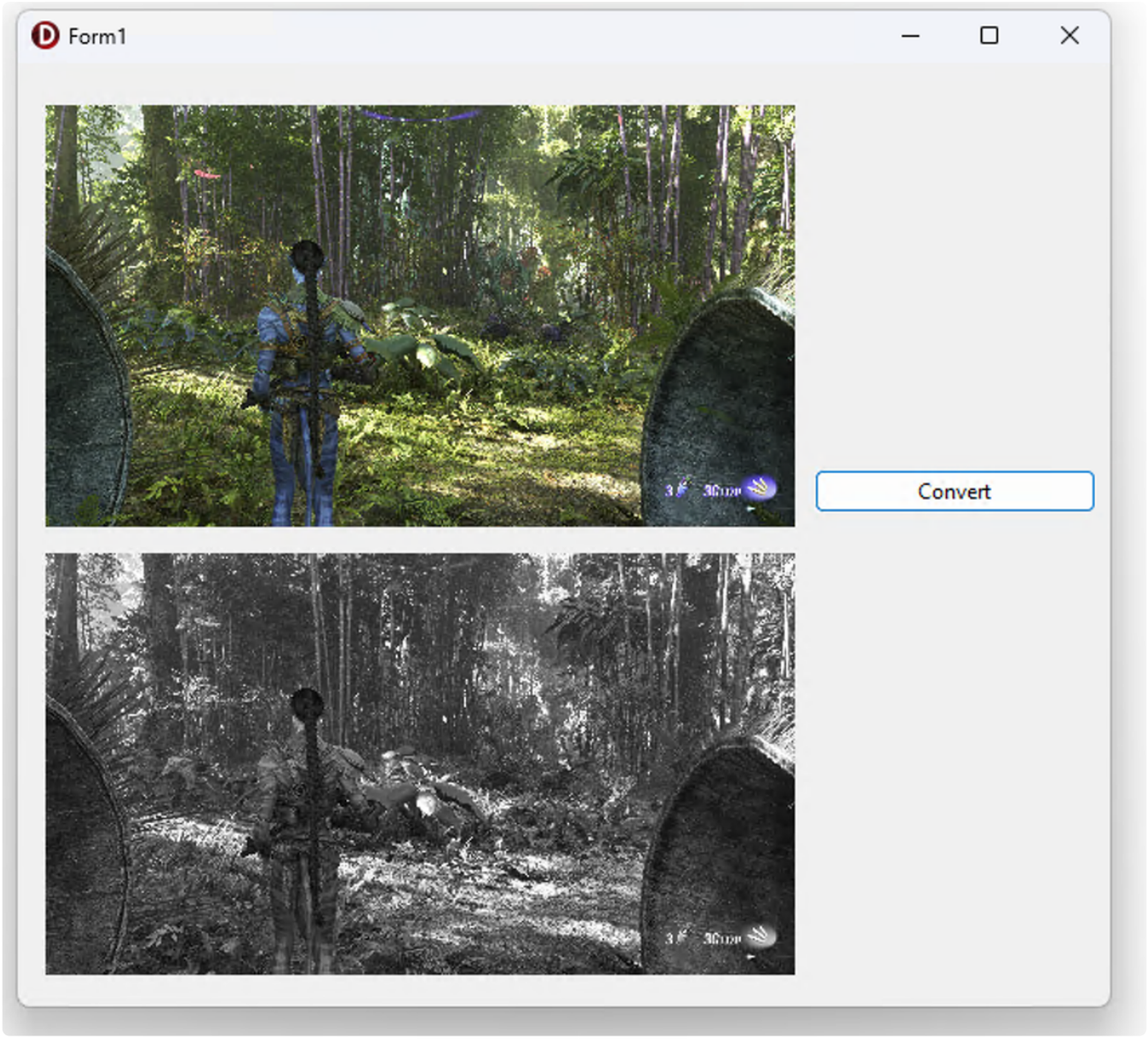
NEWS · FEBRUARY 21, 2026

AI Won't Replace Delphi Developers. But...

By Dr. Holger Flick



Grayscale Bitmap

Invent Your Own Controls and Libraries!

The other day I saw a post on a Delphi developer forum. A fellow developer had been struggling for *days* trying to convert a bitmap to grayscale. Days. He had tried multiple third-party components, downloaded libraries, read through outdated documentation, and was still stuck. The forum thread was filling up with suggestions for yet more components to install, each with its own quirks, licensing, and version compatibility headaches.

I asked AI. It took about two minutes.

Not two days. Not two hours. Two minutes.

The Problem

The developer wanted something simple: take a color image, make it grayscale. That's it. A problem that has been solved since the 1970s. Yet in the Delphi world, the default response is still "which third-party component do I need?"

Here's the thing — you don't need one. Delphi is a powerful language with everything you need already built in. You just need to know how to use it. And if you don't know how, AI can show you.

The Solution: 60 Lines of Pure Delphi

No third-party libraries. No components to install. No license keys. Just a clean class helper that extends `TBitmap` with two methods:

```
unit Grayscale.BitmapHelper;

interface

uses
  Winapi.Windows, Vcl.Graphics;

type
  TRGBTripleArray = array[0..MaxInt div SizeOf(TRGBTriple) - 1] of TRGBTriple;
  PRGBTripleArray = ^TRGBTripleArray;

  TBitmapHelper = class helper for TBitmap
  public
    procedure ConvertToGrayscale;
    function ToGrayscale: TBitmap;
  end;

implementation

procedure TBitmapHelper.ConvertToGrayscale;
var
  Y, X: Integer;
  Row: PRGBTripleArray;
  Gray: Byte;
begin
  PixelFormat := pf24bit;
  for Y := 0 to Height - 1 do
  begin
    Row := ScanLine[Y];
    for X := 0 to Width - 1 do
    begin
      Gray := (Integer(Row[X].rgbtRed)   * 299 +
               Integer(Row[X].rgbtGreen) * 587 +
               Integer(Row[X].rgbtBlue)  * 114) div 1000;
      Row[X].rgbtRed   := Gray;
      Row[X].rgbtGreen := Gray;
      Row[X].rgbtBlue  := Gray;
    end;
  end;
end;

function TBitmapHelper.ToGrayscale: TBitmap;
begin
  Result := TBitmap.Create;
  try
```

```
    Result.Assign(Self);
    Result.ConvertToGrayscale;
  except
    Result.Free;
    raise;
  end;
end;

end.
```

That's the entire unit. Drop it in your project, add `Grayscale.BitmapHelper` to your `uses` clause, and every `TBitmap` in your application instantly gets both methods. No subclassing, no refactoring, no component installation.

Usage is dead simple:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bmp: TBitmap;
begin
  Bmp := TBitmap.Create;
  try
    Bmp.Assign(Image1.Picture.Graphic);
    Bmp.ConvertToGrayscale;
    Image2.Picture.Bitmap.Assign(Bmp);
  finally
    Bmp.Free;
  end;
end;
```

Works with JPEGs, PNGs, BMPs — anything `TGraphic` supports.

Why This Is Fast

This isn't some naive `Canvas.Pixels[X,Y]` loop. It uses `ScanLine` for direct memory access to the pixel buffer — roughly **100x faster**. The grayscale weights (`299/587/114`) follow the ITU-R BT.601 standard, the same formula used in broadcast television. It's perceptually correct because it accounts for how the human eye is more sensitive to green than red, and more sensitive to red than blue.

Integer math only. No floating point. On a 3840x2160 image, this runs in milliseconds.

The Real Point: Stop Being Afraid of AI

I didn't write this code by hand. I described what I needed to an AI, and it generated a clean, idiomatic Delphi class helper using `ScanLine`, proper memory management, and correct luminance math. It knew about `TRGBTriple` living in `Winapi.Windows`. It knew about class helpers. It used `try/except` with `Result.Free` to prevent memory leaks. It followed Delphi conventions.

Was the first attempt perfect? No. We iterated. The AI made mistakes — it initially forgot the `Winapi.Windows` import, and it took a few rounds to get the implementation section structured correctly for the compiler. But here's the key difference: **each iteration took seconds, not hours**. Instead of posting on a forum and waiting for replies, I had a conversation and refined the solution in real time.

Compare that to the developer on the forum:

- **Day 1**: Posts question. Gets suggestions for ComponentX.
- **Day 2**: Installs ComponentX. Doesn't work with his Delphi version. Tries ComponentY.

- **Day 3**: ComponentY works but has a licensing issue. Tries ComponentZ.
- **Day 4**: Still stuck. Thread has 47 replies and no working solution.

This is 2026. We have better tools now. Use them.

"But I Don't Trust AI-Generated Code"

Good. You shouldn't blindly trust *any* code — not from AI, not from Stack Overflow, not from that third-party component you downloaded from a random website. The difference is that with AI, you can interrogate every line. "Why `pf24bit`?" "Why integer math instead of floats?" "Why a class helper instead of a subclass?" It will explain its reasoning, and you'll learn something in the process.

The developer who spent days searching for a component learned nothing about bitmap internals. The developer who spent two minutes with AI now understands `ScanLine`, pixel formats, luminance calculations, and class helpers. Who's the better programmer at the end of the day?

A Challenge to the Delphi Community

Stop reaching for third-party components as your first instinct. Before you install anything, ask yourself: "Could I build this myself with AI assistance in under an hour?"

For most things — grayscale conversion, thumbnail generation, CSV export, JSON parsing helpers, custom validators — the answer is yes. And the code you get will be:

- **Yours** — no licensing, no dependencies, no version conflicts
- **Understood** — because you participated in creating it
- **Maintainable** — because it's 60 lines, not a 500-file component library
- **Exactly what you need** — not a Swiss Army knife when all you needed was a blade

Delphi is a fantastic language. It has `ScanLine` for raw pixel access. It has class helpers for clean API extension. It has generics, anonymous methods, inline variables, and record helpers. It's more capable in 2026 than most developers realize — because they stopped exploring the language and started collecting components instead.

AI won't replace Delphi developers. But Delphi developers who use AI will replace those who don't.

The train is leaving. Get on it.

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —