Java Application: IconEdit

Written by: Keith Fenske, http://kwfenske.github.io/
First version: Thursday, 22 March 2007
Document revised: Saturday, 13 February 2010

Description

IconEdit is a Java 1.4 graphical (GUI) application to edit icon files (*.ICO) for Windows. An icon file has several images in sizes like 16×16, 24×24, 32×32, 48×48, and 64×64 pixels. Icons are square from 8×8 to 256×256 pixels. Colors may be 4-bit (16 colors), 8-bit (256 colors), or 24-bit (millions). For 4-bit color, only the standard Windows color palette is used, although other palettes will be read and converted. For 8-bit color, only the standard 216 "web safe" colors are used. Other palettes will be read and converted. There are no restrictions on 24-bit colors; all RGB (red-green-blue) values are accepted. Pixels may be transparent and let the background show through. Please note that IconEdit is an old program and does not support alpha channels, compressed data (PNG images), or icons larger than 256 pixels.

When you first run IconEdit, you are given a selection of icon sizes and a partially-hidden dialog for choosing colors. You may read icons from a file with the "Open Icon File" menu item. Icons will appear in the same order as they are defined in the file. Tabs on the left-hand side identify icons by their size; click on a tab to select an icon. To change an icon, first select a color with the color chooser. Then left click (primary mouse click) on an icon square to paint the selected color. Right click (or control click) to erase a square and make it transparent, which is shown as the current background color (see the slider). Shift click on a square to select its color without painting. You may save all non-empty icons to a new file with the "Save Icon File" menu item.

These drawing tools are crude. This program is meant more for loading icons, making minor changes, and saving them again. For better drawing tools, use your favorite bitmapped image editor and copy-and-paste to this application. Images on the system clipboard don't retain transparency data. This was added to the BufferedImage class starting in Java 5.0 (1.5).

Linux and Macintosh users should note that "favicon.ico" files used to bookmark web pages can be created with this program, because they are in fact Windows icons.

Apache License or GNU General Public License

WARRANTY, without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the license(s) for more details. You should have received a copy of the licenses along with this program. If not, see the http://www.apache.org/licenses/ and http://www.gnu.org/licenses/ web pages.

Installation

You must have the Java run-time environment (JRE) installed on your computer. IconEdit was developed with Java 1.4 and should run on later versions. It may also run on earlier versions, but this has not been tested. You can download the JRE from Oracle (formerly Sun Microsystems):

JRE for end users: http://www.java.com/getjava/
SDK for programmers: http://www.oracle.com/technetwork/java/
IDE for programmers: http://www.netbeans.org/

Once Java is installed, you need to put the program files for IconEdit into a folder (directory) on your hard drive. The name of the folder and the location are your choice, except it is easier if the name does not include spaces. Assume that files will go into a "C:\Java" folder. Then create the folder and unpack the Java *.class files into this folder (if you received the program as a ZIP file). The files look something like this:

ApacheLicense20.txt (12 KB, legal notice)
GnuPublicLicense3.txt (35 KB, legal notice)
IconEdit1.class (30 KB, executable program)
IconEdit1.doc (51 KB, this documentation in Microsoft Word format)
IconEdit1.gif (23 KB, sample program image)
IconEdit1.ico (87 KB, icon for Windows)
IconEdit1.jar (25 KB, archive file with same class files inside)
IconEdit1.java (137 KB, source code)
IconEdit1.manifest (1 KB, main class manifest for archive file)
IconEdit1.pdf (96 KB, this documentation in Adobe Acrobat format)
IconEdit1Draw.class (4 KB, helper class for main program)
IconEdit1Filter.class (1 KB)
IconEdit1Icon.class (10 KB)
IconEdit1User.class (1 KB)
RunJavaPrograms.pdf (60 KB, more notes about running Java)

To run the program on Windows, start a DOS command prompt, which is Start button, Programs, Accessories, Command Prompt on Windows XP/Vista/7. Change to the folder with the program files and run the program with a "java" command:

```
c:
```

```
cd \java
java IconEdit1
```

The program name “IconEdit1” must appear exactly as shown; uppercase and lowercase letters are different in Java names. Some systems (Macintosh) will run a main “class” file by clicking on the class file name while viewing a directory in the file browser (Mac Finder). Many systems will run a “jar” file by clicking (or double clicking) on the jar file name (Windows Explorer). The command line is the only guaranteed way of running a Java program. Should you find this program to be popular, you can create a Start menu item or desktop shortcut on Windows XP/Vista/7 with a target of “java.exe IconEdit1” starting in the “C:\Java” folder.

One complication may arise when trying to run this program. Java looks for an environment variable called CLASSPATH. If it finds this variable, then that is a list of folders where it looks for *.class files. It won’t look anywhere else, not even in the current directory, unless the path contains “.” as one of the choices. The symptom is an error message that says:

```
Exception in thread "main" java.lang.NoClassDefFoundError: IconEdit1
```

To find out if your system has a CLASSPATH variable defined, type the following command in a DOS window:

```
set CLASSPATH
```

To temporarily change the CLASSPATH variable to the current directory, use the following command line:

```
java -cp . IconEdit1
```

To permanently change the CLASSPATH, you must find where it is being set. This is in Control Panel, System, Advanced, Environment Variables on Windows XP/Vista/7.

Removal or Uninstall

To remove this program from your computer, delete the installation files listed above. If the folder that contained the files is now empty, you may also delete the folder ... if you created the folder, of course, not the system. If you created desktop shortcuts or Start menu items, then delete those too. There are no hidden configuration or preference files, and no information is stored in the Windows system registry. You don’t need an “uninstall” program.

Graphical Versus Console Application

The Java command line may contain options for the position and size of the application window, and the size of the display font. Only one additional parameter may be given: the name of an icon file to open for reading. See the "-?" option for a help summary:

```
java IconEdit1 -?
```

The command line has more options than are visible in the graphical interface. An option such as -u14 or -u16 is recommended because the default Java font is too small.

Restrictions and Limitations

Icons are actually more complicated than having solid colors mixed with transparent squares. Standard icons in Windows use two bit masks, first to AND existing screen pixels, and second to XOR icon pixels onto the screen. For the most part, this amounts to replacing or not replacing screen pixels with icon pixels. However, an icon can be designed to invert screen pixels by setting bits in both the AND mask and the XOR mask. Such an icon doesn't contain its own picture information, isn't recognized by this program, and may open as ugly color data. Another new type of icon in Windows Vista has 24-bit color (8-bit red, 8-bit green, 8-bit blue) with an 8-bit alpha channel to control the degree of transparency. IconEdit ignores alpha channels and uses the AND mask for fully transparent or fully opaque (solid).

Icon File Format for Windows

Written by: Keith Fenske, http://kwfenske.github.io/
First version: Thursday, 22 March 2007
Document revised: Tuesday, 27 March 2007

The following information was obtained from web pages, from official Microsoft documentation, and by examining icon files from various sources. Some information may be incorrect, especially concerning new features added for Windows Vista (2007). All numbers are unsigned integers. Multi-byte numbers are in "little endian" order where the least-significant byte appears first. Copyright for much of the material in this section belongs to Microsoft. Additional comments are released under Apache License 2.0 or GNU GPLv2 or later. It is difficult to distinguish between them unless you refer to original Microsoft documents.

Icon Header (ICONDIR)

The file begins with an icon header to identify this as an icon file and to provide basic information about each image in the file.

idReserved	2 bytes	reserved: must be zero.
idType	2 bytes	resource type: must be 1 for icons. Cursor files have a very similar format, and you can read most cursor files as icons if you accept a value of 2 for the idType field. However, don't write cursor files as icons, because some of the unused fields in an icon have a different purpose in a cursor.
idCount	2 bytes	number of images in this icon file. Obviously, there should be at least one.
idEntries[]	IconDirEntry	one "icon directory" entry for each image in this icon file.

Icon Directory Entry (ICONDIRENTRY)

There is one icon directory entry for each image in the icon file. Some of the fields here duplicate fields in the bitmap information header, and because of that, you may see icons that set different values for fields that should be the same.

bWidth	one byte	width of this icon image in pixels. The most common sizes are 16 and 32. Also in use are 24, 48, and 64. Windows Vista claims 96, 128, and 256 — although it is not clear how the number 256 can be encoded as a byte value, unless the value zero takes on a special meaning. In general, the original specification allows icon sizes from 1 to 255 pixels.
bHeight	one byte	height of this icon image in pixels. While the specifications do allow different values for height and width, almost all icons are square and have the same size in both directions.
bColorCount	one byte	number of colors in image. For 1-bit color, this should be 2. For 4-bit color, this should be 16. For 8-bit, 24-bit, and 32-bit color, this must be zero. This field is not reliable and is often set incorrectly. For example, you will see icons that set this field to the number of colors actually used by the icon.
bReserved	one byte	reserved: must be zero.
wPlanes	2 bytes	number of color planes: should be 1, but may appear as zero in some files. Ignore this value when reading an icon.
wBitCount	2 bytes	number of bits per pixel: 1 for monochrome color (2 colors), 4 for 4-bit color (16 colors), 8 for 256 colors, 24 for RGB color (millions), or 32 for RGB color with an alpha channel. This field should match biBitCount in the "icon image" entry, but note that it is sometimes incorrect. Trust biBitCount more than wBitCount.
dwBytesInRes	4 bytes	total number of bytes in the "icon image" entry that follows.
dwImageOffset	4 bytes	location of the "icon image" entry as a byte offset from the beginning of the file.

Image Data (ICONIMAGE)

Each image in the icon file has an "icon image" entry that actually totally defines the image. This entry is a specialized "device independent bitmap" (DIB). It can, in theory, appear anywhere in the file after the icon directory is finished. In practice, image entries appear in the same order as the directory entries, but you should not assume that the bytes are consecutive in the file. That is, always locate an image entry from its dwImageOffset field.

icHeader[]	BitmapInfoHeader	"DIB" header (see following table).

icColors[]	RGBQuad	palette with color table. Only used for 1-bit, 4-bit, and 8-bit colors. Not used and must not appear for 24-bit and 32-bit color. For 1-bit color, there must be exactly 2 entries. For 4-bit color, there must be exactly 16 entries. For 8-bit color, there must be exactly 256 entries. Each entry has four bytes as follows: rgbBlue: blue component of color rgbGreen: green component of color rgbRed: red component of color rgbReserved: must be zero for palettes Note that the order is BGR (blue-green-red), not RGB.
icXOR[]	byte array	XOR color data. The size of this array depends upon the height of the icon, the width of the icon, and the number of bits per pixel (see biBitCount below). For 1-bit, 4-bit, and 8-bit color, there is one entry of that bit size for each pixel. Each entry is an index into the color table or palette (see icColors above). For 24-bit color, there are three bytes per pixel, in the same order as the first three bytes of the RGBQuad above. When writing transparent pixels for color depths from 1 to 24 bits, you should set the icXOR color to black (all zeros). For 32-bit color, there are four bytes per pixel in the same format as RGBQuad, except that the fourth byte is the alpha channel and controls the degree of transparency (from 0 for completely transparent to 255 for completely opaque). The byte array begins with the bottom row or “scan line” of the image, not the top row as you might expect. Each row does begin with the first column, as expected. For 1-bit color, the most significant bit of the first byte is the first column. For 4-bit color, the most significant nibble of the first byte is the first column. For all color depths, there is a special requirement that is almost undocumented: all rows or “scan lines” must be a multiple of 32 bits (4 bytes), and any unused bits at the end of a row must be set to zero.

icAND[]	byte array	AND bitmask. There is one bit per pixel, in the same order as icXOR above and with the same restriction on having multiples of 32 bits per row. A bit value of zero means to remove the existing screen pixel and replace it with a pixel from the icon (that is, the icon pixel is "solid" or "opaque"). A bit value of one means to leave the existing screen pixel, in which case the XOR color data should be zero to make the icon transparent for this pixel. If the AND bit is one and the XOR data is not zero, then screen pixels will be inverted where the XOR data has one bits (useful for cursors, but not particularly useful for icons). The icAND field must appear for all color depths, including 24-bit and 32-bit, even though 32-bit color has alpha channel information.

Bitmap Information Header (BITMAPINFOHEADER)

Not all of these fields are used for icons, but they must all appear. Unused fields are generally set to zero. There are extra fields because the same structure is used throughout Windows.

biSize	4 bytes	size of the BitmapInfoHeader structure in bytes (always 40 decimal or 0x28 hexadecimal).
biWidth	4 bytes	width of bitmap (that is, the icon) in pixels. Should match bWidth above.
biHeight	4 bytes	height of bitmap in pixels. Must be twice the value of bHeight above, because it's the combined height of the XOR data plus the AND mask. In the official Microsoft documentation for all bitmap images, there is an explanation that says, "If biHeight is positive, the bitmap is a bottom-up DIB with the origin at the lower left corner. If biHeight is negative, the bitmap is a top-down DIB with the origin at the upper left corner." It is not known if any icon files use negative values for this field.
biPlanes	2 bytes	number of color planes. Must be 1. Ignore this value when reading an icon.
biBitCount	2 bytes	number of bits per pixel: 1, 4, 8, 24, or 32. Should match wBitCount but wBitCount has been known to be wrong. Trust biBitCount more than wBitCount.
biCompression	4 bytes	compression used: should be zero. Ignore this value when reading an icon.

biSizeImage	4 bytes	size of the pixel data. Not always set consistently. Sometimes you will see the size of the XOR color data alone. When creating a new icon, set this to the total size of the XOR color data plus the AND bitmask, in bytes.
biXPelsPerMeter	4 bytes	horizontal resolution, in pixels per meter: should be zero. Ignore this value when reading an icon.
biYPelsPerMeter	4 bytes	vertical resolution, in pixels per meter: should be zero. Ignore this value when reading an icon.
biClrUsed	4 bytes	number of colors used. Ignore this value when reading an icon. When creating a new icon, set this to zero, or to the palette size for 1-bit, 4-bit, or 8-bit color. The official Microsoft documentation says, "If the bitmap uses 8 bpp [bits per pixel] or less, the bitmap has a color table immediately following the BitmapInfoHeader structure. The color table consists of an array of RGBQuad values. The size of the array is given by the biClrUsed member. If biClrUsed is zero, the array contains the maximum number of colors for the given bit depth; that is, $2^{biBitCount}$ colors."
biClrImportant	4 bytes	number of important colors. Ignore this value when reading an icon. When creating a new icon, set this to zero, or to the palette size for 1-bit, 4-bit, or 8-bit color.

Additional Information

Microsoft: Icons in Win32
http://msdn2.microsoft.com/en-us/library/ms997538.aspx

Microsoft: Windows GDI: BITMAPINFOHEADER
http://msdn2.microsoft.com/en-us/library/ms532290.aspx

Microsoft: Windows Vista: Icon Development Guidelines
http://msdn2.microsoft.com/en-us/library/aa511280.aspx
