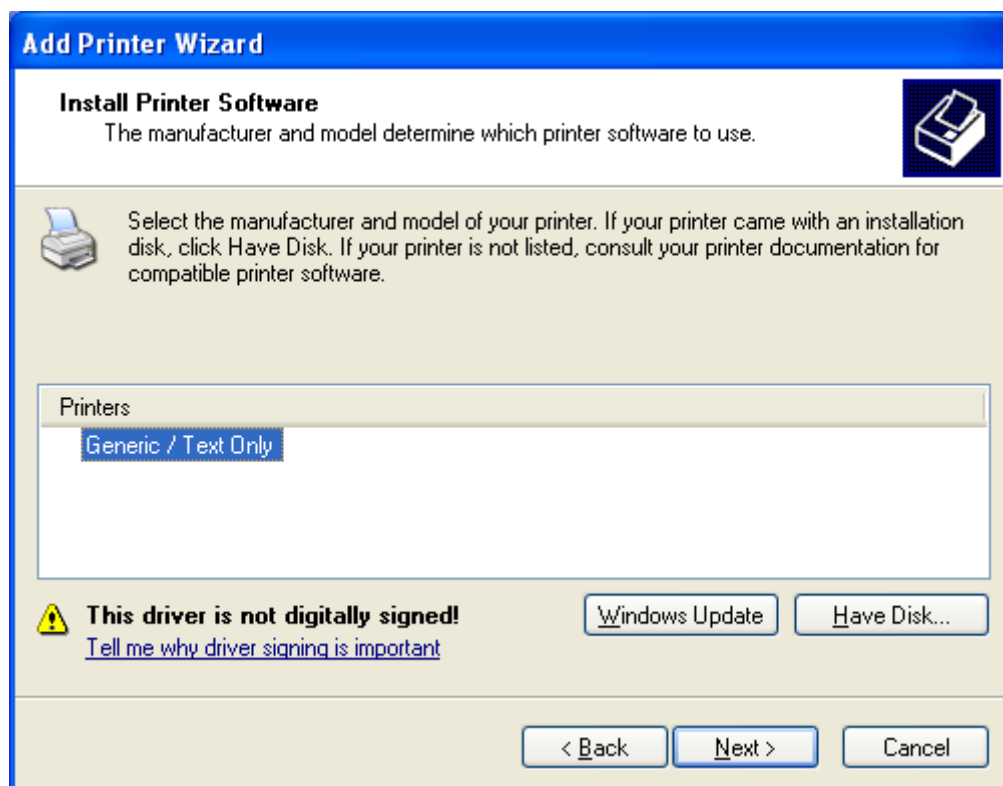
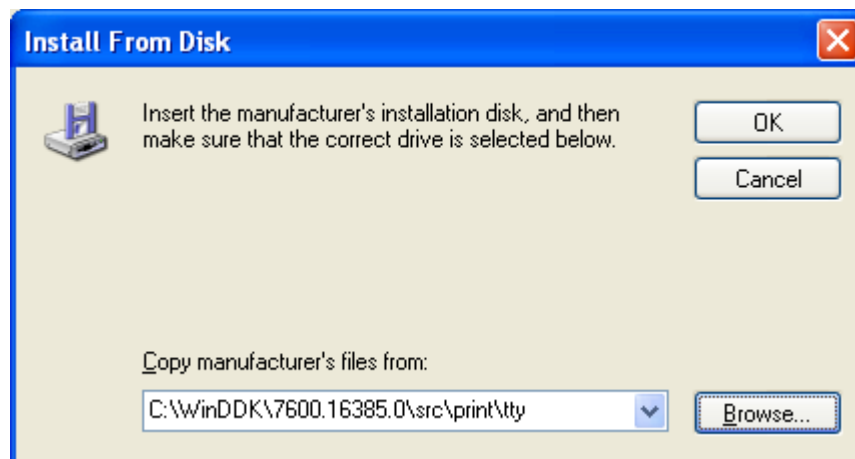


☐ Create a new port:

Type of port: Adobe PDF Port

< Back   Next >   Cancel








**Add Printer Wizard**

**Name Your Printer**  
You must assign a name to this printer.



Type a name for this printer. Because some programs do not support printer and server name combinations of more than 31 characters, it is best to keep the name as short as possible.


Printer name:

Do you want to use this printer as the default printer?

☐ Yes  
☒ No

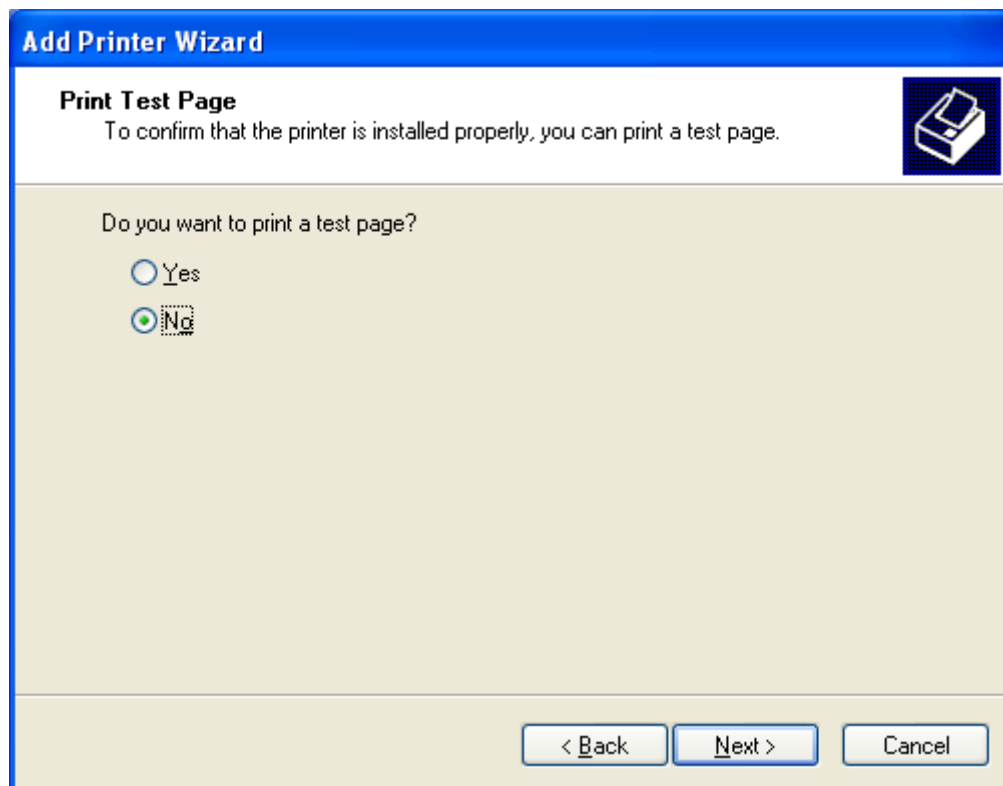
**Add Printer Wizard**

**Printer Sharing**  
You can share this printer with other network users.

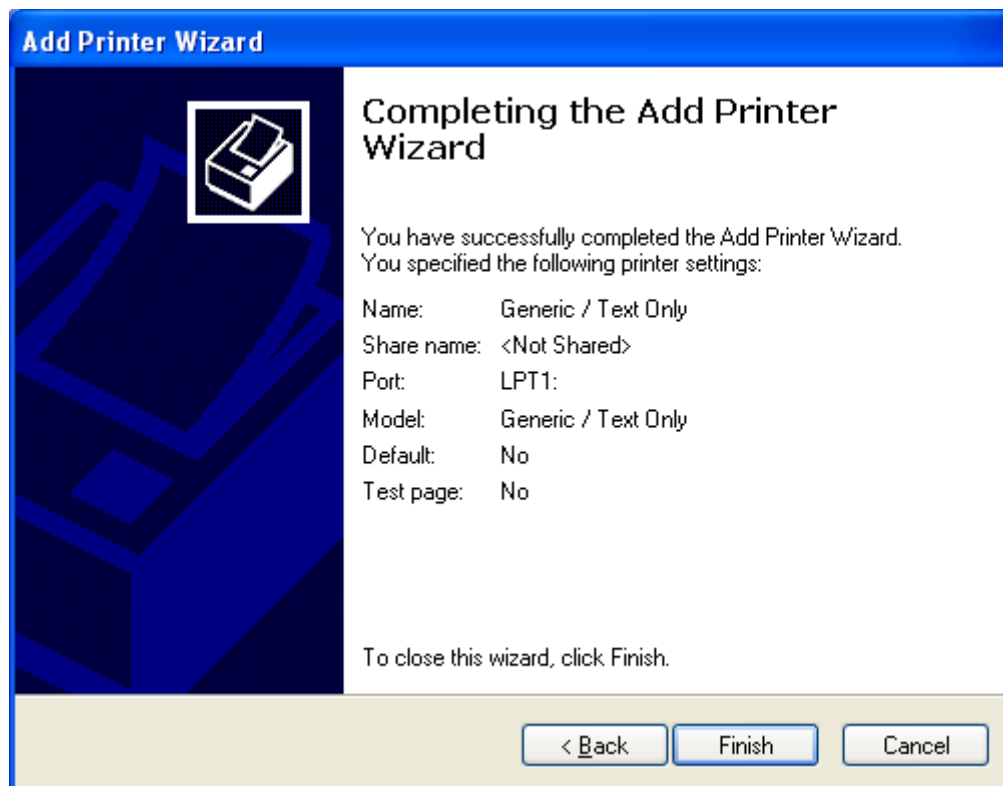


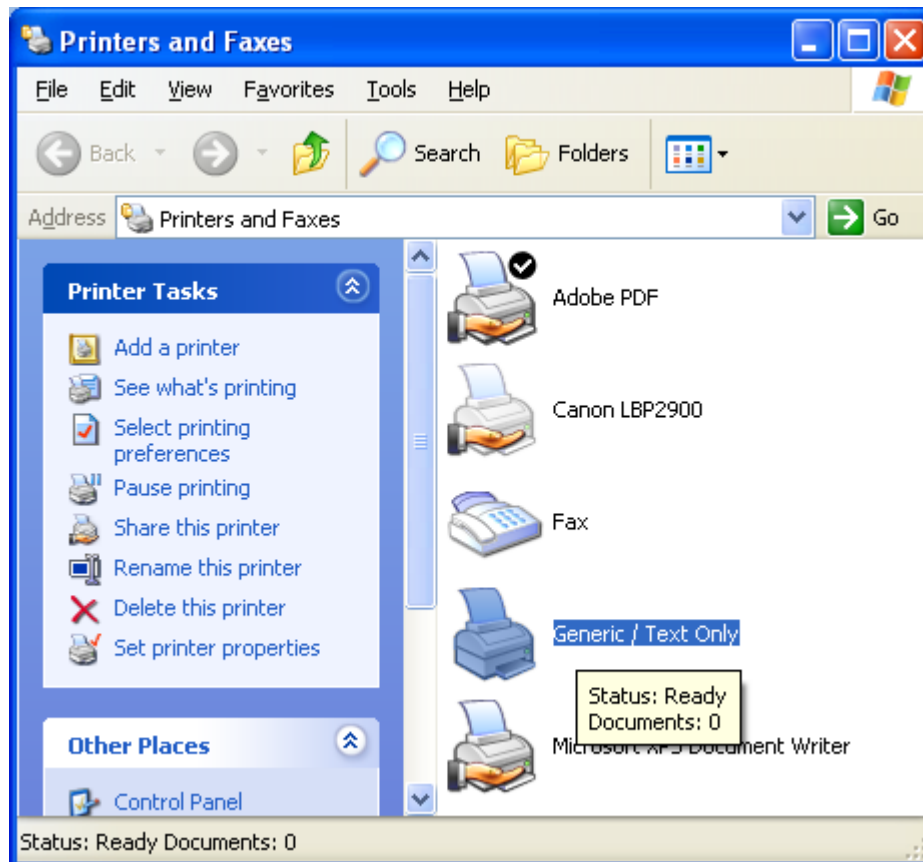
If you want to share this printer, you must provide a share name. You can use the suggested name or type a new one. The share name will be visible to other network users.

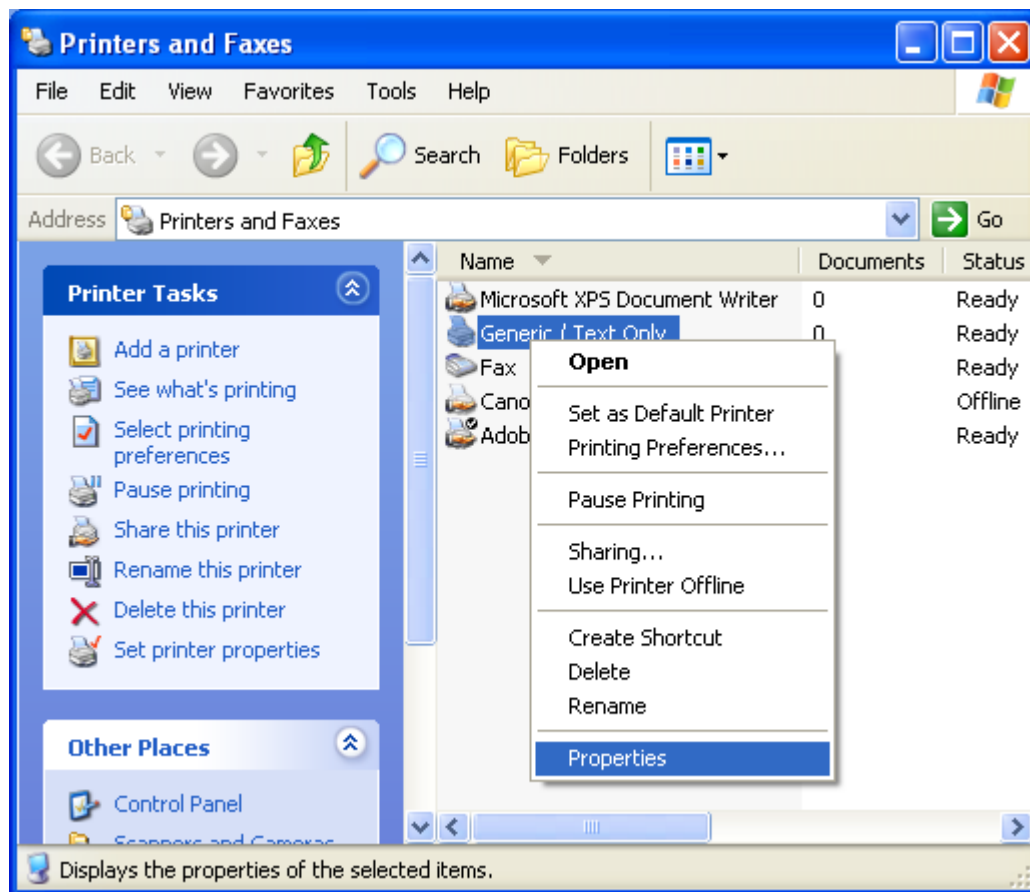
☒ Do not share this printer  
☐ Share name:

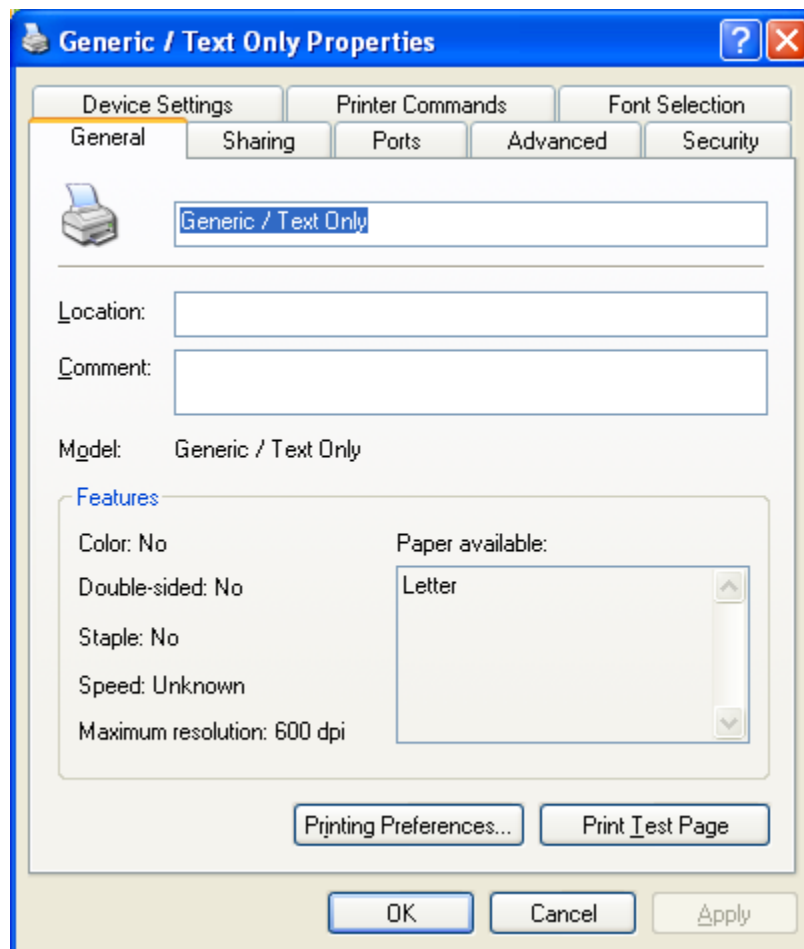


If there is error stating file cannot be found, point to the appropriate folder.









### Useful WDK references:

1. [Advanced Driver Debugging](#) (PPT by Microsoft)
2. [Windows Device Driver Development](#) (PDF – simplified steps)
3. [WDK Build Utility Reference \(MSDN\)](#)