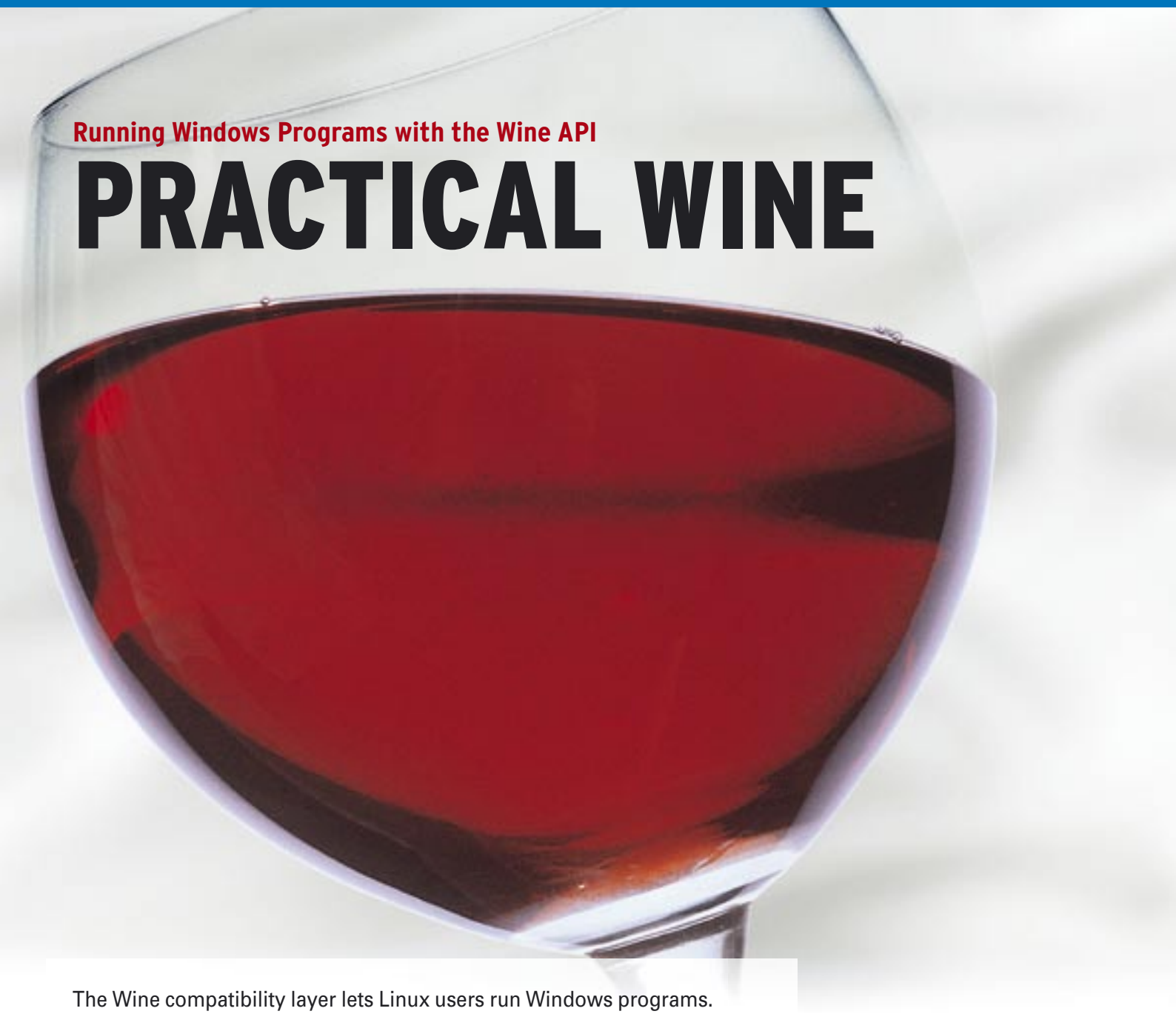

Running Windows Programs with the Wine API

PRACTICAL WINE

The Wine compatibility layer lets Linux users run Windows programs. Unfortunately, configuring Wine is anything but trivial, and it helps if you enjoy experimenting. **BY JOACHIM VON THADDEN**

Linux offers a number of options for users who need to run an occasional Windows program. You can emulate a complete machine using an application such as VMware or Qemu, or you can simulate a machine subset with Win4Lin. These solutions work quite well but also have some drawbacks: for one thing, users need to buy both the emulation environment and the operating system, both of which take up valuable space on the host machine. Emulation also taxes performance and soaks up memory. An average program running in an emulator will achieve just 50 to 80 percent of its normal native performance, even with increased static RAM memory use by the guest operating system.

An alternative to emulation is simply to provide Windows libraries for the Linux system. In theory, this approach would allow the program to run in Linux. Unfortunately, this approach is complicated. First, the Windows library functions, or so called API (Application Programming Interface), is not adequately documented. And in many cases, programs written by Microsoft, in particular, call undocumented functions. As there is a lot of redundancy in the API, one might suspect that Microsoft has actually lost track of the many thousands of system calls and their extremely complex calling parameters. Additionally, some functions are tailored to suit the Windows environment and make many assumptions about the underlying system. For example, the Linux filesystem is massively different from the Windows filesystem. To mitigate this issue, it would be useful to have the ability to use the original Windows libraries. Of course, this means owning a Windows license, but most people have one any-

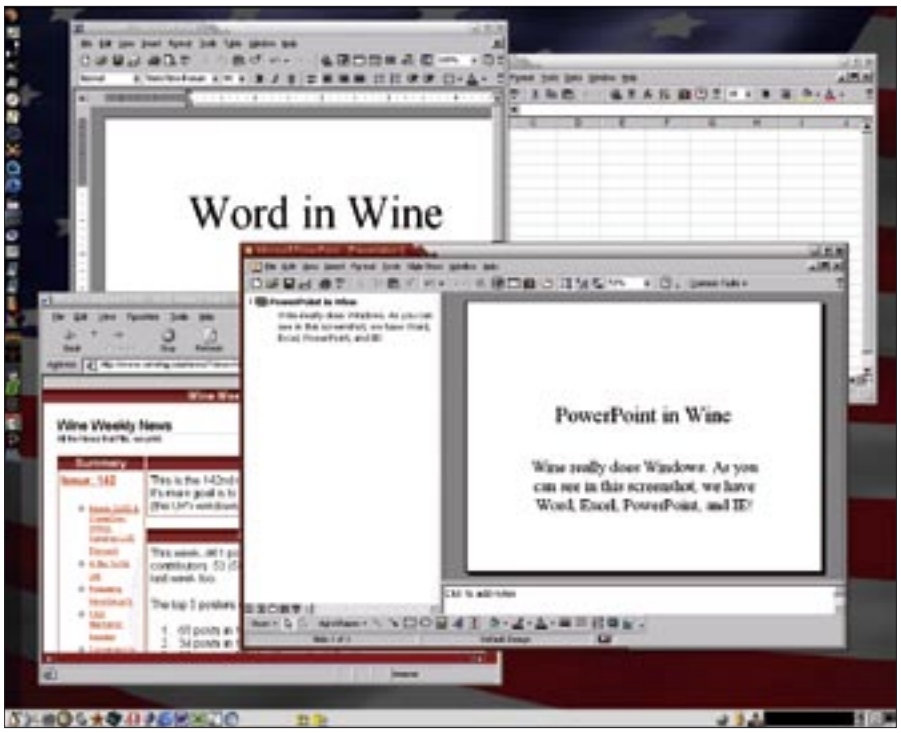

Figure 1: The rewards – complex Windows programs like Microsoft Word show good performance on Linux.

way, so why not do something useful with it?

The open source Wine project provides programming functionality and also integrates original libraries. The recursive acronym "Wine," which stands for "Wine is not an emulator," expresses what Wine truly is: not an operating system emulator, but a compatibility layer. After a long development period, and with support from commercial enterprises, Wine has now reached a stage of development that allows users to run an amazingly large number of Windows programs on Linux. This article takes a practical look at how to use Wine.

Installing Wine

As Wine is subject to rapid development cycles, the version you use is critical. It does not make much sense to go for the latest version; in fact, I would suggest adopting a conservative approach and keeping a working version. Restrict experiments with new systems.

The Wine version that I use myself, 20041019, is not the latest. Wine packages always use the release date as their version number. The October 2004 release is extremely stable and gives me all the functionality I need. The versions released since then have been highly experimental and do not lend themselves to productive use. The latest version at this time of writing, 20050419, looks more promising, but it refuses to cooperate with Microsoft Word.

Fortunately, installing Wine is now child's play. Long-winded build and install processes are a thing of the past for most distributions, as the official Wine Site has matching, and well maintained, packages for Suse, Red Hat, Fedora, Debian, Mandrake, Slackware, and even FreeBSD [1]. Older packages are also available [2]. After downloading the package, users with RPM-based distributions can simply become root and give the following command

```
rpm -Uvh wine-200xxxxx-*
```

to install Wine on their systems. If a previous version gets in the way of the install, typing *rpm -e wine* should remove the offending version. On Fedora, the Prelink security mechanism, which stores prelinked versions of all system libraries, can cause a few issues. If you do not want to disable Prelink completely, first become root, and then, after installing Wine, delete the prelinked libraries as follows:

```
killall prelink;prelink -uav; ➥
rm /etc/prelink.cache; ldconfig
```

As Fedora's Prelinking mechanism checks the libraries every 24 hours, the system will be returned to a safe state after 24 hours at the latest. If you are in a hurry, you can call the *prelink* cron script to reenable the mechanism:

```
killall prelink;➥
/etc/cron.daily/prelink
```

The Wine packages supplied with Debian have gained a certain degree of notoriety for being poorly maintained and highly unpredictable. Debian users will want to deinstall any Wine packages they discover, then change repository and install the original packages from Wine headquarters, as described at [3].

At the Wine Lodge – Wine's Directory System

Wine stores a user's configuration and working files along with that user's Windows programs below the user directory in the *.wine* subdirectory. Be careful if you log on as *root*. Just as Linux system users should only log on as root to perform administrative chores, Wine users should never use privileged accounts. For one thing, Windows programs are not exactly renowned for their security and might compromise your system: for another, system calls that might otherwise fail could work for the root user, and this makes program behavior unpredictable.

You will need a new Wine directory to complete the installation steps; if you have a pre-existing Wine directory, you should rename it just to be on the safe

side. Initializing a new directory is quite simple; just type *wine* at the console. This places the *config* configuration file, and the Windows registry database, which will store files with the *reg* suffix, directly below the *~/.wine* directory. More recent Wine versions do not create a *config* file. The following command:

```
cp /usr/share/doc/wine-➥
20041019/samples/config ~/➥
.wine
```

copies a template file. Incidentally, Wine does not support the original Windows Registry file. Instead, Wine uses a clear-text format, storing the system and user Registry hives in the *system.reg* and *user.reg* files. As programs do not typically access the Registry directly, but take the recommended approach via system calls, this is a good way of making the Registry database easier for humans and machines to read and modify.

Figure 2: Wine is known to work with a variety of Windows applications.

Wine uses links below the *~/.wine/dosdevices* directory to provide the drive letters that Windows expects. The links have exactly the names that Windows programs expect of drive letters, that is, *c:*, *d:* and so on. Thus, the *c* subdirectory maps the Windows *C:* drive within the Linux filesystem tree, and this is where the beginnings of a typical collection of Windows files and subdirectories appears when you launch Wine for the first time. *c/windows* has a few symbolic links to programs such as *regedit.exe* or *notepad.exe*; these programs are part of the Wine package and should work without any additional configuration. To test this, let's call

```
wine notepad
```

This should launch our first program on Wine. Some Wine packages install these programs as hard links in the Linux filesystem tree, allowing users to simply call Notepad to launch the application.

Simple Configuration

The *config* file handles the Wine configuration. The structure of the *config* file is quite simple; just like a Windows *.ini* file, the *config* file comprises group names in square brackets, followed by "*Key*" = "*Value*" pairs. In contrast to Windows, both keys and values need to be quoted, and comments are indicated by semicolons. The group order is not important.

The *[wine]* group (Listing 1) contains a number of basic parameters that govern the filesystem structure and access to the Linux system resources. You might like to set the "*ShowDirSymlinks*" and "*ShowDotFiles*" parameters to 1 here, to allow Windows file displays to show you all the files on your Linux system. For example, hiding symlinks can mean wasting a lot of time searching for files. You can leave all the other parameters as is.

The *[Version]* section in Listing 2 allows users to modify the behavior of the Wine libraries. The "*Windows*" key can assume almost any value you like for Windows versions between Windows 3.0 and the latest Windows 2003. And this is important for a number of applications. However, most applications will run best if you select "*win98*" for this parameter. Also, this Windows 98 version setting is the setting for which the Wine API has the most functionality.

The most powerful switch-box is located below *[DllOverrides]* (Listing 3 line 6). This is where you specify which built-in Wine libraries you will be using and which you will be replacing with the original Windows versions. These settings can be prioritized; for example, specifying "*native, builtin*" indicates that Wine should search for a native library first. If this library is not available, the Wine copy is used. To use the original Windows DLL, you need to store it in the simulated Windows filesystem tree below *windows/system* or in the directory that contains the application you will be launching. However, not all DLLs are really appropriate or useful. For example, the Windows kernel, user system, and graphical subsystem are located in the DLLs *kernel.dll*, *user.dll*, and *gdi.dll*, and can't be used with Wine. The same thing applies to the 32-bit counterparts, *kernel32.dll*, *user32.dll*, and *gdi32.dll*. Low level DLLs, most drivers, and VXD files are similarly useless.

DLLs

Windows 98 DLLs are your best bet; some of them are available off the Internet [4] or from Microsoft. Avoid Windows XP libraries, as they more or less never provide error-free service. Even with Win98 DLLs, caution is recommended. Some work fine with some applications, but the built-in Wine version performs better with others. In many cases, Wine can leverage the so called controls, that is, DLLs that provide

Listing 1: config Part 1

```
01 [wine]
02 "GraphicsDriver" = "x11drv";
   (x11drv, ttydrv)
03 "ShowDotFiles" = "1"
04 "ShowDirSymlinks" = "1"
05 "Path" = "c:\\windows;c:\\
   windows\\system"
06 "Windows" = "c:\\windows"
07 "System" = "c:\\windows\\
   system"
08 "Temp" = "t:\\"
09 "Profile" = "c:\\windows\\
   Profiles\\Administrator"
```

form elements such as input fields or dialog boxes, *commctrl.dll* and *commdlg.dll* and their 32-bit counterparts; the same thing applies to the OLE DLLs that provide COM port functionality without the software installation crashing. As the whole thing is so complicated, Wine allows users to specify the DLL settings on a program-by-program basis, but more of that later. WineHQ has notes on DLL issues and information on many common Windows DLLs at [5]. And you can easily check the status of the Wine library implementation at [6].

DLL Overrides

Some components installed by Office or other programs can be a real pain. These debuggers, which pop up whenever an application crashes, can make life a misery. *mdm.exe* in the windows/system directory is a typical example. Thank goodness you can disable this kind of application in the DLLOverrides section: for example:

```
"*mdm.exe" = "builtin"
```

means search for this component in the Wine library collection. And because this program does not exist in the collection, it can't run.

Graphical Subsystem

The following group *[x11drv]* (Listing 4) describes the properties of Wine's graphical subsystem, which converts Windows API calls to X11 on the Unix operating system. The "*Managed*" and "*Desktop*" settings are of interest. A value of "*Y*" for the former tells the X11 window manager to handle Windows program windows; a setting of "*N*" means that Wine will handle this. The advantage of allowing the window manager to do this job is the improved integration with the desktop windowing system. If you allow Wine to handle window dressing, you get the typical Windows look & feel instead. In many cases, programs will only run in this mode. Setting "Y" as the default is a good idea in most cases.

The "*Desktop*" parameter allows a program to run in its own desktop window; you can specify the size, such as *800x600*. This makes sense if the program breaks your desktop or insists on running in the foreground. This is also the recommended mode for games, which tend to switch graphics modes frequently. And if a game crashes, this can leave the computer in the selected graphics mode. The various members of the Windows Install Shield family also tend to grab the whole desktop and interrupt anything else going on there. Some sections allow individual configurations for the X11 driver, but again, I will be looking at the subject in more depth later on.

The other settings in this section control accelerated graphics modes, so you should be fine with the defaults; the same thing goes for the font, I/O port, and other sections. The installation packages take care of distribution specific settings.

Sound Settings

Things start to liven up again in the "*WinMM*" section, which handles sound output. You need to enable the right driver for sound output. For KDE users, the correect driver is "*winearts.drv*". Not all of these drivers are well implemented. And for sound output in particular, you may need to resort to "*wineoss.drv*", assuming you disable sound output for all your other programs prior to making this selection to avoid conflicts. This warning particularly applies to the KDE and Gnome sound services *artsd* and *esd*.

The subsequent sections handle application defaults. *[AppDefaults\\<program>\\x11drv]* defines the graphic output settings for a specific program. You can use an optional *[AppDefaults\\<program>\\DllOverrides]* to set up DLL defaults that will take priority over the settings in the matching generic sections. This allows you to demote Install Shield to a window, or prevent Quicktime Player 5 from using DirectDraw.

The documentation at [7] gives you

Listing 3: config, Part 2

```
01 [Version]
02 ; Windows version to imitate (
   win95,win98,winme,nt351,nt40,w
   in2k,winxp,win2k3,w
03 "Windows" = "win98"
04 ;"DOS" = "6.22"
05
06 [DllOverrides]
07 ; Some native dlls won't work,
   so leave these builtin.
08 ; Do not modify these lines.
09 "advapi32"     =
   "builtin";Native version won't
   work
10 "avicap32"     =
   "builtin";Hardware related
11 "capi2032"     =
   "builtin";Completely
   implemented
12 "comctl32"     =
   "builtin";Native version cause
   bugs.
13 "comdlg32"     =
   "builtin";thunk
14 "crtdll"       =
   "builtin";Completely
   implemented
15 "ctl3d32"      =
   "builtin";thunk
16 "d3d8"         =
   "builtin";Hardware related
17 "d3d9"         =
   "builtin";Hardware related
18 [...]
19 "msi"          = "native"
20 "ole32"         = "native"
21 "odbc32"        = "native,
   builtin"
22
23 ; some spy or definitely not
   working programs we don't like
   to be started
24 "*autorun.exe" =
   "native,builtin"
25 "*ctfmon.exe" = "builtin"
26 "*ddhelp.exe" = "builtin"
27 "eMusicClient.exe" = "builtin"
28 "*findfast.exe" = "builtin"
29 "icwconn1.exe" = "builtin"
   ;Prevent from loading ICW even
   if registry key
30
31 ; default for all other dlls
   and executables
32 "*" = "native, builtin"
33 ;"*" = "builtin, native"
```

many more useful guidelines on Wine parameters, configuration, and fine tuning.

Experts will prefer the manual approach to Wine configuration; that is, they will prefer editing the configuration file. Newcomers may prefer a convenient, GUI-based tool. *winesetuptk* supports simple editing, or generation of the *.wine/config* files.

Unfortunately, the tool has not been maintained and thus uses some obsolete configuration options. A new tool titled *winecfg* is under development but not finished at this time of writing. And this leaves you with the option of manual configuration, one of the Wine-based Winetools, or Crossover Office.

Ready, steady, go!

One of the most important commands for daily use of Wine is the Windows system reboot. Fortunately, Wine helps you avoid a genuine reboot. Entering

```
wineboot
```

will initiate the boot process in Windows fashion.

Many programs use Install Shield. Unfortunately, Microsoft has not seen fit to specify a clear cut installation scheme thus far, in contrast to the schemes introduced to Linux a while back such as DPKG or RPM. This means that installers are some of the biggest obstacles to Wine, and they can cause issues with more complex programs.

If you have a program that you can't install for love or money, you might like to install the program under Windows 98 first. Make sure you back up, or better still, export the Registry before you attempt the installation. After installing the required program, copy the program directory to the appropriate directory below *~/.wine*, export the Windows Registry again, and compare the results. You will need to run the Wine *regedit* tool to modify any keys that have changed. In most cases, an installation will introduce new DLLs, and again you will need to copy these DLLs to your Wine directory. With a bit of luck, the program should run. Programs that do not use Registry keys are even more useful; Lotus Notes is an example. Although the installer will not run in Wine, you can install on Windows and then simply copy version 6.51 of Notes to the right place in your Wine filesystem tree.

Programs that prompt you for a CD, or to insert the next floppy disk during the installation or later, can prove difficult for inexperienced users. Of course, you need to mount the CD in Linux, however, you must prevent the installer from accessing the CD in the mount directory, as this attempt to access the CD will lock the drive, preventing it from being unmounted when you need to swap disks. If you are aware of this problem, you should have no difficulty responding to a change disk prompt by unmounting the CD at the console, changing the CD, and mounting the next CD.

Daily use has shown that some programs need to be launched from within their program directories to perform well. Some programs do not take kindly to users switching virtual desktops at runtime; in fact, they may hang and take

a lot of reanimation. This often means restarting the application. Programs that do not exit gracefully can leave components running in memory; typing *ps -ax|grep wine* will give you more details.

```
wineserver -kill
```

kills any and all current instances of Wine.

Programs that hog the whole screen and mess it up are also a nuisance. The best approach to handling these programs in Wine is to bundle them off into a window when they are launched, as described previously: simply remove the comment tag from the *Desktop*" = "*1024x768* line to run all of your Windows applications in windows. Or better still, create an application-specific section in the configuration to run just the offending program in a window.

The Control Center helps handle programs that are configured via the Windows System Settings. Typing the following command:

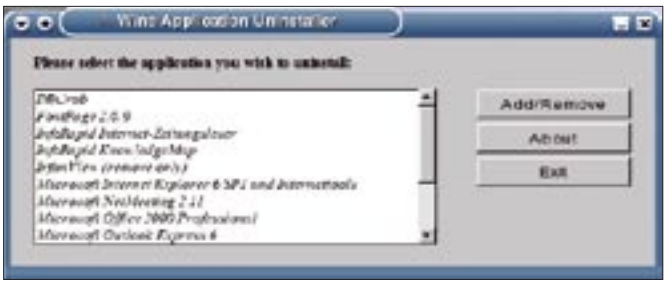

Figure 3: The Uninstaller takes you to a selection of deinstall tools.

```
wine control
```

launches the tool and gives you access to the installed system configuration components. You can also deinstall programs using the Control Center:

```
wine uninstaller
```

launches an applet that takes you to a selection of registered deinstall programs. Wine has restricted support for DOS programs.

```
wine wcmd
```

launches the command interpreter, where you can run DOS programs, but don't set your sights too high. If you want to run a DOS program, you might prefer to use *dosemu* or *dosbox* [8]. This is particularly true of DOS games.

Working with Hardware

Wine provides an emulation layer for binary files belonging to another operating system. This involves a few restrictions that make some applications unusable, or virtually unusable. For example, applications that need a specific hardware driver will not run. In fact, merely attempting to install a Windows hard-

Listing 4: config, Part 3

```
01 [x11drv]
02 ; Number of colors to allocate
   from the system palette
03 "AllocSystemColors" = "100"
04 ; Use a private color map
05 "PrivateColorMap" = "N"
06 ; Favor correctness over speed
   in some graphics operations
07 "PerfectGraphics" = "N"
08 ; Color depth to use on
   multi-depth screens
09 ;;"ScreenDepth" = "16"
10 ; Allow the window manager to
   manage created windows
11 "Managed" = "Y"
12 ; Use a desktop window of
   640x480 for Wine
13 ;"Desktop" = "1024x768"
14 ; Use XFree86 DGA extension if
   present
15 ; (make sure /dev/mem is
   accessible by you !)
16 "UseDGA" = "N"
17 ; Use XVidMode extension if
   present
18 "UseXVidMode" = "Y"
19 ; Use XRandR extension if
   present
20 "UseXRandR" = "N"
21 ...
22
23 [WinMM]
24 "Drivers" = "wineoss.drv"
   ; default for most common
   configurations
25 ;"Drivers" = "winearts.drv"
   ; for KDE
26 ;"Drivers" = "winealsa.drv"
   ; for ALSA users
27 ;"Drivers" = "winejack.drv"
   ; for Jack sound server
28 ;"Drivers" = "winenas.drv"
   ; for NAS sound system
29 ;"Drivers" = "wineaudioio.drv"
   ; for Solaris machines
30 ;"Drivers" = ""              ;
31
32 [AppDefaults\\_INS0432._MP\\
   x11drv]
33 "Desktop" = "1024x768"
34
35 [AppDefaults\\QuickTimePlayer.
   exe\\DLLOverrides]
36 "ddraw" = ""
```

INFO

[1] "Wine Headquarters": *http://www.winehq.org/site/download*

[2] Wine download site: *http://prdownloads.sourceforge.net/wine*

[3] Debian package download: *http://www.winehq.org/site/download-deb*

[4] Source for Windows DLLs: *http://www.dll-files.com*

[5] DLL topics: *http://www.winehq.com/site/docs/wine-devel/arch-dlls*

[6] Library implementation status: *http://www.winehq.com/site/status_dlls*

[7] Wine documentation *http://www.winehq.org/site/documentation*

[8] Dosbox: *http://dosbox.sourceforge.net*

[9] Applications that run on Wine: *http://appdb.winehq.org*

[10] Frank's Corner, alternative Wine site: *http://frankscorner.org*

[11] Debug documentation: *http://winehq.org/site/docs/wine-user/x1824#AEN1826*

[12] Wine forums for your questions: *http://www.winehq.org/site/forums*

[13] Crossover Office: *http://www.codeweavers.com*

[14] Cedega by Transgaming: *http://www.transgaming.com*

[15] Transgaming installation packages: *http://www003.portalis.it/115/*

ware driver is doomed to failure, as the way the two operating systems handle hardware is vastly different. Some programs that required direct hard disk or CD ROM access will run despite this. Wine gives you a direct access mode to support this, allowing such unlikely candidates as the downloadable version of Nero Burning Rom to run, whereas CloneCD provides its own drivers for drive access; drivers that, unfortunately, will not talk to Wine. No matter what application you are thinking of running, you might like to check out the application databases at [9] and [10] to find out if someone has already done the hard work or possibly created an installation report or guide.

DirectX

Games that rely on DirectX may or may not run well. Again, the Internet sources will tell you more. Wine emulates the behavior of DirectX 8, and that gives you a genuine chance of running DirectX compatible games. On the downside, anything that uses copy protection will not run on Wine. These mechanisms are so low-level that they are doomed to fail. Whatever you do, do not install the original DirectX. It won't work, but it might completely destroy your current Wine directory setup.

Debugging

Every Wine user will make the acquaintance of the *winedbg* debugger sooner or later. The debugger appears whenever an application running in Wine crashes. If you are not interested in investigating the whys and wherefores, you can simply quit. If you prefer to analyze the problem, you can type bt to request a backtrace.

The best approach with misbehaving programs is to try out the various settings for the Windows version first. Some programs are very choosy about the path from which you launch them. Again, you might like to try launching the program from its own directory, providing a complete Windows path, or trying the Linux path (Listing 5). Even though the examples in these paths are supposed to be equivalent, a Windows program might not see things the way you or I do.

Your choice of window manager might also cause a few issues. Changing the *Managed* and *Desktop* parameters can work miracles in this case, especially if a program runs but fails to draw the program window correctly.

It can also make sense to specify debugging parameters when launching a program: *WINEDEBUG* = " + *loaddll*" *wine program_path* will launch a program and tell you whether the DLLs the program uses are native or built-in. This information allows you to identify DLLs with an inadequate implementation in Wine and to replace these DLLs with the native Windows equivalents, which in turn, hopefully, will allow you to run the desired program. The Wine Debug documentation at [11] quotes a number of additional parameters that allow users to output specific runtime environment components.

If you're looking for help with Wine, the Wine-Users mailing list archives at [12] have a number of discussion threads with questions and answers on Wine issues. And, of course, new questions are welcome. ■

Future

There are a few things you need to take care of before you can put Wine to productive use. First of all, you should be aware that only 30 percent of all Windows applications will run successfully under Wine. Don't demand too much from the system; it is not designed to be a replacement for Windows; if you are looking for one, the obvious answer is Linux, with its amazing collection of useful and free applications. What Wine gives you is the chance to run important Windows applications for which you do not have a Linux replacement, thus allowing a gradual migration. If you make heavy use of Office products, you will definitely appreciate the ability to continue using these products and trying out something new at the same time. The commercial variants, Cedega for games and CrossOver Office for Office products, are definitely valuable contributions, as they provide support and help keep development work up to speed. In a commercial environment, you will find it difficult to replace them.

Listing 5: Alternative Program Launchers

```
01 wine "c:\\program files\\
   microsoft office\\office\\
   winword.exe"
02
03 wine "~/.wine/c/Program Files/
   Microsoft Office/Office/
   WINWORD.EXE"
04
05 cd ~/.wine/c/Program Files/
   Microsoft Office/Office; wine
   WINWORD.EXE
06
07 cd ~/.wine/c/Program Files/
   Microsoft Office/Office; wine
   "c:\\program files\\microsoft
   office\\office\\winword.exe"
```

Alternatives: CrossOver Office and Cedega

Wine has given rise to a number of major commercial products, all of which have drawn on the Wine codebase before specializing on specific features, which they have then given back. CodeWeavers' CrossOver Office [13] specializes in supporting major Office applications, and it has to be said that CodeWeavers have done a great job. The fact that CodeWeavers give their patches back to the Wine developers has allowed Wine to really grow over the last 12 months. CrossOver Office gives you the latest Wine libraries, a very convenient installer for the applications it supports, complete desktop integration, and support from CodeWeavers. The disadvantages of having to test new Wine versions, time-consuming, marathon installation sessions, and hours spent browsing documentation on the Web, can be a thing of the past, for a low asking price of just US$ 40 or EUR 40.

Transgaming's Cedega [14] specializes in using Wine as a gaming platform and has pushed development of the DirectX emulation. Special copy protection code allows users to run copy protected games. The software is available as a subscription version for US$ 5 or EUR 5 per month with a minimum subscription of three months. With the exception of the copy protection code, the source code for Cedega is available, and this means that users can build Cedega themselves. Ready-made installation packages for various distributions are available from [15].