Usage notes for CLIPMETA

Hippocrates Sendoukas

September 30, 1993

Introduction

clipmeta.exe is a small utility that takes a metafile or a bitmap from the system clipboard and saves it to a disk metafile. This is helpful because most programs can send a metafile or bitmap to the clipboard (using the "Cut" or "Copy" commands from the "Edit" submenu), but do not produce a disk metafile. The main motivation for writing this program was to save pictures produced by any Windows program, so they can be used with my DVI driver. I tested it with several popular programs (Excel, PowerPoint, Word for Windows, 1-2-3 for Windows, Toolbook and Paintbrush) and it works fine.

Operation

You can invoke the program as:

clipmeta [-q] [-p] [filename]

where the items is square brackets are optional. If you do not supply any parameters, the program displays some information about the "natural" size of the data in the clipboard, and then displays a "Save As" dialog box that lets you select the directory and filename to save the metafile. If you specify the "-q" (quiet) switch, the program will not display the initial information about the metafile. If you specify the "-p" (plain) switch, the produced metafile will be in plain format instead of placeable format. If you supply a filename, the program will skip the "Save As" dialog box and will use the specified filename as the destination for the metafile.

The optional command line parameters are not anachronisms even for graphical user interfaces: suppose for example that you have a chart in Microsoft Excel and you want to save it to a metafile. Suppose that the name of the chart is "graph.xlc". The standard steps to produce the metafile are:

1. Select the entire chart by clicking the mouse.
2. Select the "Copy" command from the "Edit" submenu.
3. Minimize Excel so you can see the Program Manager.
4. Run the "clipmeta" utility.
5. Press the OK button when you see the size of the metafile.
6. Select a directory and enter the name "graph.wmf" in the "Save As" dialog box.
7. Restore Excel to continue.

These steps are not that difficult, but you do not want to do them frequently. After all, the whole point of a graphical user interface is to make your life easier. For this reason, you can write an Excel macro that carries all these steps automatically. I use the following macro:

```
A1      Export_Chart (a)
A2      =get.window(1)
A3      =if(find(".XLC",A2)>0,goto(A5))
A4      =goto(A9)
A5      =window.restore()
A6      =select("Chart")
A7      =copy()
A8      =exec("clipmeta -p -q"&substitute(A2,".XLC",".WMF"),1)
A9      =return()
```

The first line (A1) is the name of the macro and specifies that it can be invoked with the Control-A key. The second line (A2) gets the filename of the current window (it should be "GRAPH.XLC"). The next two lines (A3 and A4) make sure that the name has the extension "XLC". This check is useful, if you try to run the macro either from a non-chart window, or if the chart has not been already named. The next line (A5) ensures that the chart window is not maximized or minimized (you will get better results if it is at its natural size). The next two lines (A6 and A7) select the chart and copy it to the clipboard. The following line (A8) substitutes the extension "WMF" instead of "XLC" (the filename will become "GRAPH.WMF") and executes the command "clipmeta -p -q GRAPH.WMF". The final line (A9) terminates the macro.

If you run the macro from the chart window, all the above steps will be done automatically. You can put the macro in the global macro sheet of Excel, so it is always available. Furthermore, you can put a new button in the graph toolbar, and assign the macro to that button; in this way, you can do the whole operation just by clicking on that button.

You can tailor the macro to your own preferences, and you can do similar things on any application that supports macros. My main objective was to maximize the user's convenience. You can also put the program in any of the Program Manager groups for quick access when you cannot automate the entire process.

The program can export three different types of data from the clipboard: metafiles (reported as "Picture" by the Display submenu of clipboard.exe), device independent bitmaps (DIB) (reported as "DIB bitmap" by clipboard.exe) and device dependent bitmaps (DDB) (reported as "Bitmap" by clipboard.exe). If there are multiple data in the clipboard, clipmeta will try first to find a DIB. If no DIB exists in the clipboard, then it will try to find a metafile; if it doesn't find it either, it will try to find a DDB. If it finds any bitmaps in the clipboard (DIB or DDB), it will also look for a palette which usually accompanies such bitmaps. Note that the DDB format was not really meant to be transferred between programs or devices, and it will be better if you can avoid it; I support it only as a method of last resort.

Bitmap considerations

A metafile can contain any Windows primitives including bitmaps; several programs (eg., Paintbrush) follow this approach for convenience purposes. There are however several problems with this method:

1. Bitmaps cannot be scaled easily, so such metafiles are not scalable for any practical purposes. This is not a limitation of the metafile format or the operating system: it is just a fact of life. If you truly need to scale a bitmap, either do the scaling with a specialized program before you put it in a metafile, or use a scalable format.

2. Large bitmaps consume lots of disk space and memory. Many video drivers have serious bugs when handling objects larger than 64K bytes, so you may encounter GP faults in any program which tries to read a large bitmap. The only solutions for this problem are to reduce the size of the bitmap, or use a better written video driver.

3. As mentioned above, bitmaps come in two varieties: Device Dependent Bitmaps (DDB) and Device Independent Bitmaps (DIB). The primary purpose of DDBs is for use within a single program, because they cannot be displayed properly on different video modes; as a matter of fact, they often are not displayed properly even in the same video mode if the color palette can change. Another problem specific to DDBs is that their size depends on the video mode

instead of the actual bitmap data. Consider for example a black and white 1024x1024 bitmap; its data actually occupy 1M bits, which is equal to 128K bytes. If you export this bitmap in DIB format, the file size will be approximately equal to 128K. If on the other hand you export this bitmap in DDB format and you use a 256-color mode, it will take 1M bytes. Similarly, the same DDB will take 3M bytes on a TrueColor display. It should be obvious by now that it's better to avoid DDBs whenever possible (ie., try to use programs which export DIBs instead of DDBs).

Clipmeta tries to alleviate some of the bitmaps problems: whenever it sees a DDB or DIB (but no metafile) in the clipboard, it tries to create an reasonably efficient metafile by splitting the bitmap in small pieces; experiments show that this approach circumvents the 64K limitation of the buggy video drivers. There is no time or space penalty for this trick. This however cannot work for metafiles (which may include bitmaps) because the program has no way of knowing the contents of the metafile. For this reason, clipmeta gives priority to DIBs (but not to DDBs) instead of metafiles (it assumes implicitly that the metafile contains a bitmap). Paintbrush for example can send the graphic either in the bitmap format, or inside a metafile that just contains the bitmap. Clipmeta will be able to split the bitmap in pieces in the first case, but not in the second. Therefore, make sure that you instruct Paintbrush to refrain from sending the metafile (just verify that the entry "Omit Picture Format" in the "Options" submenu of PaintBrush is checked).

Some programs (eg., Paintbrush or Ghostscript) prefer to send a DDB instead of a DIB; clipmeta can handle it under a 16-color mode (because the system palette is fixed), or under a HiColor or TrueColor mode (because there is no palette). Unfortunately, the scheme fails spectacularly under a 256-color mode, because the bitmap colors depend on the hardware palette that was current at the time the bitmap was sent to the clipboard; the palette will almost certainly be different when you import the bitmap in your document, and all colors will be wrong; in most cases, you will end up with a solid black bitmap. I am trying to find a way around this problem, but it will take some time. This is probably the biggest weakness of the DDBs and that's why I strongly suggest to avoid them whenever possible.

Other considerations

The standard extension for windows metafiles is "wmf"; you should not use any other extension for compatibility purposes. Aldus developed an extended metafile format (called a "placeable" metafile), which circumvents a limitation of standard metafiles (it includes information about the natural size of the picture). Unfortunately, they chose the same extension (wmf) as the one for standard metafiles, and this can be the source of some confusion. Most applications expect a file with a "wmf" extension to be in the placeable format, while other applications (eg. Ventura) expect the plain format. The DVI driver can read both formats without any external support. If the metafile contains text, you will get much better results by using the scalable fonts introduced in Windows 3.1. Some programs do not specify the size of the metafile when they pass it to the clipboard. In that case, clipmeta cannot use the placeable format; it will save the metafile in the standard (plain) format. Metafiles produced by Simulink (Matlab for Windows) cannot be properly scaled, because Simulink uses certain instructions reserved for the program inporting the metafile. Apart from this, its metafiles have a black background with white drawings which renders them useless for printing. Mathematica metafiles can be very demanding on the system; in many cases Windows runs out of resources (without any error indication) and parts of the graphic may be wrong or missing. The only workaround that I know is to simplify the graphic: even if you add memory in your system, the amount of resources stays the same because of Windows limitations.