How to get the image size of a JPG, GIF and PNG image file

Answer 1

This set of functions shows how to extract the dimensions (width and height) of JPG, GIF and PNG files. This code was done quite a while back and while it works fine for my purposes, it may be not handle some of the newer stuff like progressive JPEGs and such. Experimentation is highly recommened.

```
unit ImgSize;

interface

uses Classes;

procedure GetJPGSize(const sFile: string; var wWidth, wHeight: word);
procedure GetPNGSize(const sFile: string; var wWidth, wHeight: word);
procedure GetGIFSize(const sGIFFile: string; var wWidth, wHeight: word);

implementation

uses SysUtils;

function ReadMWord(f: TFileStream): word;

type
  TMotorolaWord = record
  case byte of
  0: (Value: word);
  1: (Byte1, Byte2: byte);
end;

var
  MW: TMotorolaWord;
begin
  {It would probably be better to just read these two bytes in normally and
  then do a small ASM routine to swap them. But we aren't talking about
  reading entire files, so I doubt the performance gain would be worth the
  trouble.}
  f.Read(MW.Byte2, SizeOf(Byte));
  f.Read(MW.Byte1, SizeOf(Byte));
  Result := MW.Value;
end;

procedure GetJPGSize(const sFile: string; var wWidth, wHeight: word);
const
  ValidSig : array[0..1] of byte = ($FF, $D8);
  Parameterless = [$01, $D0, $D1, $D2, $D3, $D4, $D5, $D6, $D7];
var
  Sig: array[0..1] of byte;
  f: TFileStream;
  x: integer;
  Seg: byte;
  Dummy: array[0..15] of byte;
  Len: word;
  ReadLen: LongInt;
begin
  FillChar(Sig, SizeOf(Sig), #0);
  f := TFileStream.Create(sFile, fmOpenRead);
  try
    ReadLen := f.Read(Sig[0], SizeOf(Sig));
    for x := Low(Sig) to High(Sig) do
      if Sig[x] <> ValidSig[x] then
        ReadLen := 0;
      if ReadLen > 0 then
      begin
        ReadLen := f.Read(Seg, 1);
        while (Seg = $FF) and (ReadLen > 0) do
        begin
          ReadLen := f.Read(Seg, 1);
          if Seg <> $FF then
          begin
            if (Seg = $C0) or (Seg = $C1) then
            begin
              ReadLen := f.Read(Dummy[0], 3);  { don't need these bytes }
              wHeight := ReadMWord(f);
              wWidth := ReadMWord(f);
            end
            else
            begin
              if not (Seg in Parameterless) then
              begin
                Len := ReadMWord(f);
                f.Seek(Len - 2, 1);
                f.Read(Seg, 1);
              end
              else
                Seg := $FF;  { Fake it to keep looping. }
            end;
          end;
        end;
      end;
    finally
    f.Free;
  end;
end;

procedure GetPNGSize(const sFile: string; var wWidth, wHeight: word);
type
  TPNGSig = array[0..7] of byte;
const
  ValidSig: TPNGSig = (137, 80, 78, 71, 13, 10, 26, 10);
var
  Sig: TPNGSig;
  f: tFileStream;
  x: integer;
begin
  FillChar(Sig, SizeOf(Sig), #0);
  f := TFileStream.Create(sFile, fmOpenRead);
  try
    f.Read(Sig[0], SizeOf(Sig));
    for x := Low(Sig) to High(Sig) do
      if Sig[x] <> ValidSig[x] then
        exit;
      f.Seek(18, 0);
      wWidth := ReadMWord(f);
      f.Seek(22, 0);
      wHeight := ReadMWord(f);
  finally
    f.Free;
  end;
end;


procedure GetGIFSize(const sGIFFile: string; var wWidth, wHeight: word);
type
  TGIFHeader = record
  Sig: array[0..5] of char;
  ScreenWidth, ScreenHeight: word;
  Flags, Background, Aspect: byte;
end;
  TGIFImageBlock = record
  Left, Top, Width, Height: word;
  Flags: byte;
end;
var
  f: file;
  Header: TGifHeader;
  ImageBlock: TGifImageBlock;
  nResult: integer;
  x: integer;
  c: char;
  DimensionsFound: boolean;
begin
  wWidth  := 0;
  wHeight := 0;
  if sGifFile = '' then
    exit;

  {$I-}

  FileMode := 0;  { read-only }
  AssignFile(f, sGifFile);
  reset(f, 1);
  if IOResult <> 0 then
    {Could not open file}
  exit;
  {Read header and ensure valid file}
  BlockRead(f, Header, SizeOf(TGifHeader), nResult);
  if (nResult <> SizeOf(TGifHeader)) or (IOResult <> 0)
    or (StrLComp('GIF', Header.Sig, 3) <> 0) then
  begin
    {Image file invalid}
    close(f);
    exit;
  end;
  {Skip color map, if there is one}
  if (Header.Flags and $80) > 0 then
  begin
    x := 3 * (1 SHL ((Header.Flags and 7) + 1));
    Seek(f, x);
    if IOResult <> 0 then
    begin
      { Color map thrashed }
      close(f);
      exit;
    end;
  end;
  DimensionsFound := False;
  FillChar(ImageBlock, SizeOf(TGIFImageBlock), #0);
  { Step through blocks }
  BlockRead(f, c, 1, nResult);
  while (not EOF(f)) and (not DimensionsFound) do
  begin
    case c of
    ',':  { Found image }
    begin
      BlockRead(f, ImageBlock, SizeOf(TGIFImageBlock), nResult);
      if nResult <> SizeOf(TGIFImageBlock) then
      begin
        { Invalid image block encountered }
        close(f);
        exit;
      end;
      wWidth := ImageBlock.Width;
      wHeight := ImageBlock.Height;
      DimensionsFound := True;
    end;
    ',' :  { Skip }
    begin
      { NOP }
    end;
    { nothing else, just ignore }
  end;
  BlockRead(f, c, 1, nResult);
end;
close(f);

{$I+}

end;

end.
```

Tip author unknown

Answer 2

Getting the size of a *.jpg and *.gif image:

```
{resourcestring
  SInvalidImage = 'Image is not valid';}

type
  TImageType = (itUnknown, itJPG, itGIF);

function GetImageType(Image: PByte): TImageType;
var
  pImage: PChar;
begin
  pImage := PChar(Image);
  Result := itUnknown;
  if StrLComp(pImage, 'GIF', 3) = 0 then
  begin
    Result := itGIF;
  end
  else
    if (pImage[0] = #$FF) and (pImage[1] = #$D8) then
    begin
      Result := itJPG;
    end;
end;

procedure GetImageBounds(Image: PByte; Size: Integer; var Width: Cardinal;
  var Height: Cardinal);
const
  SizeSegments = [#$C0, #$C1, #$C2];
var
  pImage: PChar;
  ImageType: TImageType;
  cSegmentType: Char;
  nSegmentSize: Word;
  nPos: Integer;
  bFound: Boolean;
begin
  ImageType := GetImageType(Image);
  pImage := PChar(Image);
  case ImageType of
  itJPG:
  begin
    nPos := 2;
    bFound := False;
    while not bFound and (nPos < Size) do
    begin
      if pImage[nPos] <> #$FF then
      begin
        EInvalidGraphic.Create(SInvalidImage);
      end;
      Inc(nPos);
      if nPos >= Size then
      begin
        raise EInvalidGraphic.Create(SInvalidImage);
      end;
      cSegmentType := pImage[nPos];
      bFound := cSegmentType in SizeSegments;
      if not bFound then
      begin
        Inc(nPos);
        if not (cSegmentType in [#$01, #$d0..#$d7]) then
        begin
          if nPos >= Size - 1 then
          begin
            raise EInvalidGraphic.Create(SInvalidImage);
          end;
          nSegmentSize := MakeWord(Byte(pImage[nPos + 1]), Byte(pImage[nPos]));
          Inc(nPos, nSegmentSize);
        end;
      end;
    end;
    if not bFound then
    begin
      raise EInvalidGraphic.Create(SInvalidImage);
    end;
    Inc(nPos, 4);
    if nPos >= Size - 1 then
    begin
      raise EInvalidGraphic.Create(SInvalidImage);
    end;
    Height := MakeWord(Byte(pImage[nPos + 1]), Byte(pImage[nPos]));
    Inc(nPos, 2);
    if nPos >= Size - 1 then
    begin
      raise EInvalidGraphic.Create(SInvalidImage);
    end;
    Width := MakeWord(Byte(pImage[nPos + 1]), Byte(pImage[nPos]));
  end;
  itGIF:
  begin
    nPos := 6;
    if nPos >= Size - 1 then
    begin
      raise EInvalidGraphic.Create(SInvalidImage);
    end;
    Width := MakeWord(Byte(pImage[nPos]), Byte(pImage[nPos + 1]));
    nPos := 8;
    if nPos >= Size - 1 then
    begin
      raise EInvalidGraphic.Create(SInvalidImage);
    end;
    Height := MakeWord(Byte(pImage[nPos]), Byte(pImage[nPos + 1]));
  end
  else
  begin
    raise EInvalidGraphic.Create(SInvalidImage);
  end;
  end;
end;
```

Tip by Frank Simon

Answer 3

This is a customization of Answer 1:

```
function GoodFileRead(fhdl: THandle; buffer: Pointer; readsize: DWord): Boolean;
var
  numread: DWord;
  retval: Boolean;
begin
  retval := ReadFile(fhdl, buffer^, readsize , numread , Nil);
  result := retval And (readsize = numread);
end;

function ReadMWord(fh: HFile ; Var value: Word): Boolean;
type
  TMotorolaWord = record
    case byte of
      0: (Value: word);
      1: (Byte1, Byte2: byte);
  end;
var
  MW: TMotorolaWord;
  numread : DWord;
begin
  { It would probably be better to just read these two bytes in normally and
  then do a small ASM routine to swap them.  But we aren't talking about
  reading entire files, so I doubt the performance gain would be worth the
  trouble.}
  Result := False;
  if ReadFile(fh, MW.Byte2, SizeOf(Byte), numread, nil) then
```

```
    if ReadFile(fh, MW.Byte1, SizeOf(Byte), numread, nil) then
      Result := True;
  Value := MW.Value;
end;

function ImageType(Fname: String): Smallint;
var
  ImgExt: String;
  Itype: Smallint;
begin
  ImgExt := UpperCase(ExtractFileExt(Fname));
  if ImgExt = '.BMP' then
    Itype := 1
  else
  if (ImgExt = '.JPEG') or (ImgExt='.JPG') then
    Itype := 2
  else
    Itype := 0;
  Result := Itype;
end;

function FetchBitmapHeader(PictFileName: String; Var wd, ht: Word): Boolean;
{similar routine is in "BitmapRegion" routine}
label ErrExit;
const
  ValidSig: array[0..1] of byte = ($FF, $D8);
  Parameterless = [$01, $D0, $D1, $D2, $D3, $D4, $D5, $D6, $D7];
  BmpSig = $4d42;
var
  {Err : Boolean;}
  fh: HFile;
  {tof : TOFSTRUCT;}
  bf: TBITMAPFILEHEADER;
  bh: TBITMAPINFOHEADER;
  {JpgImg  : TJPEGImage;}
  Itype: Smallint;
  Sig: array[0..1] of byte;
  x: integer;
  Seg: byte;
  Dummy: array[0..15] of byte;
  skipLen: word;
  OkBmp, Readgood: Boolean;
begin
  {Open the file and get a handle to it's BITMAPINFO}
  OkBmp := False;
  Itype := ImageType(PictFileName);
  fh := CreateFile(PChar(PictFileName), GENERIC_READ, FILE_SHARE_READ, Nil,
           OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
  if (fh = INVALID_HANDLE_VALUE) then
    goto ErrExit;
  if Itype = 1 then
  begin
    {read the BITMAPFILEHEADER}
    if not GoodFileRead(fh, @bf, sizeof(bf)) then
      goto ErrExit;
    if (bf.bfType <> BmpSig) then  {'BM'}
      goto ErrExit;
    if not GoodFileRead(fh, @bh, sizeof(bh)) then
      goto ErrExit;
    {for now, don't even deal with CORE headers}
    if (bh.biSize = sizeof(TBITMAPCOREHEADER)) then
      goto ErrExit;
    wd := bh.biWidth;
    ht := bh.biheight;
    OkBmp := True;
  end
  else
  if (Itype = 2) then
  begin
    FillChar(Sig, SizeOf(Sig), #0);
    if not GoodFileRead(fh, @Sig[0], sizeof(Sig)) then
      goto ErrExit;
    for x := Low(Sig) to High(Sig) do
      if Sig[x] <> ValidSig[x] then
```

```
        goto ErrExit;
      Readgood := GoodFileRead(fh, @Seg, sizeof(Seg));
      while (Seg = $FF) and Readgood do
      begin
        Readgood := GoodFileRead(fh, @Seg, sizeof(Seg));
        if Seg <> $FF then
        begin
          if (Seg = $C0) or (Seg = $C1) or (Seg = $C2) then
          begin
            Readgood := GoodFileRead(fh, @Dummy[0],3);  {don't need these bytes}
            if ReadMWord(fh, ht) and ReadMWord(fh, wd) then
              OkBmp := True;
          end
          else
          begin
            if not (Seg in Parameterless) then
            begin
              ReadMWord(fh,skipLen);
              SetFilePointer(fh, skipLen - 2, nil, FILE_CURRENT);
              GoodFileRead(fh, @Seg, sizeof(Seg));
            end
            else
              Seg := $FF;  {Fake it to keep looping}
          end;
        end;
      end;
  end;
  ErrExit: CloseHandle(fh);
  Result := OkBmp;
end;
```

Tip author unknown

Original resource:	The Delphi Pool
Author:	Various
Added:	2013-01-27
Last updated:	2013-01-27