DDKWizard v1.2.0a Manual

How to use DDKWizard

- Setup of `DDKBUILD`
- Usage of the wizard
- Configuration of the wizard
- Tech-speak ...
- Frequently Asked Questions (FAQ)
- Deutsche Informationen

May the source be with you, stranger ... ;-)

CREATED: 2008-08-15
AUTHOR: Oliver Schneider

`$Id: DDKWizard_Help.tex 29 2008-06-27 00:11:35Z oliver $`

Preface

DDKWizard was written **because I can be very lazy**, plain and simple!

It is a laziness that is so irrational if you think about it twice. Programmers tend to spend hours to solve a problem *generically*™ instead of solving it specifically and tailored to the specific problem within a few minutes (but potentially over and over again).

When I started using OSRs `DDKBUILD.BAT` script[1] I soon thought I could rewrite it to make use of all these neat tricks that are allowed by the NT script interpreter[2]. You can just download the original version (`.bat`) as well as the rewritten one (`.cmd`) from OSR Online[3] or from the DDKWizard website at `ddkwizard.assarbad.net`. Soon, however, the need for generic `DDKBUILD` project creation became "overwhelming" ...

As a side-note: There is another[4] `DDKBUILD` script from Mark Roddy at `www.hollistech.com` that does the same as the two scripts from OSR. However, I did not take the time to check the compatibility issues that may arise from usage of DDKWizard together with this `DDKBUILD` "flavor". If someone ever does it, please let me know and I will probably modify DDKWizard to incorporate the necessary settings. Currently it relies on the fact that the batch and the NT script versions from OSR take the same arguments, so using it with "the other" `DDKBUILD` is not recommended, but of course you can try it if you are curious.

Compatibility

DDKWizard works with the DDKs of Windows 2000, Windows XP, Windows 2003 Server and newer WDKs (Windows Driver Kits). It does not include any configurations for WDF projects and may never include them in future. I'll just wait for feedback before deciding about it.
It is compatible with Visual Studio .NET, Visual Studio .NET 2003, Visual Studio 2005, Visual Studio 2008 and the express editions of Visual C++ 2005 and 2008. The "normal" Visual C++ versions are supposed to work fine as well. Refer to section 2.5 and section 3.1 for some possible limitations depending on your Visual Studio version.
Visual Studio 2005 and 2008 (and its flavors) support all DDKWizard features!

Prerequisites

`DDKBUILD`**:** First of all you need to set up your copy of `DDKBUILD` according to the version that you downloaded. It may sound strange (and "selfish"), but I recommend to use the NT script

[1] Which was a traditional batch script.
[2] The NT script interpreter goes indeed far beyond what batch provides!
[3] You need to be logged on to download the files.
[4] ... as far as I know the continuation of the very first `DDKBUILD` script.

(`.cmd`) version since it provides extra functionality (see subsection 1.2.1) which can be handy sometimes. Setup of the batch version is explained in section 1.1, while the NT script version is explained in section 1.2.

Laziness: You need to be extraordinarily lazy[5] to even install this wizard! If you are not lazy enough, that is below 7 on a laziness scale from 1 to 10, you must not use this wizard. **Warning: having this wizard installed on your workstation in your company will be a clear hint to your boss that you are lazy. So be careful when installing and using it ;-).**[6]

How to use this manual

If you are viewing this document in a PDF viewer, you can take advantage of the clickable elements such as links to external websites and links between chapters and sections - click the chapter, section or footnote numbers to see the referenced item. Also if you visit the *Contents* at the end of this document you can click the items there (not the page numbers, though). This should make it easier for you to navigate the document and follow back and forward references which occur every now and then — yet it leaves the document in a state where it can be printed out conveniently.
Kindly note that I use the terms WDK and DDK interchangeably. Unless I want to refer to a specific version, I try to stick to the traditional term DDK.

Do you like DDKWizard? Donations & contributions

If you like DDKWizard, please consider donating to any charity organization you like and trust. But don't feel forced, I wrote it primarily to satisfy my own laziness and the laziness of others ;-).

Also, you will certainly help a lot by providing feedback about possible bugs, feature requests and of course praise. **Contributions (new templates, better source files in existing templates) are *very* welcome.**

Website

The website of DDKWizard is at `ddkwizard.assarbad.net`. There you will find this documentation for download as well as the latest version of the templates and their installer.

An advice

Nah, of course this is not "adware". I'd just like to point out how much I liked the two seminars with Mark from OSR and send some greetings to him and the other guys over at OSR. I like the published books, the mailing lists, *The NT Insider* and their commitment to the NT driver community! If you have to work with drivers professionally you *have* to take at least one of their seminars[7] - **you have to!**

[5] And for those who didn't get it just yet - yes, this is a joke.
[6] Luckily my current employer accepts this form of laziness in exchange for the increased productivity ;-).
[7] ... and not be too shy to ask a lot of "stupid" questions ;-)

License

The following license applies to all parts of DDKWizard, *except for any actual project templates*.

```
Copyright (c) 2006-2008 by Oliver Schneider (assarbad.net)

This software is provided 'as-is', without any express or implied warranty. In
no event will the authors be held liable for any damages arising from the use
of this software.

Permission is granted to anyone to use this software for any purpose, including
commercial applications, and to alter it and redistribute it freely, subject to
the following restrictions:

  1. The origin of this software must not be misrepresented; you must not claim
     that you wrote the original software. If you use this software in a product,
     an acknowledgment in the product documentation would be appreciated but is
     not required.

  2. Altered source versions must be plainly marked as such, and must not be
     misrepresented as being the original software.

  3. This notice may not be removed or altered from any source distribution.
```

The *included* project templates are all in the `PUBLIC DOMAIN`, but the same disclaimer, as stated in the above license, applies:

```
This software is provided 'as-is', without any express or implied warranty. In
no event will the authors be held liable for any damages arising from the use
of this software.
```

Actually the files, once you create a project with DDKWizard, will contain your copyright if you configure the values for `Author` and `CompanyName` as described in subsection 3.2.3 and you have at least Visual Studio 2005.

Chapter 1

Setup of DDKBUILD

1.1 DDKBUILD.BAT

This is the classic. DDKBUILD.BAT is easily installed by following these few steps:

1. Download DDKBUILD.BAT from this page or ddkwizard.assarbad.net.
2. Copy DDKBUILD.BAT into a directory where you want it to reside. I usually choose a directory that can be accessed from anywhere else. Let us say you have the DDKs for Windows XP and 2003 installed and they are located in C:\WINDDK\2600 and C:\WINDDK\3790.1830 respectively. Now you could copy DDKBUILD.BAT into the common root folder of your DDKs, in our case C:\WINDDK.
3. Append the path to the directory where DDKBUILD.BAT resides to the PATH variable of either your user account or to the system-wide PATH variable. In either case right-click *My Computer* and choose *Properties*. Then go to the *Advanced* page and click *Environment Variables*. There you can add or edit the PATH variable of the system[1] or your user account. Note that the values for the variable are delimited by semi-colons (";"). Figure 8.1 shows the dialog. Leave the dialog open for item 5.
   Leaving out this step will still enable you to use DDKBUILD.BAT from within your IDE but will prevent you from using it conveniently from the command line.
4. Start up your Visual Studio[2] IDE and from the *Tools* menu choose *Options* and in the left navigation pane choose the folder *Projects and Solutions* and then *VC++ Directories*. From the combobox *Show directories for* choose *Executable files* and add the path of the directory where DDKBUILD.BAT resides here as well. Figure 8.2 shows the dialog.
5. Depending on the DDKs you have installed, you should set the variables[3] (using the dialog opened before, in item 3) with the following names to point to the respective directories[4]:
   - **Windows 2000 DDK:**
     For this DDK set the variable W2KBASE to point to the base directory of your Windows 2000 DDK.
   - **Windows XP DDK:**
     For this DDK set the variable WXPBASE to point to the base directory of your Windows XP DDK. For the example in the first point this would be C:\WINDDK\2600.

[1] ... if you are an administrator.
[2] Referring to Visual Studio I mean specifically Visual C++.
[3] It is your choice whether you set the system-wide variables or those for your user account.
[4] For the NT4 DDK you can use the NT4BASE variable. However, this is not supported by DDKWizard.

- **Windows 2003 Server DDK:**
  For this DDK set the variable `WNETBASE` to point to the base directory of your Windows 2003 Server DDK. For the example in the first point this would be `C:\WINDDK\3790.1830` (which would be a typical path to the 2003 SP1 DDK).
- **Windows Vista/Windows 2008 Server WDK:**
  For this DDK (formerly LDK, now WDK) set the variable `WLHBASE` to point to the base directory of your WDK. The builds 6000 and 6001.18000 are supported at the time of this writing.

The *exact* version of the DDK does not matter (i.e. 2003 SP1 DDK counts as 2003 DDK). Note that you can only use one variable setting per user/system. DDKWizard attempts to work around this problem, which will be covered in section 2.5. If you are curious which DDK supports which target platforms, take a look into the FAQ.

1.2 DDKBUILD.CMD

To be honest the configuration of `DDKBUILD.CMD` is just identical to the one of `DDKBUILD.BAT` (covered in section 1.1) except for the name of the file. So to spare me the pain in my fingertips after too much typing, just imagine that I wrote `DDKBUILD.CMD` instead of `DDKBUILD.BAT` and turn to section 1.1 for a reference how to install it.

1.2.1 But, DDKBUILD.CMD is even cooler ;-)

Yeah, I know the guys at OSR are going to hate me for this, but since I never got `BINPLACE` to work properly[5] I decided to add what I consider a very useful feature in `DDKBUILD.CMD`: build scripts.

Build scripts? That is supposed to be something new? Well, no. Of course it is nothing revolutionary enough to be patented. I like it nevertheless.
If you put the following three files into the directory where your `SOURCES`/`DIRS` file[6] resides, they will be executed by `DDKBUILD.CMD` automatically. If a file does not exist, the respective feature is being turned off. The files are:

- `ddkbldenv.cmd` is being searched and started first. Use it to override any of the environment variables that `DDKBUILD` normally expects to be set! It is recommended you also read section 1.2.2.
- `ddkprebld.cmd` for any commands you'd like to execute before the `BUILD` utility of the used DDK is being called.
- `ddkpostbld.cmd` for any commands you'd like to execute after the `BUILD` utility of the used DDK has been called.

As I said, it is not revolutionary, but since you can now use the variables declared by both `DDKBUILD`[7] and `SETENV.BAT`[8] you have a wealth of information at your hands that you can use to access the files that have been built - or modify header files in advance, e.g. to increase the build number of your binary.

[5] ... even with the respective variables cleared.
[6] ... or `DIRS` file. Only one set of these scripts is allowed per invocation of `DDKBUILD`.
[7] This part would be specific to the `DDKBUILD` you chose, currently only `DDKBUILD.CMD` supports this, though.
[8] ... the script used by the respective DDK to prepare a proper build environment with all the environment variables set ...

1.2.2 Auto-detection and global variables in DDKBUILD.CMD

Version 7.0 of DDKBUILD.CMD and its numerous beta versions and release candidates had introduced a dumb auto-detection of the path to the requested DDK. As the DDK is defined by the first parameter to DDKBUILD it can start making some good guesses as to where it would possibly find the respective DDK. However, this allows only some rather dumb guesses. With version 7.1 of DDKBUILD.CMD, I introduced a much enhanced version of this auto-detection mechanism. It works like this:

1. It detects whether reg.exe is available on the system.
2. Now everything below this point applies only if you have not set the environment variable corresponding to the DDK base directory as described in section 1.1.
3. If reg.exe is available, it tries to find the install keys for the requested DDK in reverse order. This means that if you requested one of the -WLH* builds, it would first search for the latest version it is aware of and then for the next older one until it finds one. Once found, it returns the path of the newest DDK.
   Note: the detection takes into account that there are 32bit and 64bit Windows editions and handles them properly.
4. If the auto-detection through registry didn't work out, DDKBUILD will fall back to the old "brute-force" detection method that had already been included before version 7.1.
5. If neither auto-detection worked, the script will kindly remind you to set the base directory variable and even point out which one to set.

If you want to make sure to use a particular build of the DDK(s) you installed, make sure to set the base directory environment variable corresponding to your DDK. The auto-detection feature is mainly there to help pinpoint the problem if you forgot to set the variable properly.

Base directory environment variables

If you don't like the idea of having a global environment variable that is supposed to cater all your projects, but instead want to be able to build, for example, one project with the WDK build 6000 and another one with build 6001.18000 without changing a global environment variable all the time, I suggest you really use DDKBUILD.CMD over any other flavor and use the build script ddkbldenv.cmd (see subsection 1.2.1) in your projects to set this variable on a project-to-project basis.
In fact nothing keeps you from overriding the globally set environment variable inside ddkbldenv.cmd to get this level of customization. You can still keep your global variable pointing to the newest DDK (or whichever DDK you like) in parallel. DDKBUILD.CMD is all about choices and versatility.

Chapter 2

Usage of the wizard

2.1 Installing the wizard

To install DDKWizard you should have Windows, Visual Studio .NET, .NET 2003, 2005 or 2008[1] and a few clicks. Since laziness is one of the pillars on which this software was built, the installer is as easy as can be. If the installer finds Visual Studio to be installed it will install the project wizard for the found version(s). As far as I know version 2003 and 2005 don't always go well together, but the installer is even prepared to handle this case.

The first page shows the license terms and a disclaimer to which you have to agree in order to proceed. The next page lets you choose the location to which the main files will be installed (usually `%ProgramFiles%\DDKWizard`). After clicking *Next* the installer will either quit with an error if no supported Visual Studio installation could be found or show the final page which you can close then.

After these few steps you can start the VS IDE (actually it won't hurt if it was already open) and try to create a new project. Depending on the version this is either `Shift+Ctrl+N` (2003) or `Ctrl+N` (2005 and 2008) in the default configuration.
You will notice a new folder *DDK Project* in the left pane and you can choose the type of project you want to create. As of this writing the only supported project types are *Driver*, *NativeApp*, *EmptyDrv*[2] , *Win32CUI* (for `DDKBUILD` Win32 console projects) and *Win32DLL*, although I already plan to include maybe two more specific driver types such as *Display Mirror Driver* and *FS Driver*. The biggest difference would be the contents of the files created for the project.

From the right pane choose the type of project to create and set the name of the project and solution as well as the other options at the bottom.

DDKWizard does not leave any footprint in the registry except for the entry into the *Add or Remove Programs* list.

2.2 Uninstalling the wizard

Who needs this anyway?

[1] ... only Visual C++ instead of the full-fledged Visual Studio suite will be enough as well.
[2] This is currently the only project type that cannot be build straight after creation.

Okay, even if you choose this option, I am not going to cry, I promise ... ;). To uninstall DDKWizard go to the *Control Panel* and from *Add or Remove Programs* choose DDKWizard to remove it from your system. The uninstaller should simply remove the copied files and quit.
Warning: If you ever store any files inside the directories that have been created by DDKWizard these files are going to be deleted upon uninstallation of DDKWizard. So if you want to keep these files, create a backup of them manually *before*!

2.3 Wizard project types

As of this writing the wizard for all project types provides the same options, except for *EmptyDrv* which provides some less. But these options will not be shown anyway. The configuration is very simple and I am just going through all the options and give a brief description for each of them:

- Choose the DDKBUILD you want to use (default: `DDKBUILD.CMD`):

  This option allows you to choose between `DDKBUILD.BAT` (section 1.1) and `DDKBUILD.CMD` (section 1.2).

- Which DDK(s) do you want to use (default: `Windows 2003 DDK + WDK`):

  This allows you to choose multiple DDKs from the list. A project will be created for every DDK you choose and prefixed with the handle of the DDK.
  Value name: `listbox.DDK_CHOICE`

- Support 64bit builds (default: `off`):

  This option allows you to create project configurations for 64bit build types. Note, that starting with version 1.0.3 you can choose which of the two 64bit processor platforms - AMD (default: `on`) and Intel (default: `on`) - you want to use.
  Value name: `checkbox.SUPPORT64BIT`
  Value name: `checkbox.SUPPORT64BIT_AMD64`
  Value name: `checkbox.SUPPORT64BIT_IA64`

- Create PREfast configuration (default: `on`):

  This option allows you to create cloned project configurations that are preconfigured for a PREfast[3] build. Note, that starting with version 1.0.3 you have the option to create the PREfast clone project configuration only for checked build configurations (default: `on`).
  Value name: `checkbox.PREFASTCONFIG`
  Value name: `checkbox.PREFASTCONFIG_ONLYCHECKED`

- Stricter type checking (`.cpp`) (default: `on`):

  This option allows you to create `C++` files instead of `C` files. This causes the compiler to perform stricter type checking and allows to use the syntax of `C++` (e.g. `C++`-style type-casts[4], declare variables just before you use them or create operators to convert between `WCHAR*` and `UNICODE_STRING`, for example).
  Value name: `checkbox.STRICTERCPP`

- Auto-increment build number (default: `on`):

  This feature is only available if you use the `DDKBUILD.CMD` script.
  It will create another `.cmd` script that will automatically increment the build number with every build. It is a matter of convenience so you can turn it on or off at will.
  Value name: `checkbox.AUTOINCBUILDNR`

[3] If you do not know PREfast, read on it. This is definitely a must-use for any driver writer. It is also available for user mode development starting with the Windows Vista SDK.
[4] ... which I personally prefer ...

- Create pre/post build scripts (default: on):

  This feature is only available if you use the DDKBUILD.CMD script.
  If you turn it on, three sample scripts will be created as described in subsection 1.2.1. This option is mandatory if you checked the previous option ("Auto-increment build number").
  Value name: checkbox.BUILDSCRIPTS

- Create makefile.inc (default: off):

  This will just include a makefile.inc into your SOURCES file into which you can place other custom build steps. The respective option in the SOURCES file is NTTARGETFILES.
  Value name: checkbox.MAKEFILEINC

- Create message file (.mc) (default: off):

  This creates a template message file which is integrated into the created project. That is, it will be built and linked into the binary once you build the project.
  Value name: checkbox.MESSAGEFILE

- Use property sheet (default: on):

  This option and the following ones are only available to users of Visual Studio 2005 (and later).
  This option creates a property sheet that is being inherited by the projects created. This allows you to manipulate the behavior of the project by modifying some default options. Also environment variables can be declared. By default the variable DDKBUILD_PATH is declared.
  Value name: checkbox.PROPERTYSHEET

- Override (full) DDKBUILD path (default: off):

  If you enable this option, you can override the value of the environment variable DDKBUILD_PATH for the created projects. This allows you for example to customize DDKBUILD and point the projects to your own version of DDKBUILD[5].
  Note that you have to give the full path including the filename of your DDKBUILD script.
  Value name: checkbox.OVERRIDEDDKBUILD and text.OVERRIDEDDKBUILD

- Override W2KBASE (default: off):

  Similar to the previous option this allows you to override an environment variable. In this case the variable W2KBASE used by DDKBUILD to locate the Windows 2000 DDK.
  Value name: checkbox.OVERRIDEW2KBASE and text.OVERRIDEW2KBASE

- Override WXPBASE (default: off):

  Similar to the previous option this allows you to override an environment variable. In this case the variable WXPBASE used by DDKBUILD to locate the Windows XP DDK.
  Value name: checkbox.OVERRIDEWXPBASE and text.OVERRIDEWXPBASE

- Override WNETBASE (default: off):

  Similar to the previous option this allows you to override an environment variable. In this case the variable WNETBASE used by DDKBUILD to locate the Windows 2003 Server DDK.
  Value name: checkbox.OVERRIDEWNETBASE and text.OVERRIDEWNETBASE

- Override WLHBASE (default: off):

  Similar to the previous option this allows you to override an environment variable. In this case the variable WLHBASE used by DDKBUILD to locate the Longhorn/Vista DDK (aka WDK).
  Value name: checkbox.OVERRIDEWLHBASE and text.OVERRIDEWLHBASE

[5] DDKWizard still makes the assumption that the options are compatible with OSRs DDKBUILD versions.

As you can see the last few *value names* appear in pairs. This is due to the fact that you have to check the checkbox in order to override one of the text fields in the dialog. So naturally the same holds for the customization of DDKWizard through the configuration file.

2.4 Projects and configuration created by the wizard

DDKWizard creates a project for each DDK you choose in the options. So if you choose to create configurations for the Windows 2000 and XP DDKs, for example, DDKWizard will create two projects with the names `<ProjectName>.W2K` and `<ProjectName>.WXP` respectively[6]. Each of these created projects will contain the configurations for the specific DDK and the solution will contain a *solution configuration* for each of the *project configurations* in all projects.

The *solution configurations* are pre-configured to include only the project which is relevant to the current (solution) configuration. Therefore it is irrelevant which one of the projects you choose as startup project - the important setting is the *solution configuration* that you select. The following solution configurations can occur depending on your choice of DDKs.

- `W2K free` and `W2K checked`:
  Use the Windows 2000 DDK to build a 32bit driver with Windows 2000 as the target operating system platform.
- `W2K64 free` and `W2K64 checked` (only if 64bit support was activated):
  Use the Windows 2000 DDK to build a 64bit (Intel Itanium) driver with Windows 2000 as the target operating system platform.
- `WXP free` and `WXP checked`:
  Use the Windows XP DDK to build a 32bit driver with Windows XP as the target operating system platform.
- `WXP64 free` and `WXP64 checked` (only if 64bit support was activated):
  Use the Windows XP DDK to build a 64bit (Intel Itanium) driver with Windows XP as the target operating system platform.
- `WXP2K free` and `WXP2K checked`:
  Use the Windows XP DDK to build a 32bit driver with Windows 2000 as the target operating system platform.
- `WNET free` and `WNET checked`:
  Use the Windows 2003 Server DDK to build a 32bit driver with Windows 2003 Server as the target operating system platform.
- `WNET64 free` and `WNET64 checked` (only if 64bit support was activated):
  Use the Windows 2003 Server DDK to build a 64bit (Intel Itanium) driver with Windows 2003 Server as the target operating system platform.
- `WNETXP free` and `WNETXP checked`:
  Use the Windows 2003 Server DDK to build a 32bit driver with Windows XP as the target operating system platform.
- `WNETXP64 free` and `WNETXP64 checked` (only if 64bit support was activated):
  Use the Windows 2003 Server DDK to build a 64bit (Intel Itanium) driver with Windows XP as the target operating system platform.

[6] Where `<ProjectName>` is the name of the project you chose in the *New Project* dialog.

- `WNETAMD64 free` and `WNETAMD64 checked` (only if 64bit support was activated):
  Use the Windows 2003 Server DDK to build a 64bit (AMD64, x64, EM64T) driver with Windows 2003 Server as the target operating system platform.
- `WNET2K free` and `WNET2K checked`:
  Use the Windows 2003 Server DDK to build a 32bit driver with Windows 2000 as the target operating system platform.
- `WLH free` and `WLH checked`:
  Use the Longhorn/Vista WDK to build a 32bit driver with Windows Longhorn/Vista as the target operating system platform.
- `WLH2K free` and `WLH2K checked`:
  Use the Longhorn/Vista WDK to build a 32bit driver with Windows 2000 as the target operating system platform.
- `WLHXP free` and `WLHXP checked`:
  Use the Longhorn/Vista WDK to build a 32bit driver with Windows XP as the target operating system platform.
- `WLHNET free` and `WLHNET checked`:
  Use the Longhorn/Vista WDK to build a 32bit driver with Windows 2003 Server as the target operating system platform.
- `WLHNETI64 free` and `WLHNETI64 checked` (only if 64bit support was activated):
  Use the Longhorn/Vista WDK to build a 64bit (Intel Itanium) driver with Windows 2003 Server as the target operating system platform.
- `WLHNETX64 free` and `WLHNETX64 checked` (only if 64bit support was activated):
  Use the Longhorn/Vista WDK to build a 64bit (AMD64, x64, EM64T) driver with Windows 2003 Server as the target operating system platform.
- `WLHI64 free` and `WLHI64 checked` (only if 64bit support was activated):
  Use the Longhorn/Vista WDK to build a 64bit (Intel Itanium) driver with Windows Longhorn/Vista as the target operating system platform.
- `WLHX64 free` and `WLHX64 checked` (only if 64bit support was activated):
  Use the Longhorn/Vista WDK to build a 64bit (AMD64, x64, EM64T) driver with Windows Longhorn/Vista as the target operating system platform.

As you can see these names follow the naming of the options for `DDKBUILD` and the list of configurations can get quite long. Therefore I recommend to select only the DDKs which you are definitely going to use.

Please note as well that, depending on your selection to create PRE*f*ast configuration clones, one additional configuration will appear per above described configuration with the hint "PRE*f*ast" in its name.

2.5 Property sheets (VS 2005 or later)

With Visual Studio 2005 Microsoft introduced a neat feature called *Property Sheets*. This is no more than a file which provides default settings for the build tools of a project. The specific feature DDKWizard makes use of is the *Macros* setting. There you can set (or *override* global) environment variables for the current project. Now that I mentioned it, many of you are already saying ... "aha

so I can ..." ... yep, you can do almost anything. Finally you can create custom projects for a DDK which is useful if the DDK is checked into your version control system. I had never before considered version control systems for such a use, but I was taught this approach at the `NTDEV` mailing list.
At least if you use Visual Studio 2005 or later I recommend you let the respective checkbox checked to include support for *Property Sheets*. The property sheet that will be included in the project is named *`ProjectName`*`.vsprops` and contains just the declaration of a variable named `DDKBUILD_PATH` in the default configuration[7].

Leaving the option active will give you more flexibility during the course of your project. For example if you ever decide that you need to customize one of the environment variables you will be able to do so without much effort. The only thing you need to do is going to the *View* menu, choosing *Other Windows* and then *Property Manager* which provides a convenient way to customize the already-included property sheet.

This is the method used by DDKWizard to overcome certain limitations such as that only one environment variable per user/system can be declared. Also the grouped overriding options work through this method.

But I only have Visual C++/Studio .NET/2003?!

Worry not. Once again `DDKBUILD.CMD` comes to the rescue. You can use `ddkbldenv.cmd` as described earlier in subsection 1.2.1 and section 1.2.2 to achieve similar results.
Also do not underestimate the use of this additional build script. While property sheets are a pretty powerful feature, they will only work *inside* Visual Studio. `DDKBUILD` on the other hand will work from the command line or other integrated development environments and consequently will build scripts such as `ddkbldenv.cmd`.

Of course, if you use neither a modern Visual C++/Studio nor `DDKBUILD.CMD`, you are definitely not going to get far with property sheets or `ddkbldenv.cmd`. Nothing *I* could do about this, though.

[7] Yes, the variable `DDKBUILD_PATH` will be set even if you do not override the path to `DDKBUILD`. The default setting will just be `ddkbuild.bat` or `ddkbuild.cmd` according to the chosen version. This allows you to override this setting as well as others at a later point without changing too much in your project's properties, since the property sheet is already included.

Chapter 3

Configuration of the wizard

Starting with version 1.1.0 DDKWizard can be customized by the user. This works via the manipulation of an `INI`-style configuration file that is put into the installation directory of DDKWizard.
I found that it might be a bit strange to force users of DDKWizard to use *my* favorite settings. So I decided to let the users customize their settings. Even better, this will give you the power to customize the behavior of DDKWizard *per project type*.

3.1 Compatibility limitation(s)

There is a major caveat with this new feature when used with Visual Studio .NET and .NET 2003. If your Internet Explorer security zones are configured accordingly (the default), you will always get a nasty message box warning you of an ActiveX control being run[1]. This can be a major nuisance, so I added the means to enable and disable this new feature easily and *it is by default disabled* for all Visual Studio versions prior to 2005! Visual Studio 2005 (or later) users can skip the rest of this section and proceed to the next one.

I am not willing to fix this in any way since Visual Studio 2005 fixed it already. Also I am not willing to include the means to adjust the security zones in the installer as it would not be right in my opinion ... and some system admins might decide to hunt me down and torture and kill me for doing it ;-) ... However, if you are a user of Visual Studio .NET or .NET 2003, you can easily enable the configuration file feature by changing the very last line of the script file "`DDKWizIniFile.js`" so that it reads:

```
var bOverrideOldVsConfig = true;
```

But again, be prepared to get the nasty warning message every time you use DDKWizard then. So to disable it again, replace "`true`" by "`false`" again or simply reinstall DDKWizard.

You find this script file in the installation folder of DDKWizard - which is by default in the program files folder (`%ProgramFiles%\DDKWizard`) but that can be overridden during the installation.

As you can see, the decision is up to you. I for one would *only* enable it if I really need the customization feature and leave it disabled otherwise. Usually DDKWizard would not be used on a daily basis, so, while being a major nuisance to me as the author, it might be a very minor nuisance to you compared to the utility configuration of DDKWizard provides to you.

[1] The control being run is `Scripting.FileSystemObject` which is required to read the configuration file that is the heart of this new feature. So sadly there is no way of replacing this particular code.

3.1.1 Workarounds?!

You might want to work around the problem with Visual Studio .NET and .NET 2003 by simply editing the file `default.htm` which exist in the subfolder of
`%ProgramFiles%\DDKWizard\Wizards\<ProjectType>\HTML\1033`
for each of the project types "`Driver`", "`EmptyDrv`", "`NativeApp`" , "`Win32CUI`" and "`Win32DLL`". Of course you should know best where you installed DDKWizard, so the path may vary depending on your choice (the sample path above is the default location).
The `default.htm` contains a number of `symbol`-tags at the beginning whose `value` can be adjusted according to your needs. Unlike customizations in the actual configuration file, reinstalling DDKWizard or upgrading/updating it will cause *these* adjustments in `default.htm` to be lost (see section 3.3). So use this method carefully!

3.2 The `INI` file explained

In section 2.3 you will find a comprehensible list of all the options. Along with the explanation of every option, you will find its *value name* inside the configuration file given.
But let us start with the general overview of the `INI` file (this is how I will call the configuration file from now on). Here are a few simple hints.

- Any section name can occur more than once.
- Any *value name* that is not recognized will be ignored.
- The last value for a given *value name* will override any previous setting for this *value name* in the same section.
- Section names *are* case-sensitive!
- Comment lines start with a semi-colon: "`;`". There may be an arbitrary number of blank spaces or tabs before that semi-colon for formatting purposes.
- There are no multi-line comments.
- The default `INI` file will give you an overview of all available *value names* and sections.
- Lines with empty values are ignored!

3.2.1 Special values

There are special rules for the values. A value of "`default`" means that the respective setting will not be overridden! This also means that text fields or user variables can never be set to "default" as a result of the customization[2].

3.2.2 Value types

There are four types of values:

- **The user variables** are simple text fields and can contain virtually any text. The same holds for values **whose value names start with "`text.`"**. For both of them, the leading and trailing blank spaces (as well as tabs) are removed from the value. This might be of importance for troubleshooting, but I consider it a minor nuisance, so this is a non-issue for me and needs no bug-reporting.

[2] Basically the same as for empty values.

- **Value names that start with "`checkbox.`"** can have exactly three valid values: `default`, `true` and `false`. The first has been explained and the latter two are easily explained with the state of the checkbox which is being overridden. So `true` will cause the checkbox to become checked and `false` means the checkbox will be unchecked. Any other value will be ignored.
- **Value names that start with "`listbox.`"** can have special meanings, so each of the "`listbox.`" values will be separately explained in subsection 3.2.5 *ff.*

3.2.3 Section `[General]`

There is one `INI` section `[General]`, which contains settings that override what I call "user variables". These will be replaced in the template files when the new project is created. To give you an example - there is a value called `Author`, which will put the author's name into the file headers of source files. The same holds for the `CompanyName`[3], which will also replace the respective placeholder value in the file headers. `Website` allows you to give a website that will be used in the version resource of the created binaries.
Currently these are the only supported "user variables" inside this `INI` section.

3.2.4 `[Project.*]` sections

Each of the following section names is valid as well and overrides the default dialog settings for the respective project type:

- `[Project.Driver]`
- `[Project.EmptyDrv]`
- `[Project.NativeApp]`
- `[Project.Win32CUI]`
- `[Project.Win32DLL]`

Note, that the values you override here will have influence on the default settings of the DDKWizard dialog for the respective project types, but they will not allow to override "user variables" on a per-project-type basis. "User variables" are always global.
All the values and their meaning are explained in section 2.3. So once you start customization, the default values given there are nonsense, of course.

3.2.5 *Value name* `listbox.DDK_CHOICE`

This is one of the value names prefixed with "`listbox.`" and therefore needs further explanation. Note, that this will change over time if new DDKs get supported by DDKWizard.

Currently this value is a comma-separated list of numbers from zero to three. Each of these values stands for one of the supported DDKs, as follows:

- `0` = Windows 2000 DDK
- `1` = Windows XP DDK
- `2` = Windows 2003 DDK

[3] This is the value that appears right after the "by" in the copyright notice. So it is not necessarily a company name for you.

- `3` = Windows Vista/Windows 2008 Server WDK

So if the following line appears for one of the project types "`listbox.DDK_CHOICE = 0,2`" this means that the Windows 2000 and 2003 DDKs will be preselected for the respective project type. If you want DDKWizard to force none of the DDKs to be selected, use the following setting "`listbox.DDK_CHOICE = none`". Practically any value that is not a valid comma-separated list of numbers and not empty or "`default`" will have the same effect as "`none`". This is due to the fact that empty values are ignored by the `INI` parser code.

3.3 Is the configuration persistent?

Yes, once the `INI` file is being put into the installation folder, you will be asked the next time whether you want to overwrite your (possibly customized) `INI` file with a new default one or keep your (modified) version if you install a newer version of DDKWizard over an older one or reinstall the current version.
So don't be afraid, your settings will be preserved - given you choose to keep your modified file ;-) ... However, the uninstaller *will remove* the whole installation directory. So if you intend to install DDKWizard again, please make a manual backup of the configuration file in a safe place *before* you start the uninstaller!

Chapter 4

Tech-speak ...

4.1 Custom wizards

Frankly speaking the custom wizard created by Visual Studio - if you choose a to create a new project of just this type - is horrible. The HTML that is being created is so ugly and so far from any standard that I basically replaced all of it except for the symbols in the beginning and the JScript parts.

4.2 `./HTML/1033/default.htm`

This is the HTML file which is being displayed if you choose to create one of the project types supplied by DDKWizard. It contains several symbols as well as all the logic of which options to enable/disable according to the version of the Visual Studio IDE and in relation to the other options.

Most symbols are *statically* defined. One which is added *dynamically* is `DDK_LISTBOX` because I found no other way that would allow me to provide a multiple choice listbox than to read the selected entries once the user clicks *Finish*. The other symbols coincide with the options presented in section 2.3 for each project type.

4.3 `./Scripts/1033/default.js`

As all programmers do for their spare-time projects, I'll supply the comments if I ever find time :o) ...

In fact you will find some comments in the source, already.

Chapter 5

Creating your own custom project templates (and contribute them)

If you like to create custom project templates, the best starting point is to pick one of the existing templates from DDKWizard and modify them. In any case you will have to register the new templates with each version of Visual Studio or Visual C++.

5.1 Registration files

For the sake of brevity, I will assume you have Visual Studio 2005 and that it is installed inside `%ProgramFiles%\Microsoft Visual Studio 8`. The other versions of Visual Studio are very very similar in this respect, so you need no more than common sense to figure it out.
I will call the installation directory just `<VSDIR>` from now on to make it clearer where the respective *subfolders* are located.

5.1.1 `YourTemplate.vsz` - where are your template files?

Inside `<VSDIR>\VC\vcprojects\DDK Project` create a new `.vsz` file or copy an existing one. Inside there you will have to change the values `WIZARD_NAME` and `ABSOLUTE_PATH` to fit your environment (and taste, in case of the name). Here is how it looks for my `EmptyDrv.vsz`:

```
VSWIZARD 7.0
Wizard=VsWizard.VsWizardEngine.8.0
Param="WIZARD_NAME = EmptyDrv"
Param="ABSOLUTE_PATH = C:\Program Files (x86)\DDKWizard\Wizards\EmptyDrv"
Param="FALLBACK_LCID = 1033"
```

5.1.2 `YourWizard.vsdir` - registering with Visual Studio

Note: this refers to `<VSDIR>\VC\vcprojects\DDK Project` as well.
You will have to adjust the file `DDKWizard.vsdir` to contain some information to be shown in the project creation dialog. You will have to figure out the format of the file yourself, though.

5.2 Your templates

Now we come to the interesting parts. You may have guessed it already: templates are usually simple text files. However, there is more than simple copying involved. If you have some file(s) in

the directory you specified as `ABSOLUTE_PATH`, as was described in subsection 5.1.1, you will have to navigate your file manager to this location. Otherwise it is now time to create the specified path and copy files from the existing DDKWizard template or create new ones.
Into this folder copy any and all files you want to have in the new projects (apart from the `.vsproj` files, of course). There is only one thing to keep in mind. Whenever you call the file `PROJECT_NAME.somthing` it is going to be called like the project with the suffix (file extension) "`.something`".
Furthermore, text files are subject to some pre-processing when being copied into the folder you selected for the project. See the following section.

5.3 The "user variables"

The pre-processing replaces certain variables which are all defined by DDKWizard or, in case of `DDKWIZ_USERVARIABLE_*` by you, with the content of the respective variables. To achieve this, give the variable in the following form

```
[!output VARIABLE_NAME]
```

Here the list of variables. Please note, that especially the ones starting with `OVERRIDE*` are really of limited use only.

- `DDKBUILD_COMBOBOX`
  The *value* of the combo-box which offers the selection of `DDKBUILD` scripts. Currently only `ddkbuild.cmd` and `ddkbuild.bat`.
- `DDKWIZ_USERVARIABLE_AUTHOR` and `DDKWIZ_USERVARIABLE_AUTHOR_C`
  `DDKWIZ_USERVARIABLE_COMPANY` and `DDKWIZ_USERVARIABLE_COMPANY_C`
  `DDKWIZ_USERVARIABLE_WEBSITE` and `DDKWIZ_USERVARIABLE_WEBSITE_C`
  Values of the respective user variables as defined in the INI file (if you have support for this enabled, i.e. by default not prior to VS 2005).
- `OVERRIDEDDKBUILD_TEXT`
  Name (and optionally path) to `DDKBUILD`. Of course this could be just some third-party compatible script.
- `OVERRIDEW2KBASE_TEXT`, `OVERRIDEWXPBASE_TEXT`, `OVERRIDEWNETBASE_TEXT` and `OVERRIDEWLHBASE_TEXT`
  I can not imagine where these are useful, but they will only be defined if you use Visual Studio 2005 or later and have support for property sheets. They are the base directories for the DDKs/WDKs that can be overridden in the project creation dialog.
- `PROJECT_DATE` and `PROJECT_YEAR`
  The date when the project was created in ISO format (`YYYY-MM-DD`) and the year from the date (useful for copyright notices).
- `PROJECT_FILENAME`
  Name of the main source file, i.e. project name with ending `.c` or `.cpp`.
- `PROJECT_NAME`
  Name of the project without `.vcproj`.
- `PROJECT_PREFIX`
  A prefix that is all uppercase and has been derived from the project name. It will not contain any characters which would be unsuitable for function or variable names unless your project name starts with a number ;-).

Chapter 6

Frequently Asked Questions (FAQ)

- **How to create browse information for a project?**
  In the `SOURCES` file you will find the following line:
  `#BROWSER_INFO=1`
  Just uncomment this line to force `BUILD` to create the browse information for your project. To remove the comment, remove the hash mark (`#`).

- **How to override the browse file path?**
  In your `SOURCES` file set the symbol `BROWSERFILE` to the path. Of course you can incorporate variables there (e.g. `$(O)` - that is the letter "O").

- **How to override or set C defines?**
  Nothing easier than that. Modify the line which starts with `C_DEFINES` in your `SOURCES` file to your liking. The syntax is the same as on the command line, i.e. to incorporate previous settings of `C_DEFINES` and define the symbols `UNICODE` and `_UNICODE` use this:
  `C_DEFINES=$(C_DEFINES) /DUNICODE /D_UNICODE`

- **What is the advantage of using DDKWizard-created Win32 projects?**
  The projects created by DDKWizard use a DDK compiler/linker. Luckily, although the newer "Visual" products from Microsoft don't support it anymore, the DDK can link Win32 applications against the MFC 4.x libraries and the original `MSVCRT` libraries. The advantage should be clear now. You will not have to deploy the newer DLLs for the new MFC versions and no updated CRT DLL either. Consequently any program built this way will run - without extra files - on many legacy systems and will also be smaller in size.
  Trivia: the headers in the DDK take the improved standard compliance of the DDK compiler into account. Also a nice way to migrate or just recompile old VC6 projects.

- **What the heck is `CUI`?**
  Some people know it as `CLI` or *Command Line Interface* - I personally prefer `CUI` or *Console User Interface* as opposed to `GUI` or *Graphical User Interface*.

- **I get strange errors, what can I do?**
  The first thing to do would be to check whether you can find the cause yourself. However, if there are errors similar to "file not found" in the output of `DDKBUILD`, the cause could be, that one of the scripts (your customized build scripts or those included in the DDK) don't like long path names. That is, path names which exceed the 8.3 DOS file naming conventions. The DDK tools do not like long file names and appear to get confused occasionally. `DDKBUILD.CMD` addresses this issue with version 7.0RC2 and later. If all fails, you might want to consider contacting me ;-).

- **What DDK (WDK) should I use?**
  In general I would recommend you use the latest available DDK, if possible. However, there are a few limitations. Let us consider your target platform is NT 4.0, your only choice for a DDK which supports it officially, would be the NT4 DDK. However, it is well possible to build drivers compatible with Windows NT 4.0 using a newer DDK and setting Windows 2000 as the target platform.
  Furthermore, all DDKs up to and including the Windows 2000 DDK, where dependent on an external driver. The Windows XP DDK was the first to address this issue. Neither DDKWizard nor `DDKBUILD` offer any ways to make this part easier. In fact the NT4 DDK is deprecated for the newest `DDKBUILD` and was never supported by DDKWizard.
  Here is a list of targets supported by the DDKs along with the base directory environment variable as described in section 1.1:
  - **Windows NT 4.0 DDK** (`NT4BASE`)
    This DDK requires an external compiler and linker. It can be used to compile drivers targeted at the Windows NT 4.0 platform. As far as I am aware, support for all processor platforms that were officially supported by Windows NT 4.0 was included.
  - **Windows 2000 DDK** (`W2KBASE`)
    This DDK requires an external compiler and linker. It can be used to compile drivers targeted at the Windows 2000 platform including x86 and IA-64 ("Itanium").
    Note: To compile drivers for Windows 2000 on IA-64 ("Itanium"), this is your only option.
  - **Windows XP DDK** (`WXPBASE`)
    This DDK can be used to compile drivers targeted at the Windows 2000 platform but only on x86 as well as drivers for Windows XP (x86 and IA-64).
    Note: The Windows XP x64 edition is based on Windows 2003 Server and therefore not supported by this DDK.
  - **Windows 2003 Server DDK** (`WNETBASE`)
    This DDK can be used to compile drivers targeted at the Windows 2000 platform but only on x86 as well as drivers for Windows XP (x86, x64 and IA-64) and Windows 2003 Server (x86, x64 and IA-64). Note: Use the x64 configuration type for Windows 2003 Server to build a driver compatible with Windows XP x64.
  - **Windows Vista/Windows 2008 Server WDK** (`WLHBASE`)
    With this DDK (formerly "Longhorn Driver Kit", now "Windows Driver Kit") you can target almost all of the previously mentioned platforms. Support for Windows 2000 does not include IA-64 ("Itanium"). Note: Use the x64 configuration type for Windows 2003 Server to build a driver compatible with Windows XP x64.
- **Why are you so lazy, Oliver?**
  You use DDKWizard and *still* complain? Tss tss ...

6.1 Important change in version 1.1.1b

In previous versions, DDKWizard-created projects contained a bug. This bug *could* cause a BSOD in the worst case. To make a long story short, I returned a value using the dereferenced pointer from a completed IRP. Not good ;-) ... this is fixed.
Please note that you have to fix the issue by yourself for older projects created with DDKWizard. To do this, check all the dispatch functions of your driver for the following line:

```
return Irp->IoStatus.Status;
```

Note that this `return` statement needs to reference a variable on the stack, not a completed IRP. So the correct version would look something like this:

```
NTSTATUS status; // = whatever default
// ... whatever actions
status = Irp->IoStatus.Status;
// Complete the IRP
return status;
```

Other variants are of course possible, but you get the point.

6.2 Build script use cases

In subsection 1.2.1, I introduced you to the build scripts that can be used by `DDKBUILD.CMD`. Now let us discuss some use cases.
`ddkbldenv.cmd` should be used to override any of the base directory variables, but in fact you could even override some `DDKBUILD`-internal variables. I cannot recommend this, though, since I may change the logic in the main script one day and your `ddkbldenv.cmd` could break in that case. Example (overrides `WLHBASE`):

```
@echo off
set WLHBASE=X:\WINDDK\6000
```

`ddkprebld.cmd` is being called just before the actual call to the `BUILD` utility of the DDK. Therefore you can inject any kind of actions that you need to be done at this point. If it doesn't matter whether at this point or earlier you *may choose between* this script and `ddkbldenv.cmd`. Example (sets a variable used in `SOURCES`):

```
@echo off
set ENGINEBASE=X:\MyProduct\Engine
```

`ddkpostbld.cmd` could be used for any step after the build has finished. Typical tasks at this stage would be to move the binary files to some other location, to sign them or to perform other actions on them.

Chapter 7

Deutsche Informationen

Wie unter 1.1 schon erwähnt, müssen für `DDKBUILD` unter Umständen noch Einstellungen in Visual Studio vorgenommen werden. Deshalb hier die deutschen Namen der entsprechenden Menüpunkte, welche ich ebendort für die englische Version von Visual Studio genannt habe.

Tools wird zu *Extras*

Options wird zu *Optionen*

Projects and Solutions wird zu *Projekte und Projektmappen*

VC++ Directories wird zu *VC++-Verzeichnisse*

Show directories for wird zu *Verzeichnisse anzeigen für*

Executable files wird zu *Ausführbare Dateien*

Chapter 8

Figures

Figure 8.1: Dialog to set environment variables

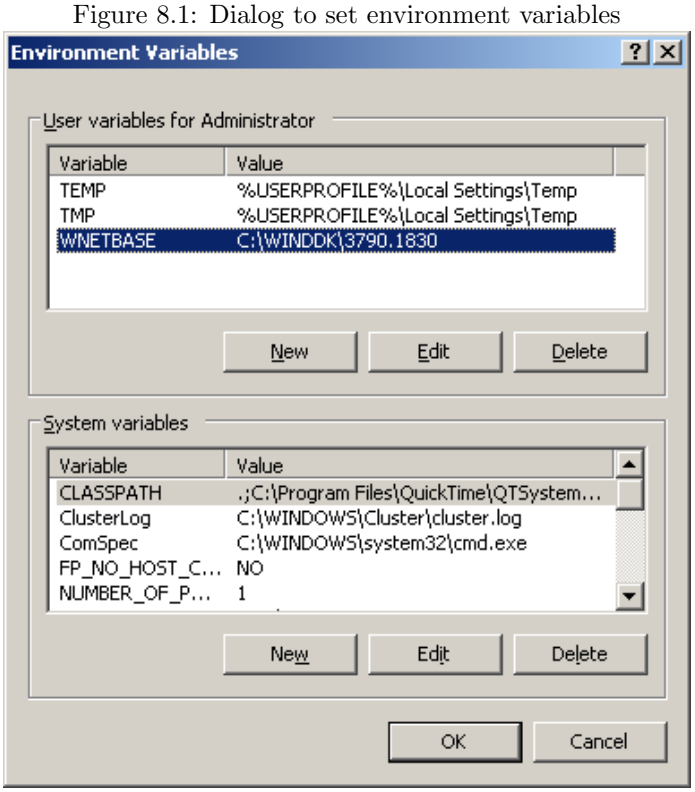
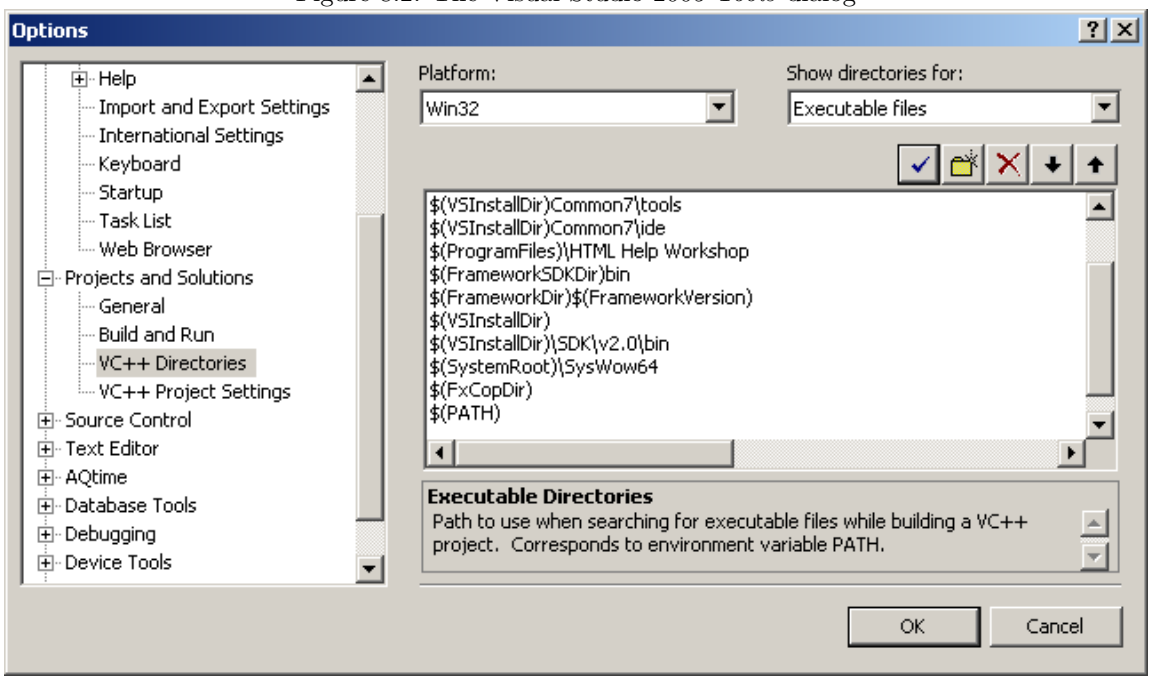


Figure 8.2: The Visual Studio 2005 *Tools* dialog



Contents