

Tentu, mari kita bedah kode FileExporter.cs.

Secara umum, kelas ini memiliki satu fungsi spesifik: **mengambil data warna dari setiap titik (pixel) pada gambar dan menuliskannya ke dalam sebuah file teks (.txt)**. Ini biasanya digunakan untuk debugging atau analisis data gambar secara manual.

Berikut adalah penjelasannya baris demi baris:

1. Import Library (Pustaka)

C#

```
using System.Drawing.Drawing2D;
using System.Text;
using System.IO;
using System;
```

- using System.Drawing.Drawing2D;: Mengimpor pustaka grafis tingkat lanjut. Meskipun di kode ini kita hanya melihat penggunaan objek dasar seperti Bitmap dan Color (yang biasanya ada di System.Drawing), pustaka ini sering disertakan bersamaan untuk manipulasi grafis.
 - using System.Text;: **Penting**. Mengimpor pustaka untuk menggunakan StringBuilder. Tanpa ini, menyusun teks dari ribuan pixel akan membuat aplikasi sangat lambat.
 - using System.IO;: Mengimpor "Input/Output". Ini wajib ada karena kita akan melakukan operasi penyimpanan file (File.WriteAllText).
 - using System;: Pustaka dasar C# untuk tipe data umum dan penanganan *Error* (Exception).
-

2. Namespace dan Kelas Statis

C#

```
namespace MiniPhotoshop.Logic.IO
{
    public static class FileExporter
    {
```

- namespace MiniPhotoshop.Logic.IO: Menandakan bahwa kode ini adalah bagian dari folder/kategori IO (Input/Output) dalam logika aplikasi MiniPhotoshop.
 - public static class FileExporter: Mendefinisikan kelas **statis**.
 - **Artinya:** Anda tidak perlu membuat *instance* objek (tidak perlu new FileExporter()). Anda bisa langsung memanggil fungsinya dari mana saja dengan FileExporter.ExportPixelDataToTxt(...).
-

3. Fungsi Utama: ExportPixelDataToTxt

C#

```
public static void ExportPixelDataToTxt(Bitmap bmp, string filePath)
{
```

- Ini adalah deklarasi fungsi (method).
 - void: Fungsi ini tidak mengembalikan nilai apa-apa (hanya menjalankan tugas).
 - Parameter Bitmap bmp: Gambar sumber yang akan dibaca.
 - Parameter string filePath: Lokasi lengkap di komputer tempat file teks akan disimpan (misal: "C:\Data\gambar.txt").
-

4. Validasi (Pengecekan Keamanan)

C#

```
if (bmp == null)
```

```
{  
    throw new ArgumentNullException(nameof(bmp), "Bitmap tidak boleh null.");  
}
```

- Ini adalah *Guard Clause*.
 - Jika bmp (gambar) kosong/tidak ada, program akan langsung berhenti dan melempar *Error* (*ArgumentNullException*). Ini mencegah program *crash* di baris berikutnya saat mencoba membaca pixel dari gambar yang tidak ada.
-

5. Persiapan Penampung Teks

C#

```
var sb = new StringBuilder();
```

- *StringBuilder* adalah cara efisien untuk menyusun teks panjang di C#.
 - **Kenapa tidak pakai string biasa?** Jika Anda menggabungkan string (misal `teks = teks + "baru"`) di dalam loop ribuan kali, memori komputer akan cepat penuh dan program menjadi lambat. *StringBuilder* dirancang khusus untuk menangani penggabungan teks berulang-ulang dengan sangat cepat.
-

6. Loop (Perulangan) Membaca Pixel

C#

```
for (int y = 0; y < bmp.Height; y++)  
{  
    for (int x = 0; x < bmp.Width; x++)  
    {
```

- `for (int y = 0...:` Loop luar. Berjalan dari atas ke bawah (baris per baris) berdasarkan tinggi

gambar (Height).

- for (int x = 0...: Loop dalam. Berjalan dari kiri ke kanan (kolom per kolom) berdasarkan lebar gambar (Width).
 - Kombinasi dua loop ini memastikan **setiap pixel** di gambar akan dikunjungi satu per satu.
-

7. Mengambil dan Menulis Data Warna

C#

```
Color pixelColor = bmp.GetPixel(x, y);
sb.Append($"{{x},{y}}: R={{pixelColor.R}}, G={{pixelColor.G}}, B={{pixelColor.B}}\n");
}
}
```

- Color pixelColor = bmp.GetPixel(x, y);: Mengambil data warna pada koordinat (x, y).
 - sb.Append(...): Menambahkan baris teks baru ke dalam StringBuilder.
 - \$"{{x},{y}}: ...": Ini adalah *String Interpolation*. Format teks yang dihasilkan akan terlihat seperti:
 - 0,0: R=255, G=0, B=0 (Pixel di pojok kiri atas berwarna Merah)
 - 1,0: R=255, G=255, B=255 (Pixel sebelahnya berwarna Putih), dst.
 - \n: Simbol untuk "Enter" atau ganti baris.
-

8. Menyimpan ke File

C#

```
// Memastikan penggunaan System.IO
File.WriteAllText(filePath, sb.ToString());
}
}
```

- `sb.ToString()`: Mengubah semua data yang ditampung `StringBuilder` menjadi satu string utuh yang sangat panjang.
 - `File.WriteAllText(filePath, ...)`: Membuat file baru (atau menimpa file lama) di lokasi `filePath` dan mengisi isinya dengan teks data pixel tadi.
-

Kesimpulan Fungsi:

Jika Anda memiliki gambar ukuran 10x10 pixel, kode ini akan menghasilkan file teks berisi 100 baris, di mana setiap baris menjelaskan komposisi warna (Red, Green, Blue) dari setiap titik pada gambar tersebut.

Langkah Selanjutnya:

Apakah Anda ingin saya menjelaskan bagaimana cara memanggil fungsi ini dari Form utama (UI), misalnya ketika tombol "Save as Text" ditekan?