Readme File for BMP to QBASIC Converter

Version 1.5

1.1 Installation Notes
1.2 Program Notes and Copyrights
2.1 Introduction and Explanation
3.1 Explanation of the Various Conversions
3.11 Direct PSETing With All Colors Saved
3.12 Direct PSETing With Non-White Colors Saved
3.13 Direct PSETing Without Saving Colors
3.14 Direct Locating With All Colors Saved
3.15 Direct Locating With Non-White Colors Saved
3.16 DATA:READ With All Colors Saved
3.17 DATA:READ With Non-White Colors Saved
3.18 DATA:READ Without Saving Colors
3.19 Separate File With All Colors Saved
3.20 Separate File With Non-White Colors Saved
3.21 Separate File Without Saving Colors
3.22 Nathan's High Compression Algorithm

1.1 Installation Notes

To install this program, start Windows. Select FILE:RUN. Type **x:\SETUP.EXE** where x is your drive letter or select browse and then select SETUP.EXE from the listing of files. The program confirms creation of a default directory. You may change this if you wish. The setup program will create another group within Program Manager.

1.2 Program Notes and Copyrights

This program is the extensive hard work of two individuals. Grant Macklem was the main programmer, along with the idea of the program itself. Nathan Janos arrived at the idea for the separate file mode and had the idea of and programmed the High Compression Algorithm. The authors in no way endorse any program name or trademark, and we would like to state that each mention of a trademarked name is only for editorial purposes and to the benefit of the trademark owner with no intention of infringing upon that trademark.

2.1 Introduction and Explanation

This program was created for Grant and Nathan to use during their programming experience and was then discovered. It was thought to be a great idea to distribute this to other programmers who may find its function useful in QBASIC.

This program takes a Bitmapped, Device Independent Bitmap, or icon file and uses the various properties of Visual Basic to accurately convert that file into QBASIC instructions. There are several ways the conversion can be done, depending on your need for file size and speed. They will be explained in greater detail later. This program is only set up to work in 16 colors as of now, and a 256-color version is in the process of production. The manner in which Visual Basic returns color values greatly lengthens the programming time required for that.

NOTE: THIS PROGRAM STORES THE OUTPUTTED TEXT FILE IN THE SAME DIRECTORY AS THE ORIGINAL FILE, *SO MAKE SURE YOU HAVE ENOUGH DISK SPACE!*

3.1 Explanation of the Various Conversions

This section will generally cover the conversion, while the following sections will go into the specifics and include the code snippets you need to include along with your program to use the converted file.

The Direct PSETing conversion is by far the fastest method of display because all of the instructions are laid out for QBASIC. It also takes the biggest chunk of your program size. If you have a relatively small program, with a big title graphic, this would probably be right for you.

The Direct Locating conversion is also fast, but is limited to 80 x 60 text resolution (WIDTH 80, 60 in screen 12). This will respectively take a larger bite out of your program size, but as it is limited to only 80 x 60, it will tend to take up less space than the larger, PSETed files.

The DATA:READ conversion is the next fastest. The speed factor comes from having to read all of the data. It also takes up a smaller portion of your code since all it does is list the coordinates and the colors.

If you are running out of room in your code for these large graphics, a separate file may be the way to store all of that data -- away from the code of your program hence you have to program to input the data. This is the slowest conversion, but ways to speed it up will be forthcoming in a later version. This will also depend on your hard disk speed because your program will constantly be inputting variables and PSETing them to the screen.

Nathan's High Compression Algorithm is a life-savor to all of you who need speed, and large graphics. A 320 x 200 graphic was converted in both this and the separate file mode. This conversion displayed it to the screen in a short 96 seconds, while the separate file chugged along with a 199 second score. This conversion will create a separate file also, but only store the color values, eliminating up to 66% (and more on larger pictures) of the storage space the file needs, while also increasing speed by 50% on larger files.

You may find that you need to try all of the different conversions to find the one that will work for you. But as you get more experienced, you'll know which one you'll need for your particular task.

All of these different conversions have several different options for color saving. For some graphics, you just want one color that can be changed, and therefore don't want the color values changed. Other times, you'll want the correct image to display no matter what background color you have. In these two cases, you'd use the Don't Save Color option and the Save All Colors options respectively. The Save All Non-White Color option is for use when you want the graphic to be superimposed on the background color. The following sections describe exactly how the particular conversion works, along with the snippet to use in your code to correctly display the image.

3.11 Direct PSETing with All Colors Saved

This conversion will give a PSET instruction for every point in the image. It will usually create exceptionally large files, which will need to be incorporated into your QBASIC program. A example is listed below, as you may change the variables to fit your need.

```
SCREEN 12
x = 100
y = 100
```

The "x" and "y" coordinates will tell QBASIC where to start the top left of the image. It will be PSETed down and right from that point.

3.12 Direct PSETing with Non-White Colors Saved

This conversion will give a PSET instruction for all the colored points in the image except for white. This conversion is good for when you want the background pattern/color to show through all the white parts of the image. It will usually create exceptionally large files, which will need to be incorporated into your QBASIC program. A example is listed below, as you may change the variables or screen mode to fit your need.

```
SCREEN 13
x = 100
y = 100
```

The "x" and "y" coordinates will tell QBASIC where to start the top left of the image. It will be PSETed down and right from that point.

3.13 Direct PSETing Without Saving Colors

This conversion will give a PSET instruction for all the colored points in the image but won't save the color value. This is good for when you want a single colored graphic. It will usually create exceptionally large files, which will need to be incorporated into your QBASIC program. An example is listed below, as you may change the variables to fit your need.

```
SCREEN 13
x = 100
y = 100
z = 40
PAINT (0, 0), 15
```

The "x" and "y" coordinates will tell QBASIC where to start the top left of the image. It will be PSETed down and right from that point. The "z" value will tell QBASIC what color to PSET the graphic in. The PAINT command is optional, as it will define the background color.

3.14 Direct Locating With All Colors Saved

The “Don't Save Color” option was eliminated with this conversion style only because of practicality. This conversion will give QBASIC a series of commands to LOCATE and then PRINT a character that you define and save all the colors.

```
WIDTH , 50
x = 0
y = 0
x$ = CHR$(219)
```

"x" and "y" are the variables whose values will tell QBASIC the row and column *before* the locating is going to start. For example, in this snippet, the locating and printing will begin at (1,1). x$ can be set to any character, but CHR$(219) is a filled block, undisplayable in windows, that will look good.

3.15 Direct Locating Saving Non-White Colors

This conversion will give QBASIC a series of commands to LOCATE and then PRINT a character that you define saving all colors except for white.

```
WIDTH , 50
x = 0
y = 0
x$ = CHR$(219)
```

"x" and "y" are the variables whose values will tell QBASIC the row and column *before* the locating is going to start. For example, in this snippet, the locating and printing will begin at (1,1). x$ can be set to any character, but CHR$(219) is a filled block, undisplayable in windows, that will look good.

3.16 DATA:READ With All Colors Saved

This conversion will put a series of numbers in DATA:READ format. The first number is the number to be added to x, the second is to be added to y, and the third is the color value.

```
SCREEN 12
x = 100
y = 100
FOR t = 1 TO ####
    READ a, b, c
    PSET (x + a, y + b), c
NEXT t
RESTORE
```

You must define the number at the end of the FOR:NEXT loop. Put a high number in first and run the application. When QBASIC gives you an error code 4 message (Out of Data), go to the immediate screen by pressing F6, and type PRINT t. Write down this value, subtract one from it, and put it at the end of the FOR statement. The RESTORE command is essential at the end of the snippet to restore the reading position for the next time you'll need to read those values.

3.17 DATA:READ With Non-White Colors Saved

This conversion will put a series of numbers in DATA:READ format. The first number is the number to be added to x, the second is to be added to y, and the third is the color value.

```
SCREEN 12
x = 100
y = 100
FOR t = 1 TO ####
    READ a, b, c
    PSET (x + a, y + b), c
NEXT t
RESTORE
```

You must define the number at the end of the FOR:NEXT loop. Put a high number in first and run the application. When QBASIC gives you an error code 4 message (Out of Data), go to the immediate screen by pressing F6, and type PRINT t. Write down this value, subtract one from it, and put it at the end of the FOR statement. The RESTORE command is essential at the end of the snippet to restore the reading position for the next time you'll need to read those values.

3.18 DATA:READ Without Saving Colors

This conversion will put a series of numbers in DATA:READ format. The first number is the number to be added to x, the second is to be added to y.

```
SCREEN 12
x = 100
y = 100
z = 4

FOR t = 1 TO ####
    READ a, b
    PSET (x + a, y + b), z
NEXT t
RESTORE
```

The "z" value is necessary at the beginning of the snippet to identify the color value since you decided not to save one. You must define the number at the end of the FOR:NEXT loop. Put a high number in first and run the application. When QBASIC gives you an error code 4 message (Out of Data), go to the immediate screen by pressing F6, and type PRINT t. Write down this value, subtract one from it, and put it at the end of the FOR statement. The RESTORE command is essential at the end of the snippet to restore the reading position for the next time you'll need to read those values.

3.19 Separate File with All Colors Saved

This conversion is the slowest method, but is the only way to display a graphic while allowing the background to show through. The file created will have a series of three values in it. The first is the number to be added to x, the second to be added to y, and the third is the color value.

```
SCREEN 12
filename$ = "C:\WINDOWS\SIMEARTH.TXT"

x = 100
y = 100

OPEN filename$ FOR INPUT AS #1
DO
    INPUT #1, a, b, c
    PSET (x + a, y + b), c
LOOP UNTIL EOF(1) OR INKEY$ <> ""
```

```
CLOSE #1
```

This code will require you to define the path and name of the file to be opened. Also, if you change the file number (in this example, 1) you will need to change the EOF statement to reflect the new file number. The INKEY checking at the end will allow the users to cancel the display by pressing any key. If you do not want this, delete the OR INKEY$ <> "".

3.20 Separate File With Non-White Colors Saved

This conversion is the slowest method, but is the only way to display a graphic while allowing the background to show through. The file created will have a series of three values in it. The first is the number to be added to x, the second to be added to y, and the third is the color value.

```
SCREEN 12
filename$ = "C:\WINDOWS\SIMEARTH.TXT"

x = 100
y = 100

OPEN filename$ FOR INPUT AS #1
DO
    INPUT #1, a, b, c
    PSET (x + a, y + b), c
LOOP UNTIL EOF(1) OR INKEY$ <> ""
CLOSE #1
```

This code will require you to define the path and name of the file to be opened. Also, if you change the file number (in this example, 1) you will need to change the EOF statement to reflect the new file number. The INKEY checking at the end will allow the users to cancel the display by pressing any key. If you do not want this, delete the OR INKEY$ <> "".

3.21 Separate File Without Saving Colors

This conversion is the slowest method, but is the only way to display a graphic while allowing the background to show through. The file created will have a series of two values in it. The first is the number to be added to x and the second to be added to y.

```
SCREEN 12
filename$ = "C:\WINDOWS\SIMEARTH.TXT"

x = 100
```

```
y = 100
z = 4

OPEN filename$ FOR INPUT AS #1
DO
    INPUT #1, a, b
    PSET (x + a, y + b), z
LOOP UNTIL EOF(1) OR INKEY$ <> ""
CLOSE #1
```

You will need to set "z" to a value for the color since you decided not to save one. This conversion may be moderately faster than the previous one because it is only inputting two values for each coordinate. This code will require you to define the path and name of the file to be opened. Also, if you change the file number (in this example, 1) you will need to change the EOF statement to reflect the new file number. The INKEY checking at the end will allow the users to cancel the display by pressing any key. If you do not want this, delete the OR INKEY$ <> "".

3.22 Nathan's High Compression Algorithm

This conversion is faster than the preceding examples (separate files) but is still slower than PSETing. It also takes up the least space. The only disadvantage is that all colors must be saved because of the compression.

```
SCREEN 12
filename$ = "C:\WINDOWS\SIMEARTH.TXT"

x = 100
y = 100

OPEN filename$ FOR INPUT AS #1
INPUT #1, xmax, ymax
    FOR a = 0 to xmax -1
      FOR b = 0 to ymax
        INPUT #1, c
        PSET (x + a, y + b), c
      NEXT b
    NEXT a
CLOSE #1
```

This snippet is pretty self explanatory, as all it does is input color values and display them according to where it is in the (a, b) loop.