Running Visual FoxPro on Linux Using Wine

Session Number 37

Paul McNett
881 B Street
Hollister, CA 95023 USA
Voice: 831-636-9900
Fax: 831-636-3077
Email: p@ulmcnett.com

Overview

Visual FoxPro can run on Linux by using an open source project called Wine, which is basically a substitute API. Wine is still in alpha stage, so Visual FoxPro support on Linux is not yet complete. This session offers a practical overview of the current state of affairs of running Visual FoxPro on Wine – what works, what does not, and how to set up a reliable install of Visual FoxPro or a runtime version of an application built with VFP on your Linux workstation. This whitepaper is VFP8-centric, but should apply to other versions as well.

Introduction

Companies all over the world, and of all different sizes and makeups, have internal applications built with Visual FoxPro. Their investments in these applications are safe for the next 5 years at least, provided they stick with Microsoft Windows on the desktop.

Since 1998, when Linux really hit the mainstream in the server arena, it has been steadily gaining new users, mostly developers, until we come to early 2003 when normal users started migrating to the Linux desktop as well. No one really knows how much of Microsoft's commanding market share of the desktop will fall to Linux, but we already know of a few states, countries, and municipalities that have mandated it, and of companies large and small that have committed resources to switching their desktops from Windows to Linux. We also know that the Linux desktop market is only at the start of its growth cycle. As new features get added to the solid Linux platform, it will become an ever more compelling option for companies. As more and more businesses adopt the Linux platform, more and more vendors will start writing native applications for the Linux desktop.

We are not there yet. Software companies are still writing the vast majority of their titles solely for Windows. Even while more people are switching to Linux every day, it is still economically sensible for companies to produce software titles for just the dominant platform. The time will come, however, when these same software manufacturers will be scrambling to offer Linux ports of their popular titles, because it will become sensible from a business standpoint to do so.

Until that time, developers of custom business applications need some sort of bridge, so that they can start to gain experience on the Linux platform while still running – and developing - legacy Windows applications. After all, we've invested a lot in the applications our businesses – and our client's businesses – depend on. If our clients start moving to Linux, we will need a way to continue offering our services.

For many Windows applications – Microsoft Office and Adobe Photoshop, for example – the bridge is already there and getting stronger every day. Wine is the bridge that gives users the ability to install Windows applications on Linux. The application is completely ignorant that it is not in fact running on Windows, but interacting instead with the Wine API.

The “bridge” concept is an important one, when you consider the fact that there are already Linux native applications that perform the same functions as Microsoft Office and Adobe Photoshop. Switching from Microsoft Office on Windows to OpenOffice.org on Linux overnight is a bit drastic, however, as the user will have to immediately learn how to do things the OpenOffice.org way. It is a much better transition in a lot of cases to first switch the underlying operating system while still running the familiar user applications, and perhaps offering OpenOffice.org side-by-side with Microsoft Office. Once the user is comfortable with OpenOffice.org - and once all the company's Word documents, templates, etc. are converted, only at that point should the company uninstall Word completely. One of the biggest hurdles organizations will have in switching from Windows to Linux is with the users – and the applications those users are used to interacting with to get their jobs done. Wine provides the bridge that allows the IT staff to switch the desktop to Linux while the users can transition their use of Windows applications to Linux applications.

The case of legacy custom business applications is touchier than with shrinkwrap applications. Where shrinkwrap applications tend to have open-source replacements available for Linux, as with the case of OpenOffice.org, custom business applications may need to be rewritten with other development tools to run on alternate platforms.

Fortunately, the rewrite of the business application doesn't necessarily need to happen straight away. Just as Wine can provide a bridge for applications such as Word, Excel, and Photoshop, it can work for Visual FoxPro applications as well. You as a VFP developer can run on a Linux workstation while developing Visual FoxPro applications, and/or you can deploy your Visual FoxPro applications to a Linux platform withWine.

This paper is divided into a few sections. First, I'll outline what isn't working in the VFP/Wine setup. The list is longer that I'd like, and may well present a few show stoppers for your own situation, so this section appears first. You'll need to have some idea of where you stand with regard to your own application's potential for successfully running on Linux before you can justify spending time and money on it. Keep in mind that this list used to be longer just a few months ago, when VFP wouldn't even run without crashing, when file and record locking didn't work, and when you couldn't even instantiate a simple ActiveX control.

I'll then provide a step-by-step guide to installing Wine and Visual FoxPro on your Linux workstation, and discuss various strategies for working with this setup. Always keep in mind that what we can do today with Wine is only a fraction of what we'll be able to do tomorrow. In other words, the things that don't work today will eventually be fixed, and by knowing about the Wine option now, you'll be ready to act tomorrow, when your client asks you to put your application on his or her new Linux laptop.

Microsoft Windows will be the dominant desktop operating system for years to come, but unlike just a few years ago, there are real and practical alternatives in Linux and Mac OSX. Currently, these alternative operating systems are indeed on the sidelines, but at the same time they are making inroads. Normal people recognize the 'Linux' name now, and power users are starting to experiment with Linux. Developers of custom business applications need to have strategies in place for deploying to these other operating systems should a customer demand it in the future. Visual FoxPro/Wine can be an option, if even only a temporary one, buying time for the developer to port the application into a more portable development environment, such as Python or Java.

Potential Show stoppers

Wine is alpha software, not advertised to work and certainly not guaranteed to work. It seems like every year, Wine will be at its 1.0 release in just another year or two. It is now over ten years in the making. This is the reality - while I believe 1.0 is going to happen in the 12-18 month timeframe based on the current rate of developer activity, only time will tell if Wine ever gets a final release.

However, Visual FoxPro does work very well on Wine as it stands currently, barring a few key features and a few less important ones. The following table should help you determine if your Visual FoxPro environment is likely to work under Wine. Fortunately, VFP runtime applications are less limited than the IDE as far as things that work are concerned. Therefore, even if you can't (or don't want to) run your development environment on Linux, it is quite possible that your runtime application will still work fine with few if any changes required.

Potential Show Stoppers: VFP IDE

Potential Show Stopper	***Discussion***	***IDE***	***Runtime***
Fonts are not equivalent between Windows and Linux.	Microsoft's core fonts (Arial, Times New Roman, Courier New, Andale Mono, Verdana, Tahoma, and others) do not exist on a plain vanilla Linux distribution, so they are unavailable inside VFP. Font substitution will occur, which may result in controls not being the correct size to display all the text, and/or your controls, forms, and reports not displaying as you designed them. http://corefonts.sourceforge.net contains instructions for legally installing the Microsoft core fonts onto Linux. Once that is done, Wine will discover them and make them available to Visual FoxPro as well.	Yes	Yes

Potential Show Stopper	***Discussion***	***IDE***	***Runtime***
Printing	Reports may layout differently on Wine than on Windows. You'll have to assess this on a report by report basis. Stick to core fonts and generic printer features and it may work with minimal hassles. Developers may need to be prepared to maintain two sets of reports based on the platform being deployed to.	Yes	Yes
No HTML Help	A CHM Help Viewer application does not yet exist for Wine, so you cannot view your VFP Help file. No current workaround known. When Wine is released, it is expected that it will have tools for displaying HTML Help. For runtime applications, you could distribute a WinHelp version, or a pure-HTML version.	Yes	Yes
Version info and icon resources do not get included in EXE in VFP8	VFP8 uses a different system call to set the version info in a newly built exe, and this system call is currently not implemented in Wine, resulting in no version or icon resources being added to your EXE when built from Wine. When testing, this doesn't matter. When time for final build for deployment, build on Windows. A runtime application on Linux built with VFP8 on Windows will correctly read the version and icon information from the executable.	Yes	No
C0000005 error with JPEG images	When adding a JPEG image to a control, a C0000005 error will occur. Convert your image to a BMP instead.	Yes	Yes

Potential Show Stopper	***Discussion***	***IDE***	***Runtime***
XMLToCursor() fails	XMLToCursor() has at least one system dependency. CursorToXML() does work, however. It is possible that with a proper installation of Internet Explorer, this problem would disappear. For now, your application needs to avoid XMLToCursor()	Yes	Yes
DSN-less ODBC connections do not work	Connecting directly to a ODBC driver, as with SQLStringConnect() without specifying a DSN, will cause a stack overflow. Workarounds include constructing a file-based DSN on the fly, for use with your application. (Someone reported to me that dsn-less connections work just fine. I haven't had the chance to verify this yet.)	Yes	Yes
EULA runtime restrictions for VFP7 and VFP8	Basically, if you distribute a VFP7 or VFP8 application to the Linux platform, and you want to be in compliance with the EULA you agreed to without risk that you are misinterpreting anything, you need to purchase a full copy of Visual FoxPro for each Linux workstation you deploy to. My current recommendation would be to deploy your VFP/Linux application with the VFP6 runtimes, which aren't bound by this specious EULA clause. We somehow survived this long without TRY...CATCH after all...	No	Yes

In addition, certain features of the Visual FoxPro IDE are known not to work. These tend to all boil down to those features that have external dependencies (on Internet Explorer being installed, for example.) The following features will not work on the setup I'm outlining in this whitepaper; however, if you need these features it is quite possible that with a proper Internet Explorer installation (I have been unsuccessful in properly installing Internet Explorer on Wine, but there are instructions out there on the web) coupled with a proper Visual FoxPro installation (VFP6, 7, and 8 installs do not work as of this writing) the features will then be available:

VFP8 Task Pane

VFP8 Object Browser (class browser works fine)

VFP8 ToolBox

VFP8 TaskList

VFP8 Code References

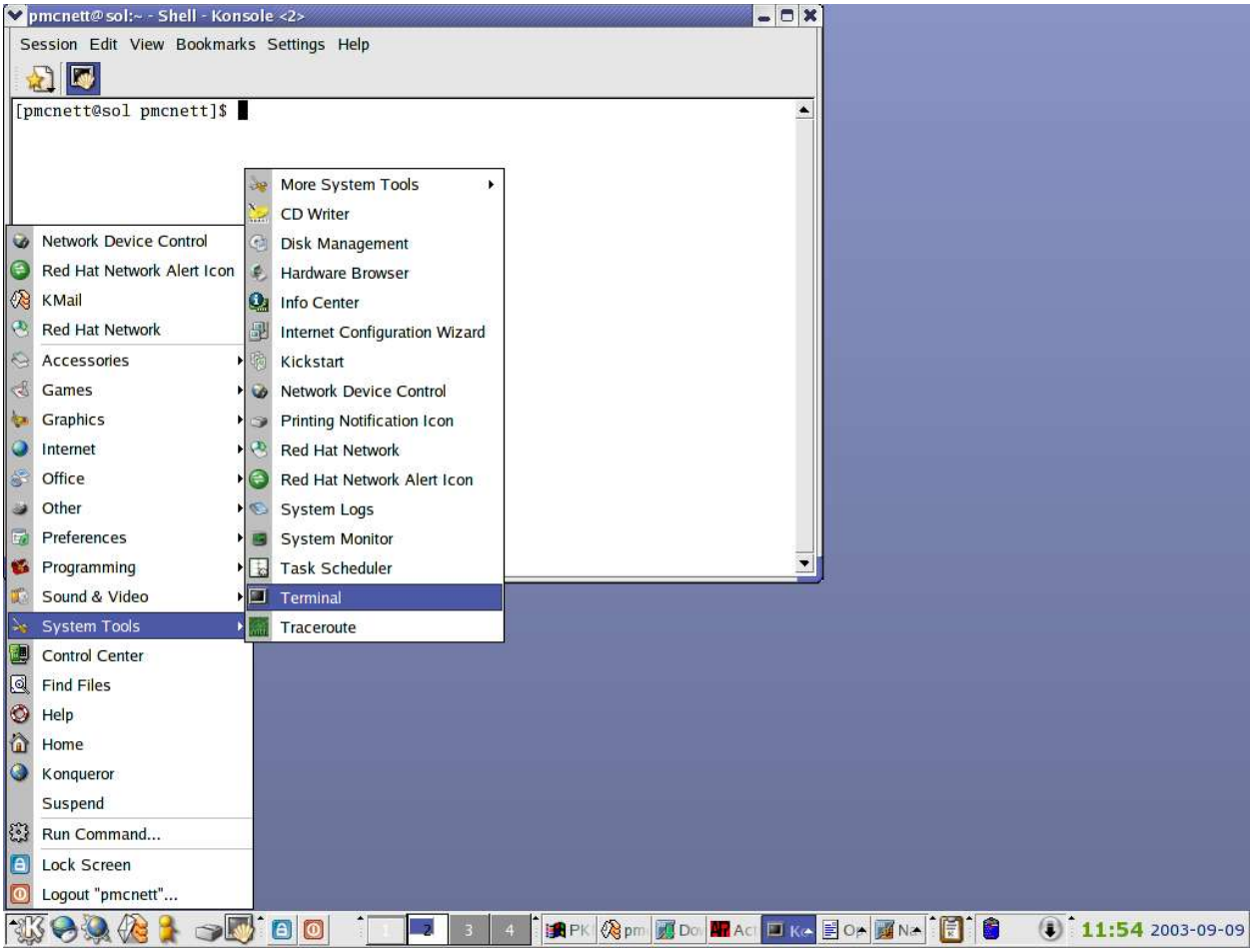
Coverage Profiler

As you can see, most of these non-working features were recently added to Visual FoxPro. Barring minor display anomalies, the rest of the product seems to work just fine. If a developer can live without the above features, as well as without the features listed in “potential show stoppers”, then that developer will likely find VFP on Wine to be a pleasant experience.

Installing Wine and Visual FoxPro on Linux

If you follow these instructions, you should be able to get VFP8 running under Wine on Linux. Along the way, you'll find that this is a pretty good primer on the very basic commands and features of the Linux filesystem and command shell. People used to Windows need such a primer to eventually become productive on the Linux platform.

To get started, fire up Linux and open up a command window. Under RedHat9, this can be done by going to the “start menu” and selecting System Tools/Terminal. The following screen shot shows a terminal window behind the menu that launched it.



Uninstalling any prior version of Wine

It is important that you start out with a clean install of Wine. If you are on a RPM-based system, removing prior installations is easily done by issuing the following commands in your terminal window:

```
su
```

```
<enter root password when prompted>
rpm -q --all | grep -i "wine"
```

For each package reported by the rpm query, enter the following command:

```
rpm -e <package name exactly as it appeared in the query>
```

And finally, end your root session:

```
exit
```

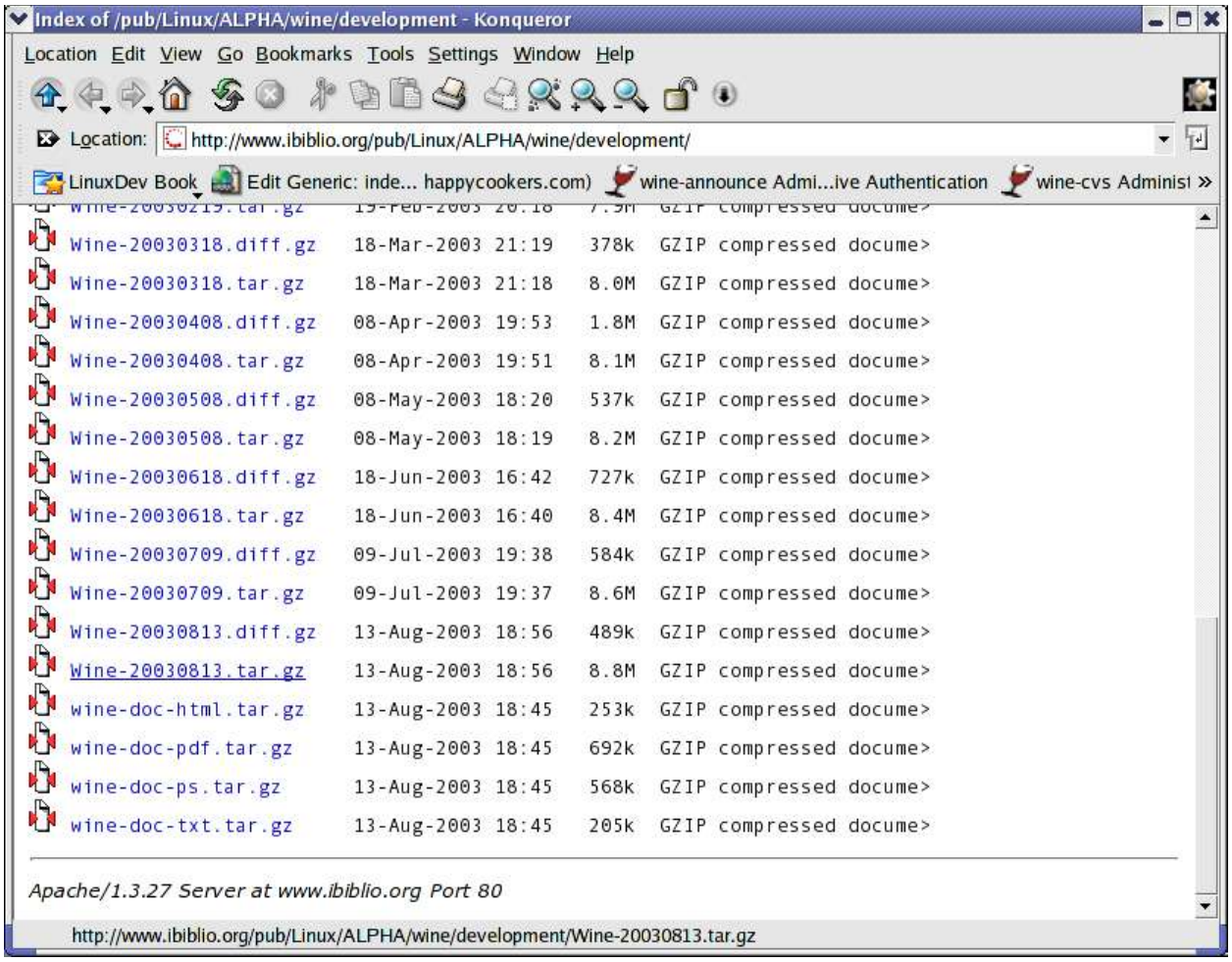
Download, Patch, Build, and Install Wine.

The Wine Headquarters, located at http://www.winehq.org, contains documentation, faqs, news, developer information, and downloads of the Wine source code and binary releases, as well as cvs access to the source tree. To install Visual FoxPro, what you need is the "Wine source from tarball" download - the RPM download will not let you apply a patch that is necessary for the proper functioning of Visual FoxPro.

First, create a directory to hold all your wine source files. Put this in your home directory, as that is the most appropriate place:

```
cd ~   (~ is a shell shortcut that represents your home directory)
mkdir wine
```

Go to http://www.winehq.org/site/download_source and then click on the "Download Source" link close to the top. You'll now be at the ibiblio site, where you can download all different versions of Wine. Scroll to the bottom, and download the latest version of Wine, which conforms to the naming convention of 'Wine-yyyymmdd.tar.gz'. In the screen shot below, I'm downloading 'Wine-20030813.tar.gz', the latest version as of this writing:



Save it to ~/wine/Wine-20030813.tar.gz, and unzip it by typing:

```
cd ~/wine
tar -xzfv Wine-20030813.tar.gz      (to save typing, press <TAB> after 'W')
```

Now you have the Wine source tree inside the directory ~/wine/wine-20030813. You can now erase the zipped file you downloaded, and additionally make a symbolic link to make it easier to access this directory:

```
rm Wine-20030813.tar.gz (remember the <Tab> feature to save typing)
ln -s wine-20030813 wine
ls -al
```

See how the symbolic link 'wine' refers to the Wine-20030813 directory? This is convenient in many ways. Eventually, you'll have at least two versions of the Wine source tree, and you can just redefine your symbolic link to refer to the version you are currently using.

Time to run the build/install script, but before that you need to apply a patch to the source file **wine/dlls/x11drv/winpos.c**. This patch works around a problem with the way WAIT WINDOW's and ToolTipText display in Visual FoxPro. You can certainly skip the patch, but WAIT WINDOW's will not show any text, and ToolTipText - such as used by Intellisense to

show you the arguments for the command or function you are typing in - will cut off all but the first few words. Thus, It is worth installing the patch.

I've named the patch **vfpwinepatchwinsize**, and you can download it from **http://www.paulmcnett.com/vfp/wine**. To apply the patch, just put it in your **/home/pmcnett/wine** directory and issue:

```
cat vfpwinepatchwinsize | patch -p0
```

You'll see some text wiz by and then you are being prompted for the name of the file to patch. Enter:

```
wine/dlls/x11drv/winpos.c
```

You have now downloaded, extracted, and patched the Wine source code. Time to build and install. You'll find that most if not all Linux programs follow the same steps to get them installed on your box, which involves running **configure**, followed by **make**, and finally **make install**.

The information on how to install a given piece of software can usually be found in a file called **INSTALL** but if that doesn't pan out it may instead be in **README**. These files will be found in all open-source projects at the top level of the source tree. There is usually way more information than you need in there, but it usually all boils down to **configure;make;make install**. Anyway, while it is no different with Wine, a tool has been written to handle the configure/make/make install calls for you, and to set up Wine to run on your machine. Issue the following in your command window:

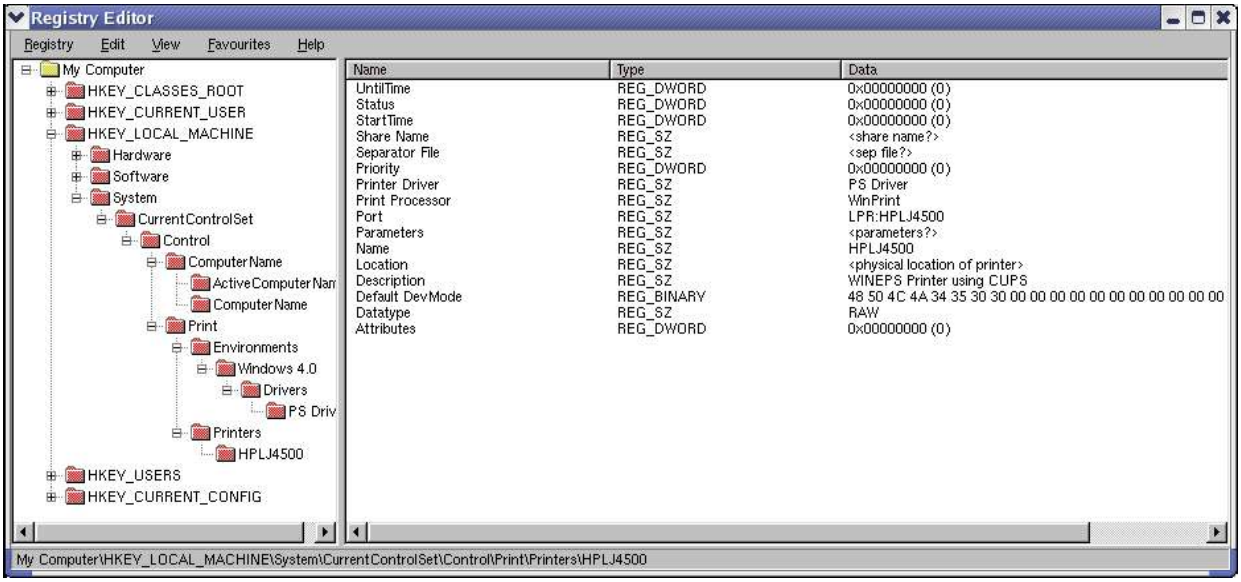
```
cd wine
./tools/wineinstall
```

It will run the configure script, and then prompt you whether you want to build and then install Wine as the root user. Answer 'Yes'. After all the sources build (20 minutes maybe), you'll get prompted for the root password so that the install script can install the Wine binaries into the **/usr/local/bin** and **/usr/local/lib trees**. Enter the root password and then wait another few minutes and you'll get to answer a short series of questions regarding your setup:

```
Create local config file ~/.wine/config?
(yes/no) yes

Searching for an existing Windows installation... not found. (no matching /
etc/fstab mount entry found)

Windows was not found on your system, so I assume you want
a Wine-only installation. Am I correct?
(yes/no) yes

Configuring Wine without Windows.
Some fake Windows directories must be created, to hold any .ini files, DLLs,
start menu entries, and other things your applications may need to install.
Where would you like your fake C drive to be placed?
(default is /home/pmcnett/c) /home/pmcnett/wine/c
Configuring Wine for a no-windows install in /home/pmcnett/wine/c...
```

```
Created /home/pmcnett/.wine/config using default Wine configuration.
You probably want to review the file, though.

Compiling regedit...
```

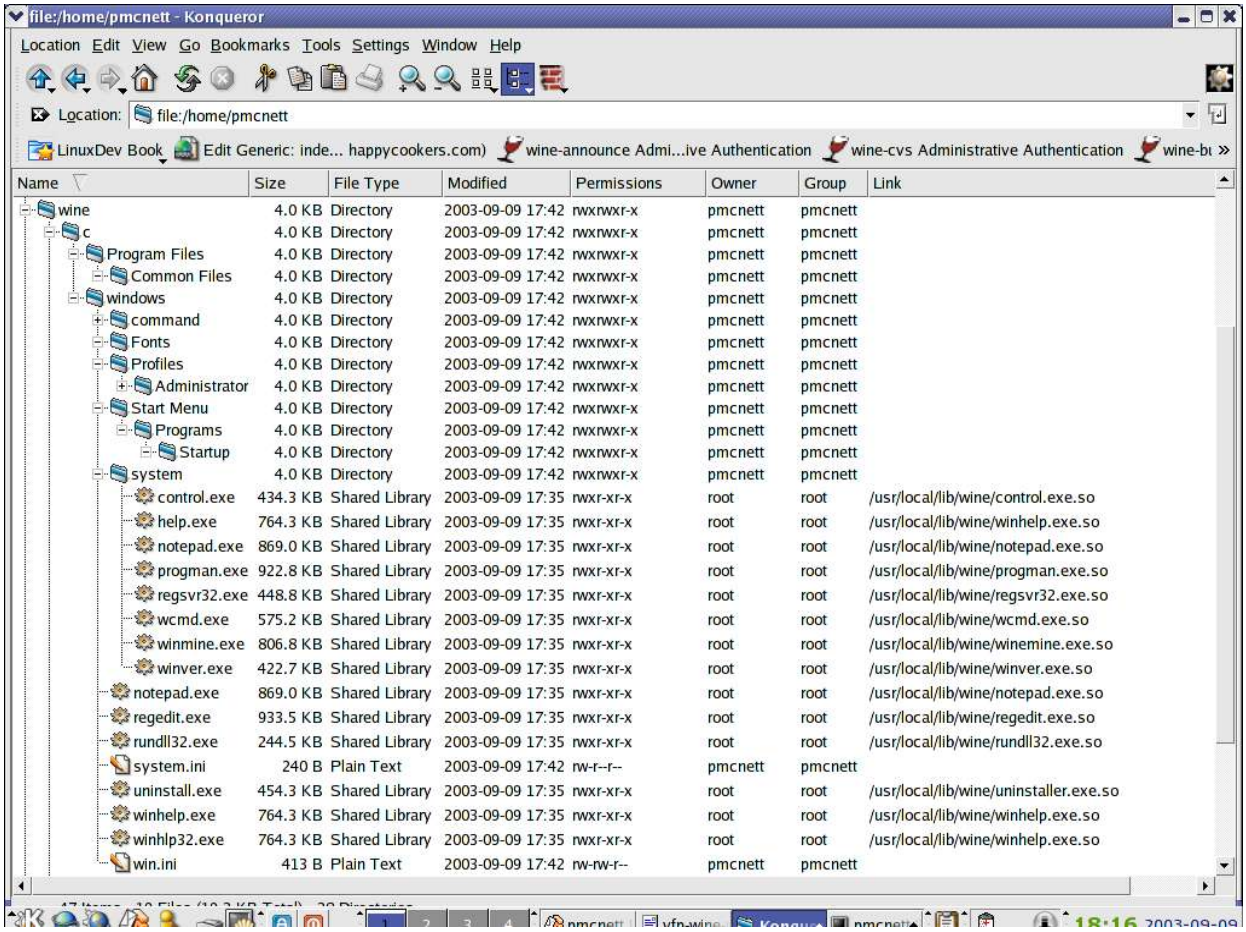
At this point, lots of text scrolls by while fonts are detected and set up for Wine, and eventually some system registry items get installed and Wine's implementation of regedit is displayed. Look how on my system, my network printer was detected and set up in the registry for me:



Exit regedit, and the wineinstall script ends, and you have your initial configuration of Wine. At this point, you have a skeleton fake-windows directory structure, **~/wine/c**, and a configuration directory, **~/.wine/**, which contains your registry (plain text files, in contrast to the binary Windows registry format) as well as the config file, which you will be interacting with quite a bit.

Configure Wine

Have a look at the structure of my fake-c drive after a fresh install of Wine:



It is a skeleton structure that Windows users should recognize, with certain system files and even some common executable files (notice that the executables are actually symbolic links to system files in the **/usr/local/lib/** and **/usr/local/bin/** directories).

As I mentioned before, you also have a **~/.wine/** directory. In Linux, any files or directories with a dot prepended are normally hidden from the user, but if you know they are there you can interact with them. The **-a** switch to the **ls** command will show all files, even the hidden ones. Inside the ~/**.wine/** directory are three registry files and a configuration file. You'll be interacting with the configuration file frequently, so why not make it a bit easier to access by making a symlink to it from your **~/wine/** directory:

```
cd ~/wine/
ln -s ../.wine/config config
```

To continue preparing for the VFP installation, we need some system files from Windows. Eventually, when Wine is released and stable, you will not need these, but for now you do. The files you need to grab from an unused Windows disk are:

oleaut32.dll (only needed for running the VFP IDE - not needed for runtime)

msvcrt.dll (you may not need this as Wine's version is pretty good.)

mscomctl32.ocx (only needed if you want to use the ActiveX common controls)

I was able to download **oleaut32.dll** and **msvcrt.dll** from http://www.dll-files.com, and **mscomctl32.ocx** from http://www.ascentive.com/support/new/support_dll.phtml?dllname=MSCOMCTL32.OCX.

Make a **~/wine/nativeDLLs/** directory, and save these files there. Then, make a symlink into your **wine/c/windows/system/** directory so that Wine can find them when the time comes:

```
mkdir ~/wine/nativeDLLs
cd ~/wine/c/windows/system
ln -s ../../../nativeDLLs/oleaut32.dll oleaut32.dll
ln -s ../../../nativeDLLs/msvcrt.dll msvcrt.dll
ln -s ../../../nativeDLLs/mscomctl32.ocx mscomctl32.ocx
```

This way, Windows programs can find the files, but at the same time it will be easy for you to keep track of them as they will not be buried in your system directory. Alternatively, you could simply copy these files right into your **~/wine/c/windows/system/** directory.

Register **mscomctl32.ocx** using the Wine version of **regsvr32.exe**:

```
cd ~/wine/c/windows/system
wine regsvr32 mscomctl32.ocx
```

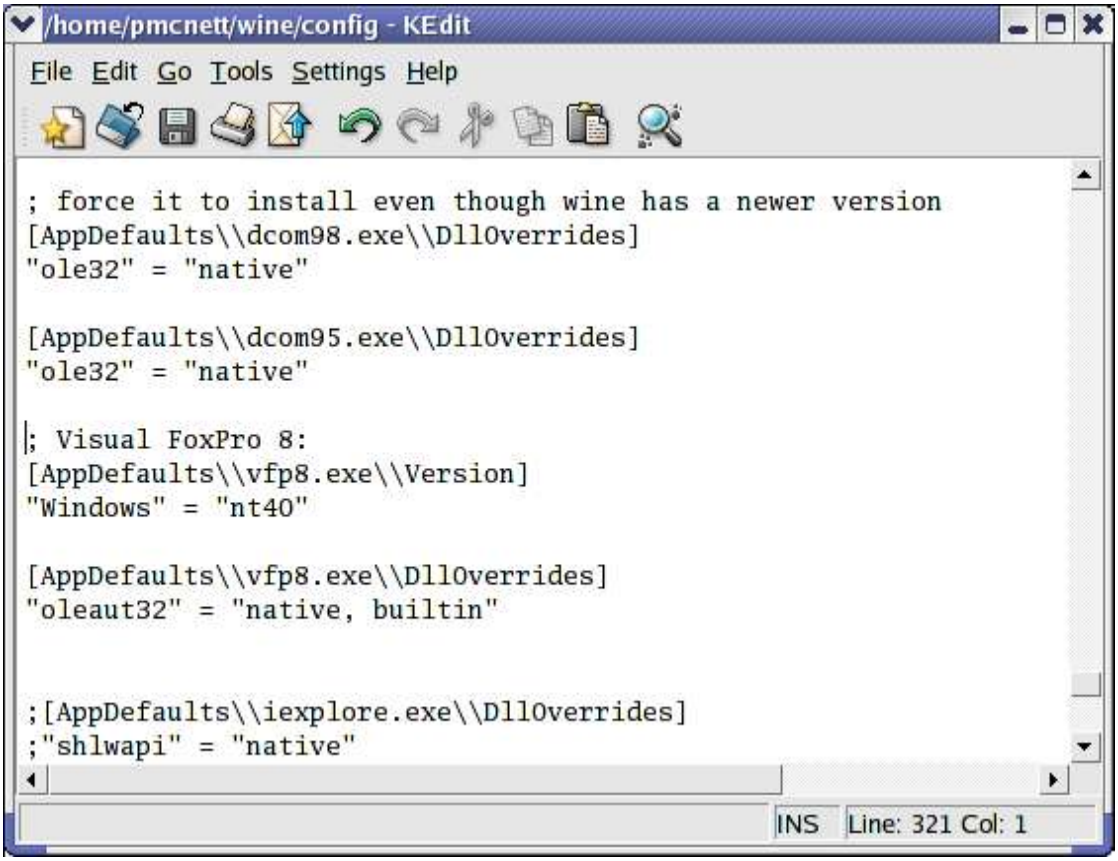
You may see a bunch of text output, followed by a message that the registration was successful. At this point, Visual FoxPro will be able to find ActiveX common controls such as TreeView, ListView, and TabControl, which are used in various parts of the VFP IDE, such as the Class Browser.

Now, make an Application override entry for VFP8 in your wine config file. The Wine config file is the current way to tell Wine how to behave, where to look for things, etc. When Wine is ultimately released, the config file will no longer exist but instead everything will be configured via a Windows-like control panel applet.

What we need to tell Wine boils down to two things. The first is “when running VFP8, you need to pretend that you are Windows NT4”. The second thing we need to tell Wine is “when running VFP8, you need to use the Windows-native version of **oleaut32.dll**, instead of Wine's built-in version”. I used a KDE text editing application, **kedit**, here to put these lines in to the config file, close to the bottom in between a couple other sample application entries:

```
kedit ~/wine/config
```

These settings override what was previously set earlier in the config file. It is recommended that for every application that needs a different configuration, a new AppDefault section for that application is added to the config file, instead of changing the default settings. Look at the screen shot to see the entries you must add in between the already existing entries for DCOM95 and IEXPLORE:



There is one other item you should change in the config file. Close to the top, in the [wine] section, there is a setting "ShowDirSymLinks". Make sure this is uncommented and set to "1", as in:

```
"ShowDirSymLinks" = "1"
```

The default config file has this line commented out, which would make your Visual FoxPro application unable to see your symbolic links. You want to see your symlinks, so therefore please uncomment this line (remove the semicolon).

Install Visual FoxPro

Insert the VFP8 cd into your cdrom drive, and copy the entire cd to **~/wine/c/Program Files/Microsoft Visual FoxPro 8**. To do this from the command line, assuming your cdrom mount is located at **/mnt/cdrom**, follow along:

```
cd /mnt/cdrom
mkdir ~/wine/c/Program\ Files/Microsoft\ Visual\ FoxPro\ 8
cp -R * ~/wine/c/Program\ Files/Microsoft\ Visual\ FoxPro\ 8
```

After a few minutes, the entire directory structure of your VFP8 CD will be copied to your program directory. This is in lieu of installing VFP8, as the installer doesn't currently work in Wine (however, the VFP5 installer recently started working, so take heart in that because one day, all installers will work. Wine is still alpha software.)

The only thing left to do is to put **vfp8enu.dll**, **msvcr70.dll**, and **gdiplus.dll** in the system directory so Wine can find them. Actually, we'll just make symlinks instead of copying them. Start off by making a couple convenience symlinks, one to your Program Files directory, and one to the main VFP8 program directory, as it would have been installed had we run an actual installation:

```
cd ~/wine/c
ln -s Program\ Files prg
cd prg
ln -s Microsoft\ Visual\ FoxPro\ 8/program\ files/microsoft\ visual\ foxpro\
8/ vfp8
```

Remember to use the <TAB> key as you type those long paths. Now, you have a short way to get to the main VFP8 program directory:

```
cd ~/wine/c/prg/vfp8
ls -al
```

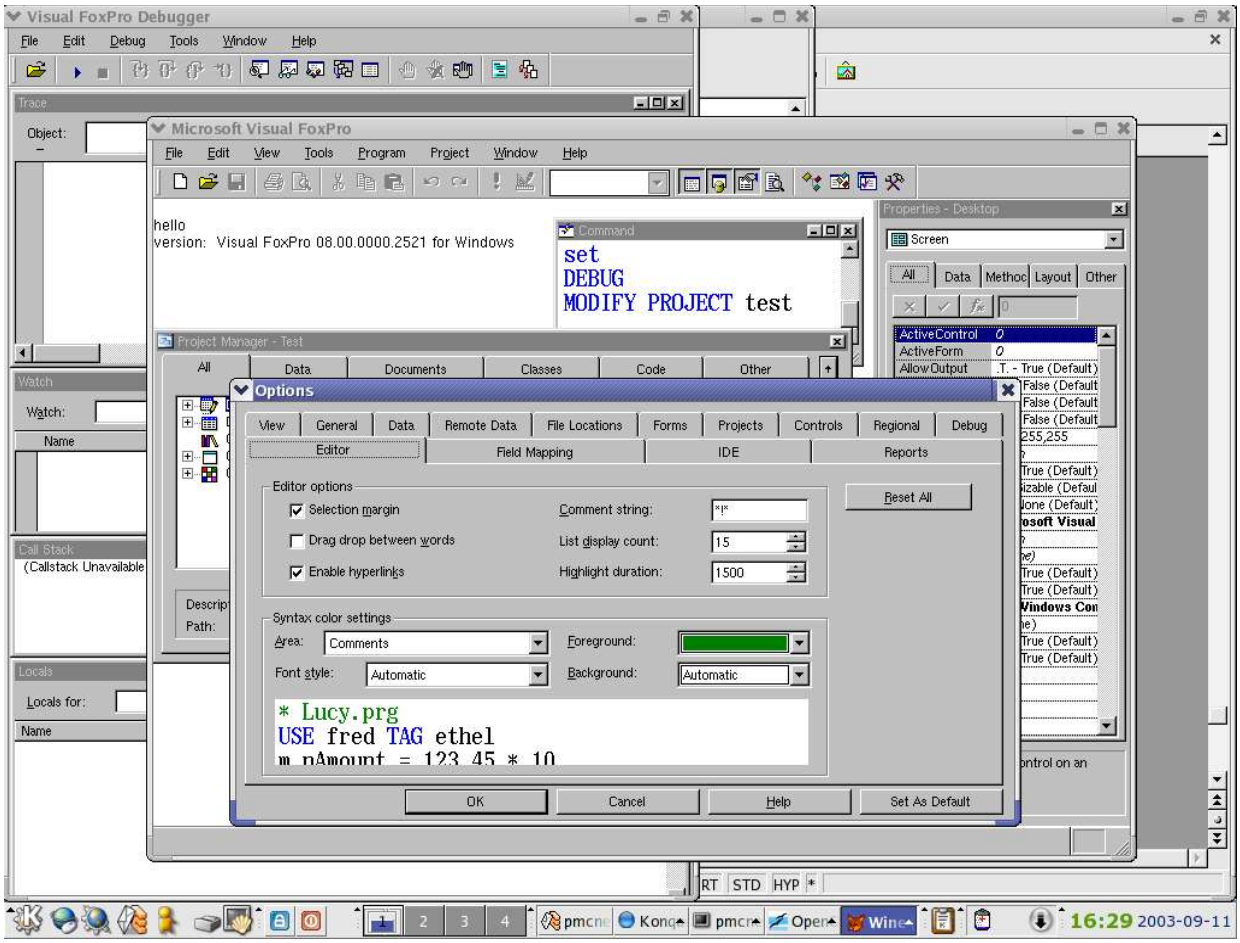
Let us now make those links to the short list of system files provided by VFP8. We could just copy them to the **windows/system** directory, but let us use symbolic linking instead:

```
cd ~/wine/c/windows/system
ln -s ../../prg/Microsoft\ Visual\ FoxPro\ 8/msvcr70.dll msvcr70.dll
ln -s ../../prg/vfp8/vfp8enu.dll vfp8enu.dll
ln -s ../../prg/vfp8/gdiplus.dll gdiplus.dll
ln -s ../../prg/Microsoft\ Visual\ FoxPro\ 8/program\ files/common\
files/microsoft\ shared/vfp/vfp8renu.dll vfp8renu.dll
ln -s ../../prg/Microsoft\ Visual\ FoxPro\ 8/program\ files/common\
files/microsoft\ shared/vfp/vfp8r.dll vfp8r.dll
```

You can now run the VFP8 IDE on Linux by typing:

```
cd ~/wine/c/prg/vfp8
wine vfp8
```

Right-click on the command window, and uncheck “dockable”. Any undocked window marked “dockable” will get in the way of the windows of other Linux applications. Alt+Tab to a different application, and then Alt+Tab back to VFP, and you'll see your cursor. Try various common things, such as creating a project, bringing up the properties window, editing the options, and bringing up the debug window. Here's my screen shot of a newly-installed VFP8 on my RedHat9 Linux laptop running the KDE window manager and Wine-20030813:



Tips and Tricks

There are a few things I've learned, that make things easier working with the VFP IDE from Wine.

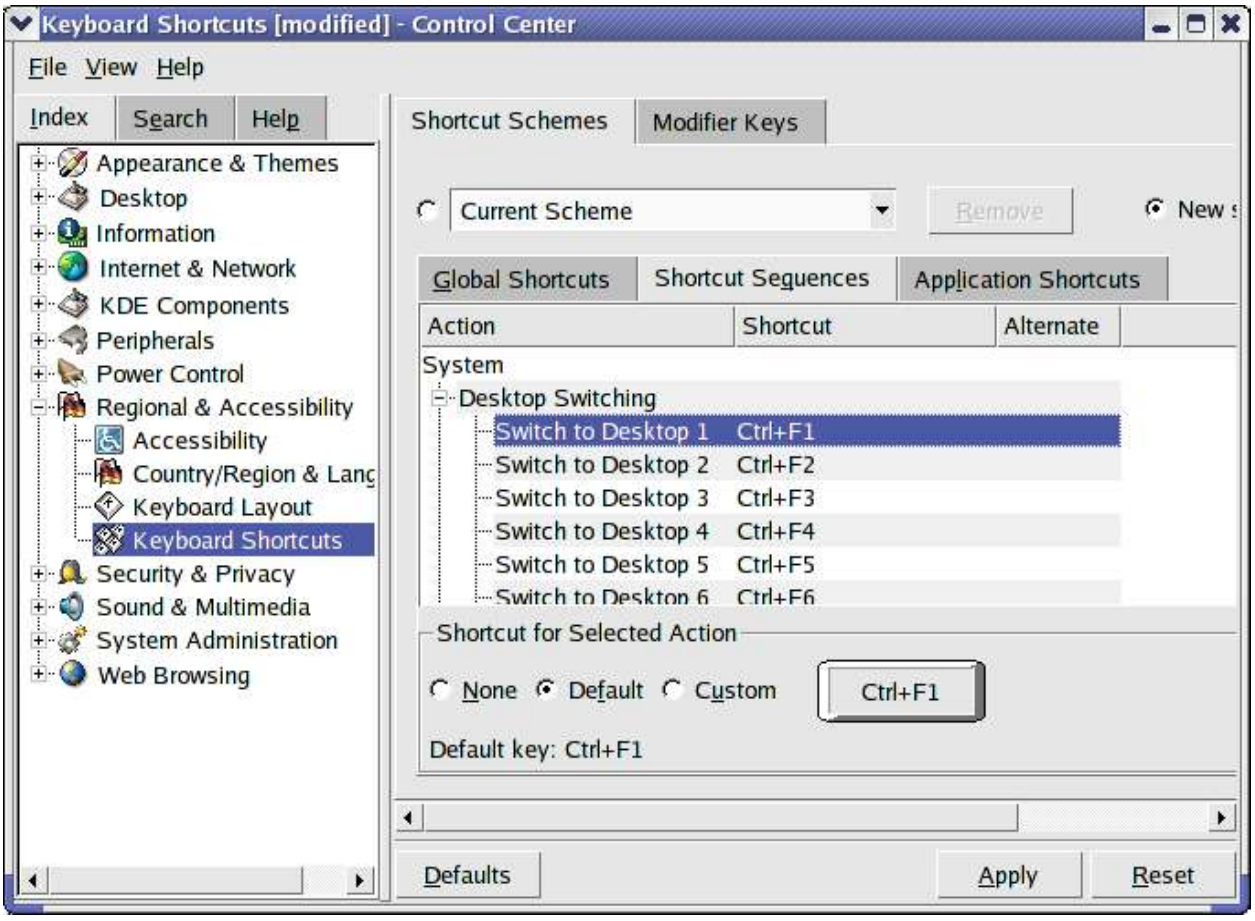
Turning off fatal error reporting

Inside your VFP8 program directory, you'll find a file named **dw15.exe**, which has as its sole mission in life to report fatal errors to Microsoft, which isn't necessarily a bad thing except that this reporting crashes Wine really bad. Turn off this behavior by renaming it to **dw15.old:**

```
cd ~/wine/c/prg/vfp8
mv dw15.exe dw15.old
```

Turning on VFP's use of Ctrl+Function keys

I'm pretty used to pressing Ctrl+F1 to switch windows inside VFP, Ctrl+F2 to get focus to the command window, and Ctrl+F4 to close a window. Most Linux window managers, however, have their own use for these keys, so they don't work in Wine or VFP. Find your Linux window manager's shortcut key control panel, and redefine these keys. For KDE, this is in Control Center/Regional and Accessibility/Keyboard Shortcuts, in the Shortcut Sequences page:



Simply select 'None' for all of these and press 'Apply': Windows users won't likely want to switch virtual desktops anyway, and even if you do you still can, just not with these shortcut keys.

Turning on VFP's knowledge that it is not running on Windows:

You may need, in your multiplatform code, a way to know if you are running on Windows or not. Because Wine fakes Windows, VFP will think it is running on Windows when in fact it is not. This behavior is by design: it means the Wine people have succeeded in their task. Nonetheless, here is a simple way to find out if you are running on Wine.

From your Wine/VFP session's command window, execute the following (actually, try it from VFP on Windows also if you like):

```
? getenv("_")
```

Wine passes along all environmental variables, plus sets some common ones that Windows apps expect to be defined. The variable _ always exists when executing Linux commands from the shell: it simply keeps a copy of the last command executed from that shell. Because Wine makes all environmental variables accessible, and because this variable gets set by our Linux shell, we can evaluate it and save it to a public variable for use within the VFP IDE and in our VFP

runtime applications. Since the last command executed in the current shell was 'wine', we know what environment we are running under. Somewhere in your startup module, place the following:

```
PUBLIC _WINE
_WINE = "wine" $ lower(getenv("_"))
```

Now, you'll have a _WINE public variable defined, that will always be set to .T. when running on Wine, and .F. when running on Windows.

Notes on deploying a runtime application

The detailed instructions on setting up VFP on Wine above can be followed for the most part in getting a runtime exe built with VFP running on a workstation. The main difference is that you do not, obviously, want to copy the VFP development environment over. Also, the native versions of **oleaut32.dll** and **msvcrt.dll** are not needed for the runtime in my experience, and if you don't use the common ActiveX controls you will not even need **mscomctl32.ocx**. In other words, a runtime distribution can be completely free of Windows system files that don't normally get distributed with a VFP-runtime install.

DLL's you need to distribute with your runtime application are **msvcr70.dll**, **vfp8r.dll**, and **vfp8rxxx.dll**, where xxx is the language resource you want, enu for English. As there are not any installers to do this yet, you'll need to make your own by simply copying these files over and putting them either into your application's program directory, or into the windows/system directory on the target system.

The Wine config file on the target computer will need to have an override section for your executable name, not for vfp8.exe as in the instructions above. For example, if you've developed a runtime executable called netgrader.exe, you would need to deploy netgrader.exe, msvcr70.dll, vfp8r.dll, and vfp8renu.dll, and make the following application override section towards the bottom of the wine config file:

```
; NetGrader:
[AppDefaults\\netgrader.exe\\Version]
"Windows" = "nt40"

[AppDefaults\\netgrader.exe\\DllOverrides]
;"odbc32" = "native"
```

I'll discuss ODBC later in this whitepaper. If you use ODBC, it needs to be native. If not, you can omit the odbc32 line, or comment it out as I have done.

If something goes wrong

I've managed to shield you from much of the twiddling that could otherwise be necessary in getting VFP and VFP runtimes working on Linux, and also managed to require a minimum of Windows system files. If you stray from my setup, problems you are likely to run into have to do with the location and accessibility of your windows temp directory, which by default in Wine maps to your Unix temp directory (/tmp), and permissions and accessibility in general. You'll get a “cannot load resources” fatal error if the language-specific resource file can't be found.

Additionally, if VFP has failed a number of times, it will run out of files to use in the temp directory, causing a silent failure the next time you try to start. When this happens, it is safe to issue the following from your Linux command shell:

```
rm -rf /tmp/*
```

This will remove any extraneous files from /tmp, possibly allowing VFP to run normally again.

If you want to fiddle with your installation, it is a good idea to make backups of the ~/.wine/ directory, as well as your /wine/c/ directory tree first, so you can get your working setup back, and/or compare two different setups. Just do something like:

```
cd ~
cp -R .wine .wine-old
cd wine
cp -R c c-old
```

Database Access in VFP on Wine

For the most part, issues with data access are the same on Linux as they are on Windows. Your VFP application will need to connect to one or more databases on a server somewhere. These databases may be native VFP tables, or they may be on a backend database server such as MySQL, Oracle, or Microsoft SQL Server.

Visual FoxPro on Wine can interact with both types of databases, and I'll describe the steps to make it work in this section. There are other methods to connect to backend data as well, such as ADO and OLE-DB, which I do not cover here - these other methods may or may not work on Wine.

Native VFP Tables

Using native VFP tables with Wine is pretty easy to set up. Wine needs to have read/write access to the tables, on a path that exists as one of your fake drive letters. UNC naming does not appear to work currently: you'll need to "map a drive letter" in your Wine config file. Connectivity to a server share needs to be set up before invoking Wine.

In this example, we have a Windows NT Server named MELDER that has a share named DATA which contains the data files for our VFP application. The first step is to mount that share as a local volume on your Linux workstation. There are many ways to do this - I offer an example of one way. Because our data is on a Windows server, we need to use the Samba networking client. Samba is an open-source implementation of the Server Message Block protocol, most widely used by Microsoft Networking. To do this, you first define a mount point somewhere on your local filesystem, and then mount that server share to the mount point. Here, I make a directory to serve as a mount point, then I login as root to mount the share (by default, root is the only one that can issue the mount command):

```
cd ~
mkdir mnt
mkdir mnt/melder-data
su
<enter root password>
smbmount //melder/data /mnt/melder-data -o username=pmcnett,password=secure
exit
```

You will have to work out the best way to get the remote directory mounted locally for your specific circumstances. Whether the remote data is on Windows, Linux, or some other operating system really doesn't matter: there will be a way to mount it locally on Linux, and as long as it is mounted locally, Wine can use it.

Now that you have the local mount to the remote data, you can configure a drive in Wine to point to that mountpoint. This happens in the Wine config file, close to the top with the other drive definitions. Here, I show a configuration you can copy and paste to your Wine config file, in the drive configuration section, which maps drive w: to mnt/melder-data underneath your home directory:

```
[Drive W]
```

```
"Path" = "${HOME}/mnt/melder-data"
"Type" = "network"
"Label" = "Melder Data"
"Filesystem" = "win95"
```

Now, from your VFP app, you can just CD ("w:\") or SET PATH TO w:, or whatever you do normally to in your application logic to point to your data directory.

If you use the Samba client, please make sure you are using the latest version available. There were bugs in earlier releases that kept file and record locking from working correctly. Also, if you are setting up a Samba server to share your VFP data files to Windows and/or Linux clients (beyond the scope of this whitepaper), you'll want to make sure you read up on the proper settings in the Samba config file that have to do with write caching and locking issues.

To share VFP-native tables to only Linux clients, you'll be better off using the Linux-native NFS (Network File System) instead of Samba. If you are sharing to both Linux and Windows, you may still want to share via NFS to the Linux clients, and Samba to the Windows clients, or Samba to both.

Connecting to Remote Data Using ODBC

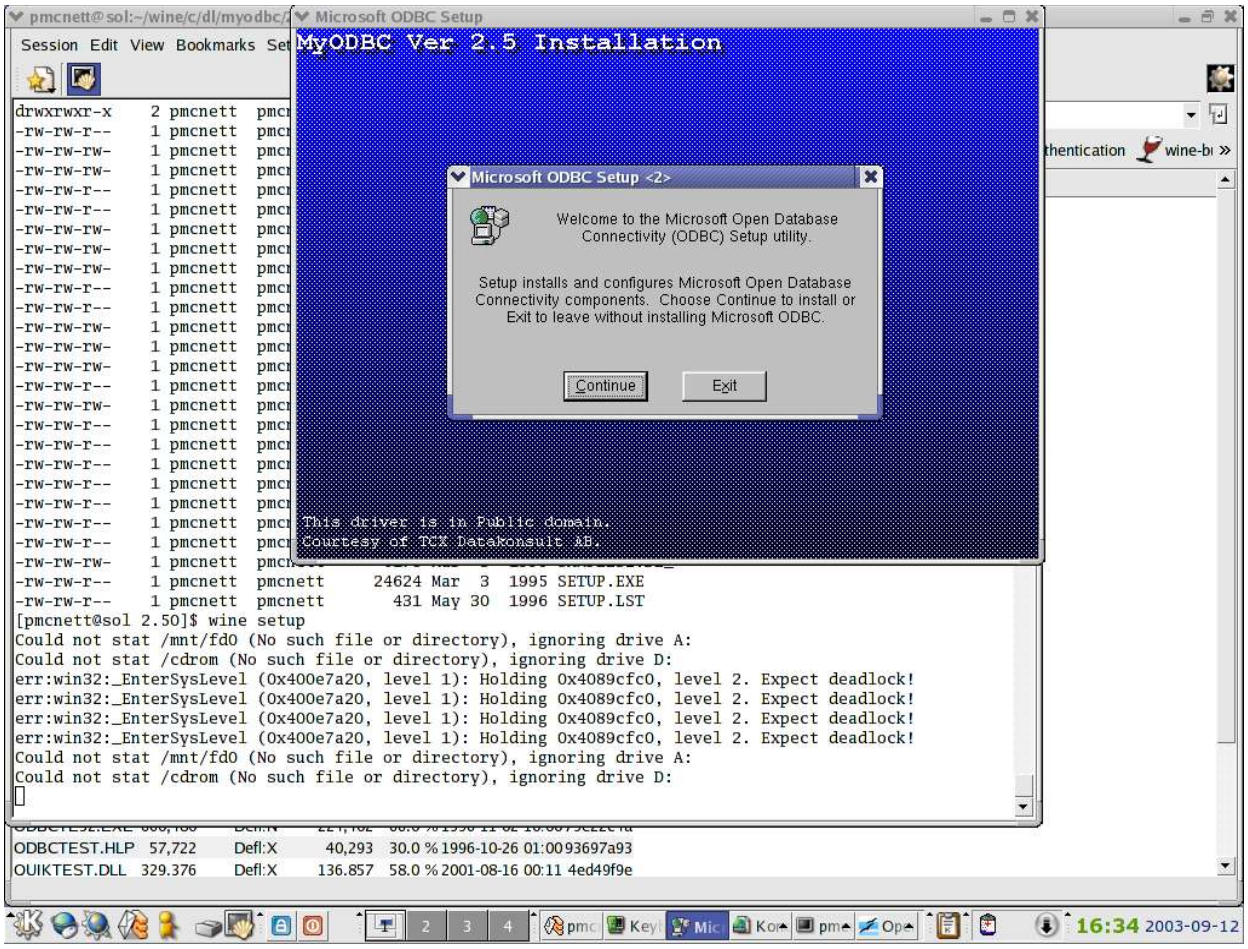
Visual FoxPro really shines in the business world for its ability to bring remote data into local cursors to manipulate using a hybrid xbase/sql approach. You lose nothing doing this on Wine as ODBC works just fine. There are a few lingering issues, however, not least of which is getting ODBC installed on Wine in the first place.

Installing ODBC

Wine comes with a builtin ODBC subsystem that does not work with VFP. That is okay, because it is pretty complex to setup anyway (it involves connecting to unixODBC, an open-source Linux implementation of ODBC). The builtin way will eventually work, but it is easy enough to get the native ODBC working.

First off, you need to get the ODBC subsystem installed. The easiest way I've found to do this is to download and install the Windows version of MyODBC 2.50, the ODBC driver for MySQL, an up and coming open-source database backend. Whether or not you use MySQL as your backend, the MyODBC setup program will install the ODBC subsystem to your Wine setup. Download it from http://www.mysql.com/downloads/api-myodbc-2.50.html, and save it to ~/wine/c/dl/myodbc. You want the one from the Windows Downloads section, the NT/2000/XP Full Setup version. Note that the 3.x version, while better if you are using MySQL, does not include the Microsoft ODBC subsystem. So, no matter what, first install the MyODBC 2.50.x driver, which also installs ODBC, and then if you like also install MyODBC 3.x, which works better for VFP/MySQL connectivity. Install MyODBC by first unzipping it and then invoking it with Wine:

```
cd ~/wine/c/dl/myodbc/2.50/
unzip MyODBC-2.50.zip
wine setup.exe
```



Follow along through the setup wizard, and after clicking 'Finish' you'll have the Microsoft ODBC32 system installed, along with the MyODBC 2.50 driver.

If you have a setup disk for your particular database odbc driver, you can try installing it with Wine by first copying the disk contents to your ~/wine/c/dl/ directory and then invoking it with Wine. If it installs correctly you'll now have that odbc driver to use. For the rest of this section, I'll be discussing MySQL specifically: I have not tried to install any other drivers.

One thing that currently doesn't work from Visual FoxPro is DSN-less odbc connections. In other words, a connect string such as:

```
iHandle = SQLSTRINGCONNECT([DRIVER=MySQL;DATABASE=netgrader;] ;
      + [SERVER=paulmcnett.com;UID=pmcnett;] ;
      + [PWD=secure;PORT=;OPTION=1;STMT=;])
```

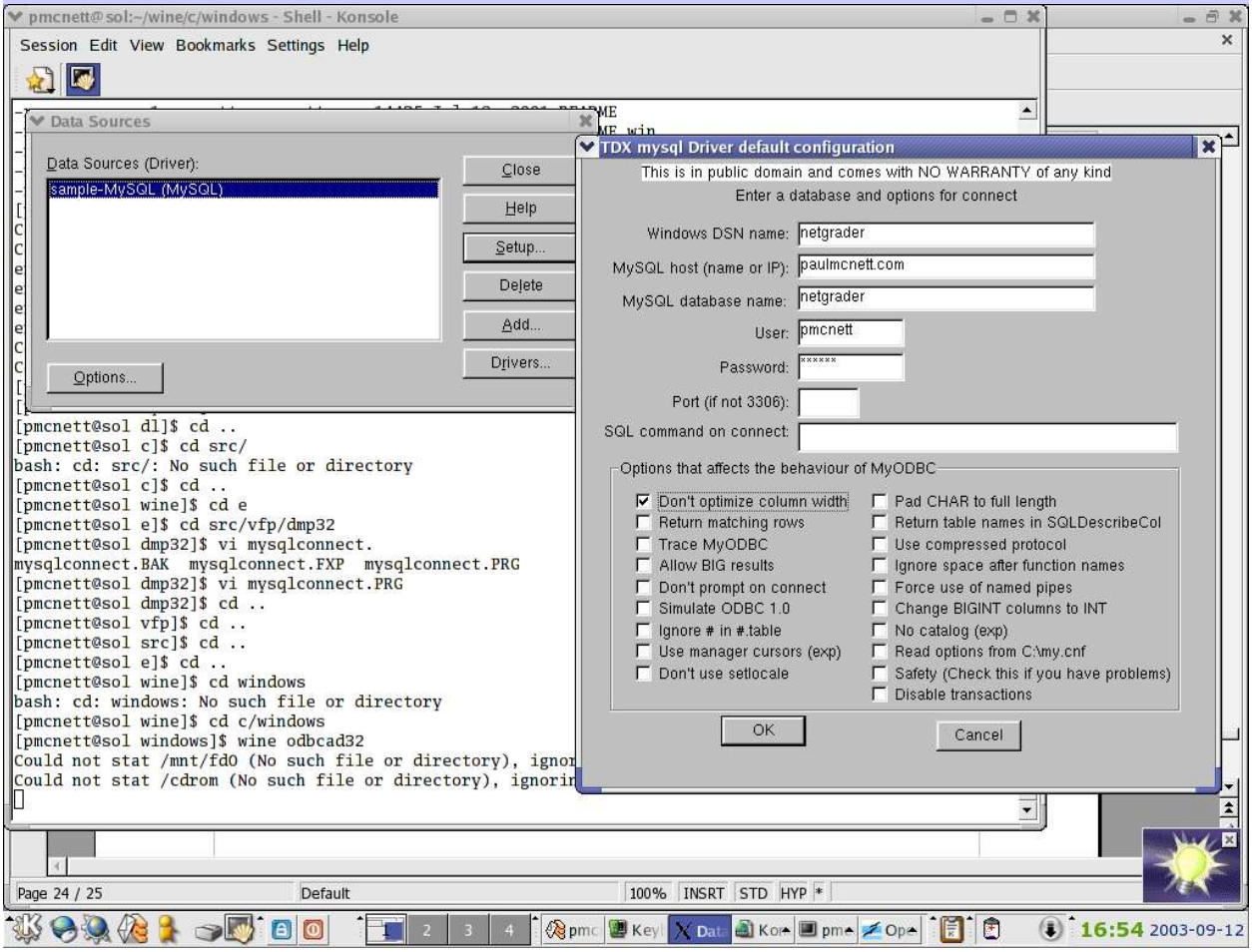
will not work. You need to define and specify a DSN, using either SQLCONNECT() or SQLSTRINGCONNECT(), as in:

```
iHandle = SQLSTRINGCONNECT([DSN=netgrader;])
```

A remote view with a connection using a DSN will also work. Obviously, you'll need to have your DSN defined. The easiest way to do this is with the odbcad32.exe, which got installed with the MyODBC driver, but you can also manually edit the registry if you desire. Another idea is to

have your application dynamically create a file-based DSN, which would be a substitute for not having the dsn-less connect option in Wine. Here, I'm adding a system DSN named 'netgrader':

```
cd ~/wine/c/windows
wine odbcad32.exe
```



The 'Don't optimize column width' is an important setting, as otherwise VFP's non-understanding of MySQL's varchar width will cause fields to be truncated prematurely.

The Wine config file needs to know to use the native version of odbc32.dll instead of the builtin one. Do this in the application override section for VFP8.exe as well as any runtimes that use ODBC. Here's what the entry for VFP8 should look like when using odbc:

```
; Visual FoxPro 8:
[AppDefaults\\vfp8.exe\\Version]
"Windows" = "nt40"

[AppDefaults\\vfp8.exe\\DllOverrides]
"oleaut32" = "native, builtin"
"odbc32" = "native"
```

From this point on, you can use VFP's SQL Passthrough functions or remote views just the same as you do in Windows.

Conclusion

As you see, most of Visual FoxPro's power can be harnessed on Linux today, and that is with Wine in an alpha stage of development. VFP on Wine is every bit as stable as it is on Windows in my experience, however admittedly when it does crash it may crash harder or just be difficult to clean up afterwards.

So from a technical standpoint, businesses can consider VFP on Linux to be an attainable solution for at least some applications. If a business wants to move some or all workstations from Windows to Linux, the freedom to also move the custom applications developed in VFP makes the entire job easier. Perhaps those applications will be rewritten in a more portable language, such as Java or C.

One last thing to touch upon, however, is the non-technical aspect of the EULA for Visual FoxPro 7 and 8, which specifically states that a runtime application must run in conjunction with a Microsoft Windows platform. My personal feeling is that this a completely unenforceable if not illegal clause in the license agreement. However, personal feelings and business decisions are not good bedfellows, and the last thing a sane business wants to invite is a lawsuit from the richest company in the world.

The EULA issue will eventually be put to rest, perhaps by a direct legal challenge, or perhaps by someone - a charity, for example - making the VFP8 runtimes available on the net for free download, along with instructions on how to install the application on Linux as well as Windows. We can all safely watch from a safe distance to see what the legal outcome of such a move may be. By that time, VFP8 on Wine will be more stable and high-performance than ever.

For now, however, while these issues are still open and unresolved, a business may want to consider deploying runtimes built with VFP6 or below, instead of with VFP7 or VFP8 which contain the unclear EULA wording. This is, in fact, exactly what I recommend to my clients when they want to switch to Linux.

Linux is only going to gain market share over the next few years. The ability to deploy your Visual FoxPro application to the Linux platform has a lot of value, however intangible it may seem today.