Dial/Panel Weirdness for Windows Users

1. Introduction

WB8RCR has currently available (as of April 27, 2003) three programs related to creating panels for homebrew equipment.

First is the Windows Dial program. This is a fairly typical Windows program, although it is quite crude. It doesn't allow you to save a dial design, but dials are really pretty simple things. This isn't a serious problem since you merely enter a few values and print your dial.

Secondly there is a command line version of Dial. This program is somewhat clumsier to use, but you can (in fact you must) save your design and it has the added advantage of being useable on a large number of platforms. WB8RCR provides both a Windows and Linux compile of this program, but the program can be recompiled for virtually any platform.[1]

Third, there is the Panel program. This program allows you to assemble multiple dials and other controls and annotation into a design for a complete panel. This is also a command line program and works a lot like the command line version of Dial. Like the command line Dial it can be recompiled for many platforms.

These "command line" programs behave differently than the typical Windows program, and thus may be somewhat confusing to the more novice PC user. Worse, these programs in particular have some other requirements that aren't terribly obvious.

2. What is a command line program

Windows programs rely on a lot of Windows specific code to interface to the user. In fact, in most amateur radio related Windows programs, well over 90% of the code written by the programmer is involved with the user interface, and the programmer only writes a tiny fraction of the code required. Thus, Windows programs tend to be large and complex, and, perhaps more to the point, very highly dependent on the specifics of the Windows operating system.

In contrast, command line programs spend relatively little effort on the user interface. Instead, they focus on the actual work to be done. It is relatively simple to write a command line program which only uses industry standard operating system capabilities, which means that the program can be used by a number of different operating systems.

2.1. Text Interfaces

Historically, people interfaced to computers through text based terminals, much like packet radio or RTTY. Standardized user interfaces, then, relied on prompting the user to type something in. Rather than clicking on an icon, these programs expected the user to start a program by typing in a command, typically the name of the program. The user might also be expected to add some parameters to the command line to tell the program to do something special.

As an example, try opening up an "MS-DOS Prompt" from Windows. You will get a black

[1] Any platform for which a standards compliant C compiler is available, which in practice means any relevant platform. Platforms which became obsolete before about 1980 tend to not have C compilers, and processors that are intended to be embedded in calculators and watches don't have the necessary capability, but almost anything that you would recognize as a standalone computer and might be tempted to attach a printer to would qualify as "virtually any platform".

window, emulating an old terminal, with a friendly prompt, something like:

```
C:WINDOWS>
```

The operating system is waiting for you to give it a command. Typing

```
DIR
```

will cause a long list to scroll by, listing all the files in the C:\WINDOWS folder. What you told Windows to do is to go out and find the DIR program and run it. The DIR program sends it's output to the "terminal", in this case, the MS-DOS Prompt Window. The only output that is valid to the terminal is text, so the output is lacking the icons and other graphical niceties you see on the Windows Explorer window, but the underlying data is the same.

2.2. Parameters

Now, if instead you typed:

```
DIR C:\
```

you will see a slightly shorter list. The C:\ is a *parameter* which tells Windows that you want the listing of the top level folder on the C drive, rather than the listing of the current directory.

2.3. Switches

You could also try typing something like:

```
DIR /?
```

Here, the /? is a *switch* which tells DIR that you want to see a short summary of the available options, rather than a directory listing. Most of the command line programs in Windows allow you to enter a /? or /H to tell the program you need help in working it. DOS, Windows, and many older operating systems use the slash to introduce a switch. Unix based operating systems like Linux use a minus sign to introduce a switch. Some DOS/Windows command line programs are descendants of Unix programs, and thus also use the minus.

2.4. Redirection

There is another set of special symbols that can be used on the command line. Normally, a command line program expects to send it's output to the "terminal" screen and receive it's input from the keyboard. However, sometimes it's nice to have a file or another program provide the input or receive the output. There are 3 special symbols provided for doing this.

This first is the pipe or | symbol. This symbol means to take the output of the program in front of the pipe symbol and send it to the program appearing to the right of the symbol. As a silly example, go to your MS-DOS Prompt window and type:

```
DIR /?|SORT
```

The result will be the same help screen from DIR you saw before, sorted alphabetically by line.

Not a very useful example, but you can see that it could be handy in some cases.

The next thing is the greater than symbol, or the output redirection symbol. If you were to type:

```
DIR > LIST.TXT
```

the output of the directory command will be redirected to a file called LIST.TXT which you could then open with Notepad or whatever. Sometimes it can be pretty handy to save the results of some program.

Finally the input redirection command, or less than symbol. This is a little less useful than the output redirection, but it still can be handy. If you ran the previous command, try

```
SORT < LIST.TXT
```

SORT expects it's input from the keyboard. This command replaces the keboard with the LIST.TXT file, sorts it, and sends the output to the screen. You can combine these two by something like:

```
SORT < LIST.TXT > SORTED.TXT
```

which will sort LIST.TXT and place the result in SORTED.TXT.

3. Printers

The other big difference between Windows and standards based operating systems is the way printers are handled. Back when the world was young, it was simple. You just sent letters to the printer a lot like you would to a terminal. As long as you didn't need graphics, it was pretty simple business.

With the advent of Lotus 1-2-3 the world changed. People began to expect to get graphs on their printouts, and graphs are pretty tough business. Printer manufacturers each came out with their own scheme for getting graphics to printers. This meant that each program needed to have special printer *drivers* to deal with each printer. Often, you had to choose programs based on what printer the program supported.

Microsoft and the rest of the industry took different approaches to solving this problem. Microsoft Windows became the first operating system to provide a way for printer drivers to be included in the operating system. It also became the first operating system with a large enough market share to force practically all printer suppliers to write drivers for Windows.

The rest of the industry took the approach that a standard method was needed to interface to printers, and came up with the PostScript printer language. This language was quite different than other printer control languages. Instead of describing the dots that make up the page as most printers at the time did for graphics, PostScript described the page and the objects on it.

There were 2 effects of this decision. First, since the amount of data that had to be sent to the printer was much smaller, PostScript printers were much faster than the dot matrix printers that were popular at the time. Offsetting this, though, the printer needed a lot of processing power, which made the printers a lot more expensive. As a result, the PostScript printers tended to be used for business printers, and non-PostScript printers became more popular for home use.

As the cost of processing has decreased, PostScript printers have become less expensive, and non

PostScript printers have included more processing power, thus reducing the performance gap. However, PostScript retains one huge advantage - PostScript printers can be used on virtually any system, while non-PostScript printers are largely relegated to Windows only.

4. Dial and Panel

The command line versions of Dial and Panel expect you to produce a file which describes what you want. This can be done in any "plain text" editor. In Windows, Notepad is the most common text editor. You can use Wordpad or Word, but you need to be careful to save the result as plain text using these editors since they are capable of saving forms other than plain text.

Dial and Panel then read this script file and produce an output file which contains PostScript. This PostScript file can then be copied to a PostScript printer.

The command line versions of both Dial and Panel require a single parameter, which is the file name, and send their output, in PostScript, to the console. You might invoke Panel with a command like:

```
Panel MyPanel.txt >MyPanel.ps
```

This would read the text file you prepared describing your panel named "MyPanel.txt" and produce a PostScript file called "MyPanel.ps".

If you had a PostScript printer attached to LPT2: and wanted to go directly to it, you could type something like:

```
Panel MyPanel.txt >LPT2:
```

On Linux systems, there is a program called ps2pdf which accepts PostScript input and produces Adobe Acrobat format as output. A Linux user wanting a PDF file instead of PostScript could do something like:

```
Panel MyPanel.txt | ps2pdf >MyPanel.pdf
```

The command line version of Dial works in exactly the same way, only the input text file is a little simpler.

This still leaves you with a problem if you are a typical Windows user who doesn't have a PostScript printer.

5. GhostScript

There is a lot of material out on the Internet in PostScript format. PostScript is also a very convenient format for programmers to use to generate graphics. As an industry standard, there really is a need to be able to display PostScript on the screen or print it on nonstandard printers.

GhostScript is a freeware program to answer that problem. GhostScript will reformat PostScript for a wide variety of outputs. GhostScript includes drivers for many popular printers, but it can also print to whatever the Windows printer is. It can also produce various forms of graphics files from PostScript.

GhostScript is installed as a standard program on most Linux distributions, but it isn't typically found on Windows systems. If you want to use GhostScript on Windows you will need to download and

install it. It's a fairly sizeable download (over 5Mb) so you will want a fast connection for the download if possible.

Unfortunately, GhostScript is a relatively challenging program to install and set up. It is available on a great many platforms, which makes it very useful, but it requires industry standard fonts, and of course, Windows uses it's own proprietary font format, so the GhostScript installation includes a large number of fonts. There are weird font mapping files which typically need some work to get the maximum quality out of the GhostScript output.

6. OK so now what

I'm afraid at this point I have little to offer. If you are a Windows user and do a fair bit of homebrewing, it may be worthwhile to download and install GhostScript. It's a very handy program, and it works well. But if you are relatively new to Windows, the whole GhostScript thing is likely to be an exercise in frustration. If you have a friend who is good at this stuff, perhaps he can help. GhostScript is pretty neat once it's installed, but the install isn't really all that easy.

I do hope to do a full Windows implementation of Panel some time in the future, but right now that looks to be a ways out. Personally, I prefer the script input sort of model to the Windows click and drag thing, and it's a LOT of work to do these Windows programs. I wouldn't go looking for something before the fall of 2003 and even that may be very optimistic.

-- de WB8RCR