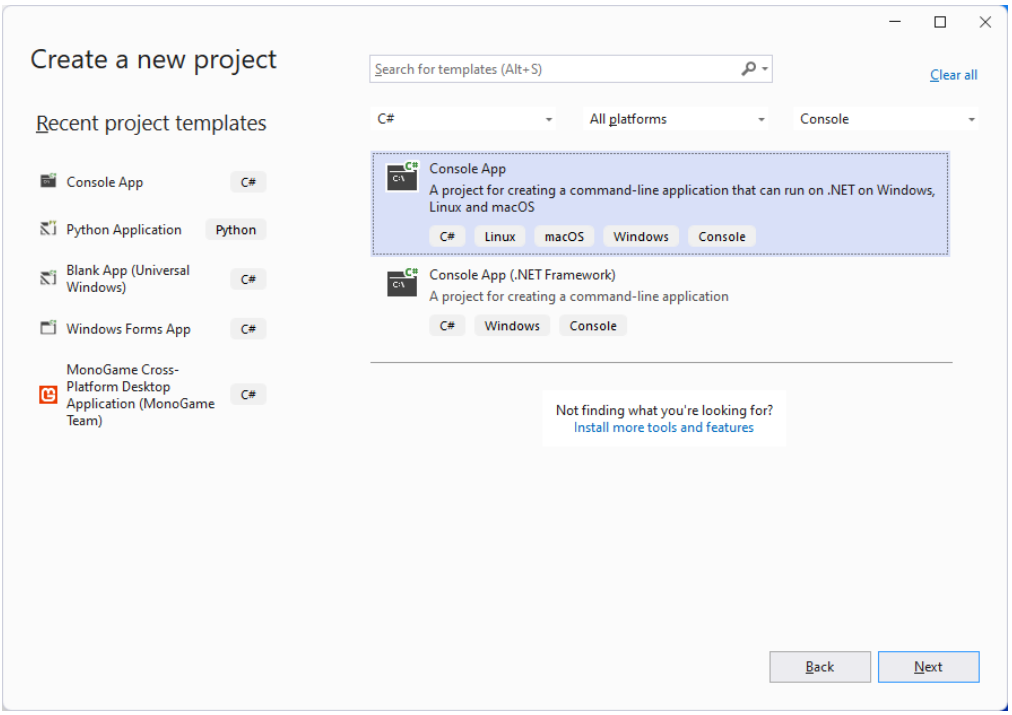
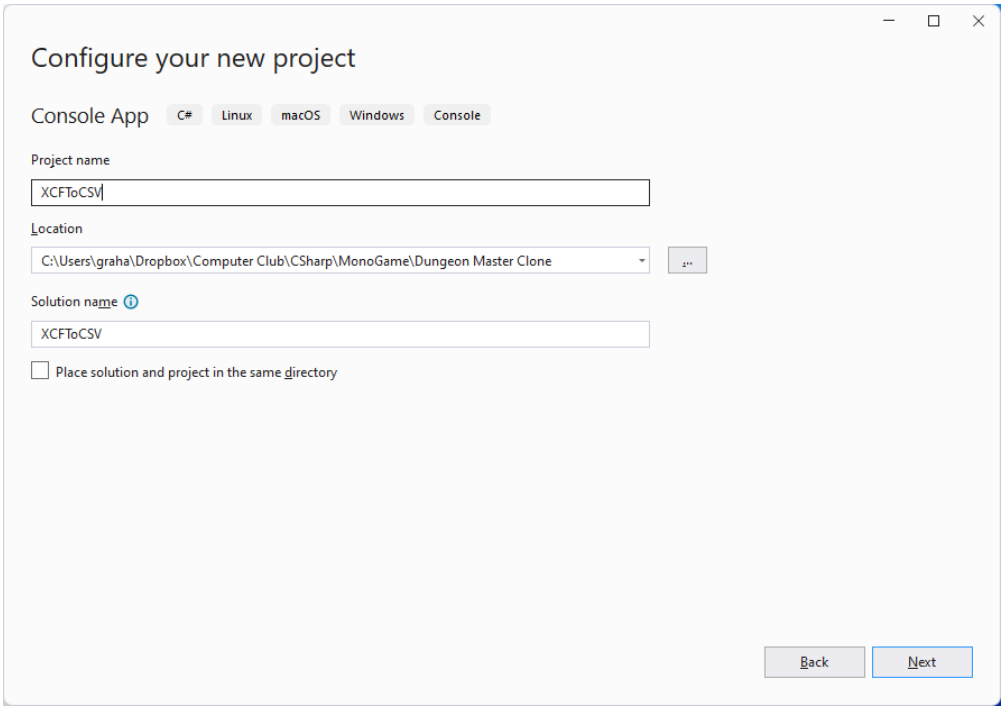
Extracting Data from .xcf files: C# Console

Go to https://github.com/Inksaver/XCFToCSV and download Program.cs and Xconsole.cs

Start a new VS2022 solution C# Console App Windows .net → Name it `XCFToCSV`.

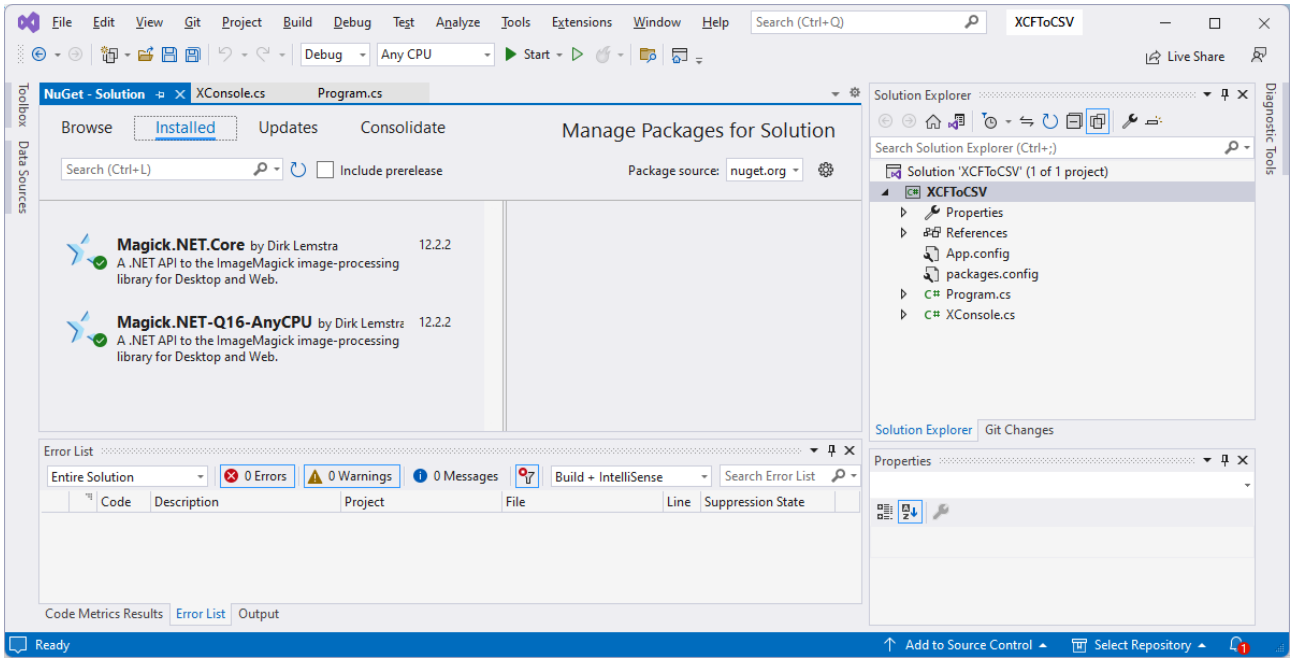




Menu → Tools → Nuget Package Manager → Manage NuGet packages for solution…

Select the Browse tab and type Magick in the search box.

Select Magick.NET-Q16-AnyCPU and install it:

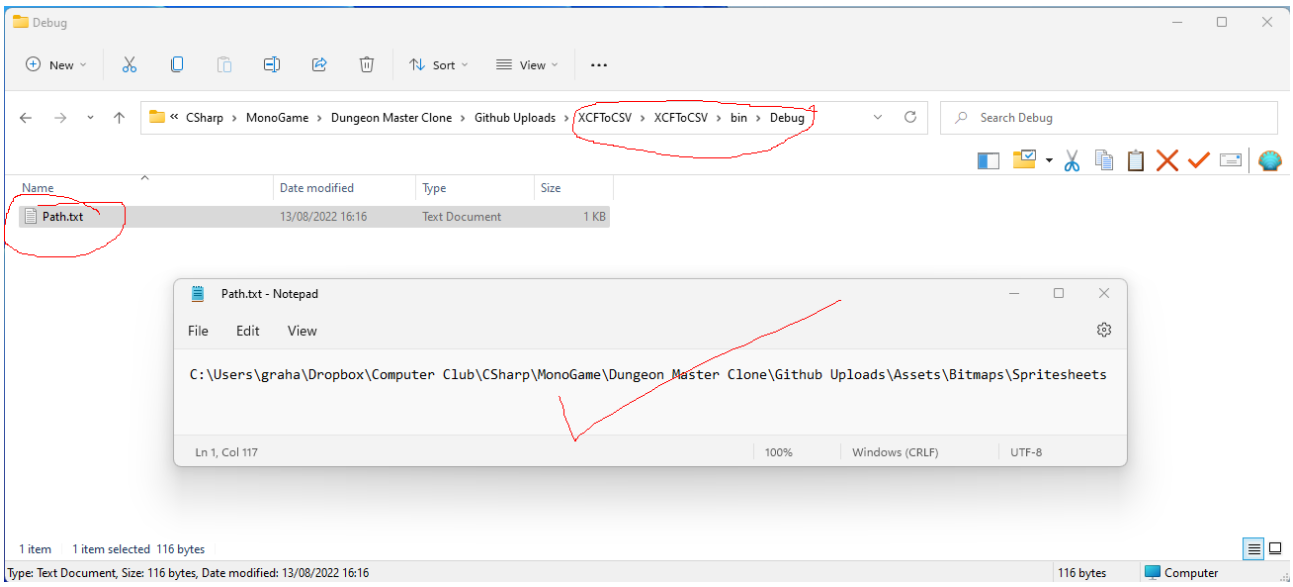


Use Windows File explorer to over-write Program.cs with the one you just downloaded, and add Xconsole.cs

You may need to right-click on the solution explorer and Add → Existing Item and browse for Xconsole.cs

Make sure the Namespaces match

Add a text file called Path.txt in the XCFToCSV\XCFToCSV\bin\Debug folder that contains the path to the Assets/Spritesheets folder you are using for your .xcf and .png spritesheets:



The compiled .exe can be placed directly in the Spritesheets folder, along with the supporting .dll files from Magick, or simply use it from the IDE.

Here is how it works:

<static void Main(string[] args)>

The static class Xconsole is called to draw an introductory Header:

```
XConsole6.Header("Gimp xcf Reader", "For extracting layer coordinates and rectangle
sizes", "White", "Yellow","DarkRed"); // note White text, Yellow Frame, DarkRed BG

public static int Header(string title, string subtitle = "", string textColour = "",
                         string frameColour = "", string backColour = "")
{
    string topHeader = "╔".PadRight(Console.WindowWidth - 2, '=') + "╗";
    string titleContent = AlignFrameText(title, "centre");
    string subtitleContent = AlignFrameText(subtitle, "centre");
    string lowerHeader = "╚".PadRight(Console.WindowWidth - 2, '=') + "╝";
    WriteLine(topHeader, frameColour, backColour);
    Write("╟", frameColour, backColour);
    Write(titleContent, textColour, backColour);
    Write("╟\n╟", frameColour, backColour);
    Write(subtitleContent, textColour, backColour);
    Write("╟\n", frameColour, backColour);
    WriteLine(lowerHeader, frameColour, backColour);

    return 4; // default 4 lines written
}
```

The first 4 lines construct strings containing lines of text to output to the Console.

The function **AlignFrameText()** is called to centre the lines in the Console by padding with spaces:

```
text = text.PadLeft(((windowWidth - text.Length) / 2) + text.Length);
text = text.PadRight(windowWidth);
```

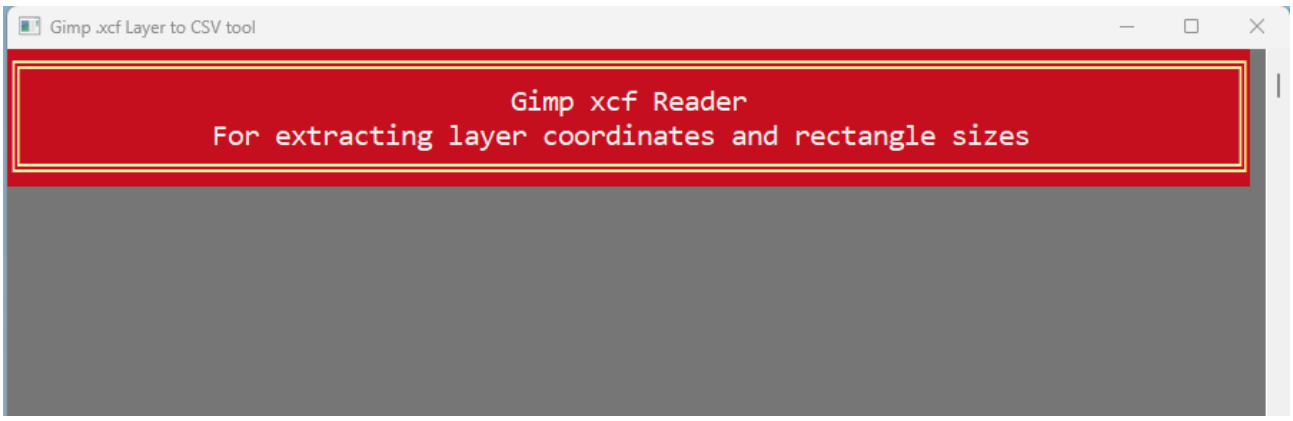
This could be done in one line, but is less clear how it works:

```
text = text.PadLeft(((windowWidth - text.Length) / 2) + text.Length).PadRight(windowWidth);
```

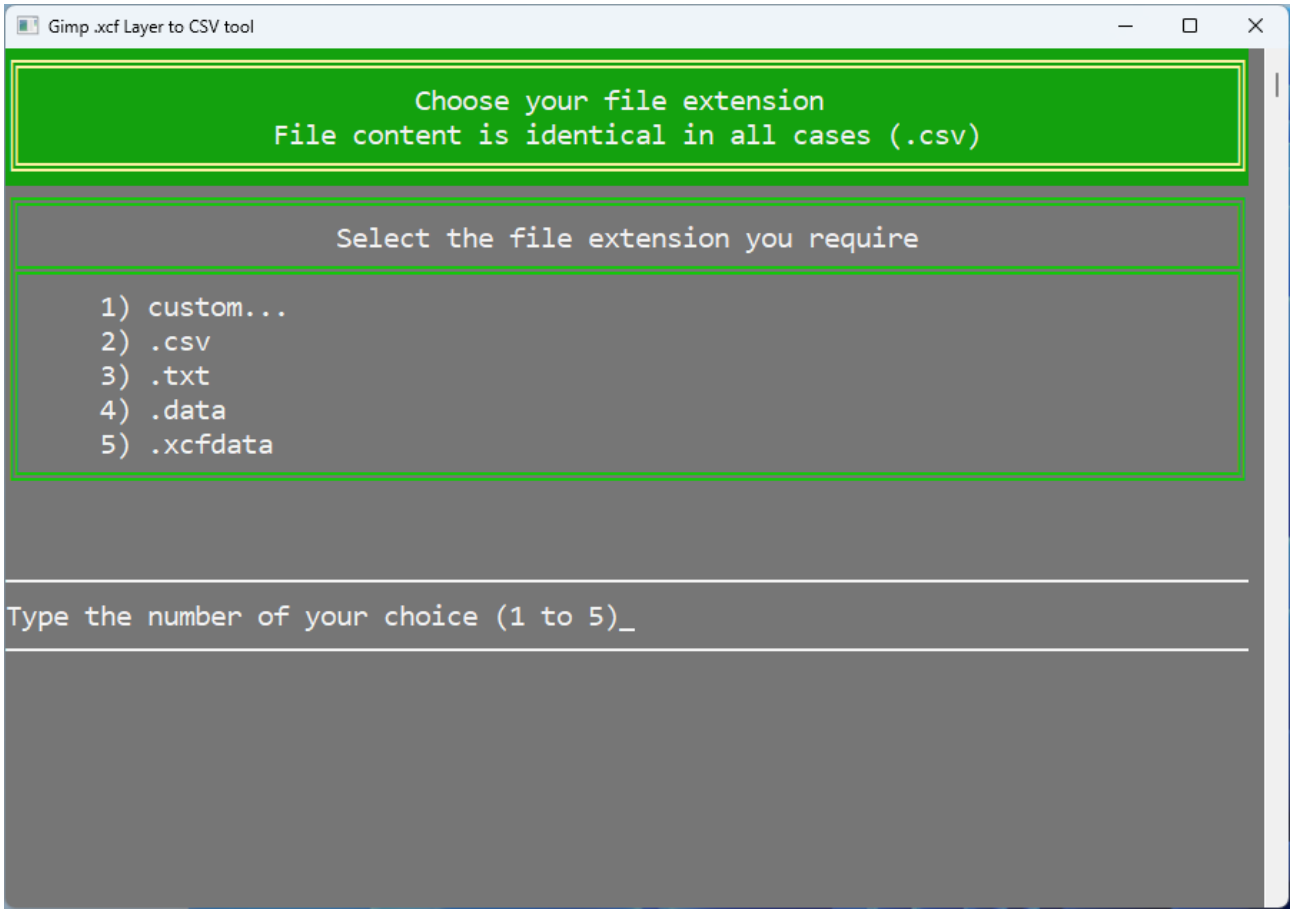
The functions Write() and WriteLine() expand on the Console.Write() and Console.WriteLine() functions by allowing passing strings representing colours:

```
WriteLine(topHeader, frameColour, backColour);
Write("||", frameColour, backColour);

// note White text, Yellow Frame, DarkRed BackGround:
```



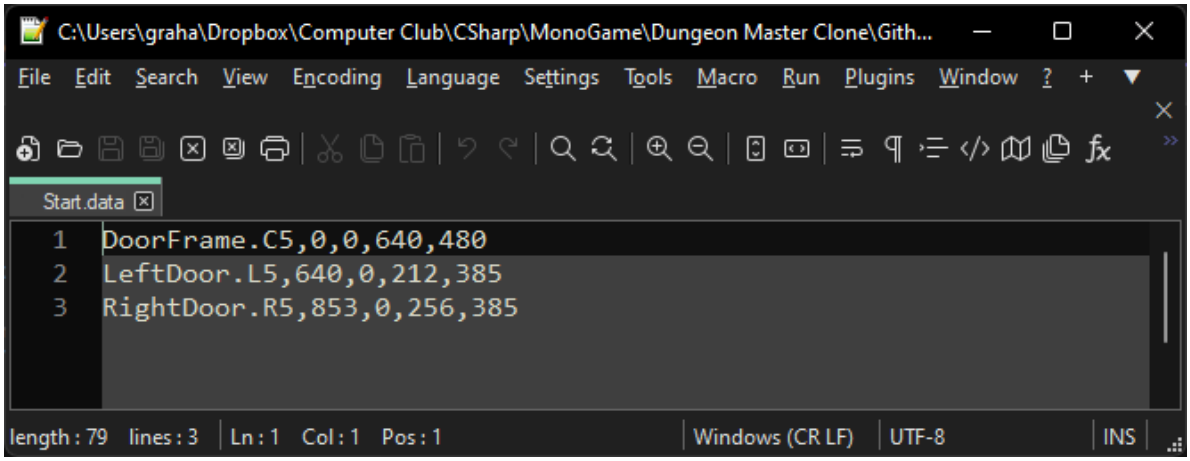
This Frame disappears after a couple of seconds and another Frame is constructed, along with a Menu of options for the preferred File Extension:



The file extension preferred for .xcf files is **.data**

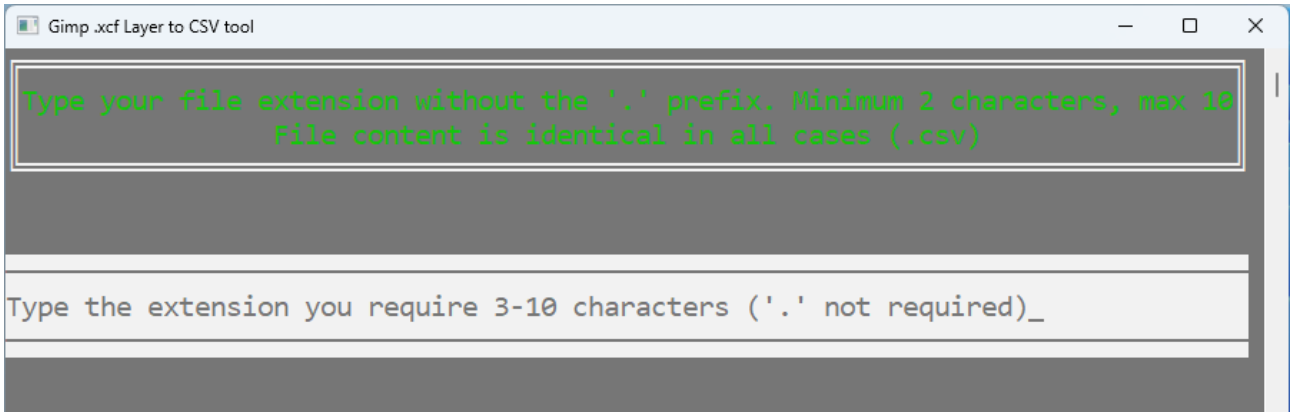
This is to distinguish the content from other files such as Items, Coordinates etc.

This is important when these files are processed into an SQLite database.
Start.data is shown below:



Type 4 and press enter to show the list of available files

If you wanted to use an alternative extension, Type 1 and Enter:



Type eg data and press Enter. This will take you to the list of available .xcf files

The code in XConsole6 that drew this Input box is called from these lines:

```
string fileExtension = ""; // initialise fileExtension as empty string
if (choice == 0) // User has chosen "custom"
{
    row = XConsole6.Clear(foreColor: FG, backColor: BG, width: 80, height: 22);
    row += XConsole6.Header("Type your file extension without the '.' prefix. Minimum 2 characters,
max 10",
       "File content is identical in all cases (.csv)", "Green", FG, BG);
    row += 2;
    fileExtension = XConsole6.GetString(prompt: "Type the extension you require 3-10 characters ('.'
not required)",
    withTitle: false, min: 3, max: 10, row: row, windowWidth: 0,
    textColour:BG, backColour: FG); // get a string 3-10 characters default BG and FG swapped
```

```
    if (!fileExtension.StartsWith("."))
        fileExtension = $".{fileExtension}";
}
else
    fileExtension = fileExtensions[choice];
```

Type in the extension you want to use and press Enter.

The input is validated by Xconsole.GetString() to be between 3 and 10 characters.

Checking for the "." is done via:

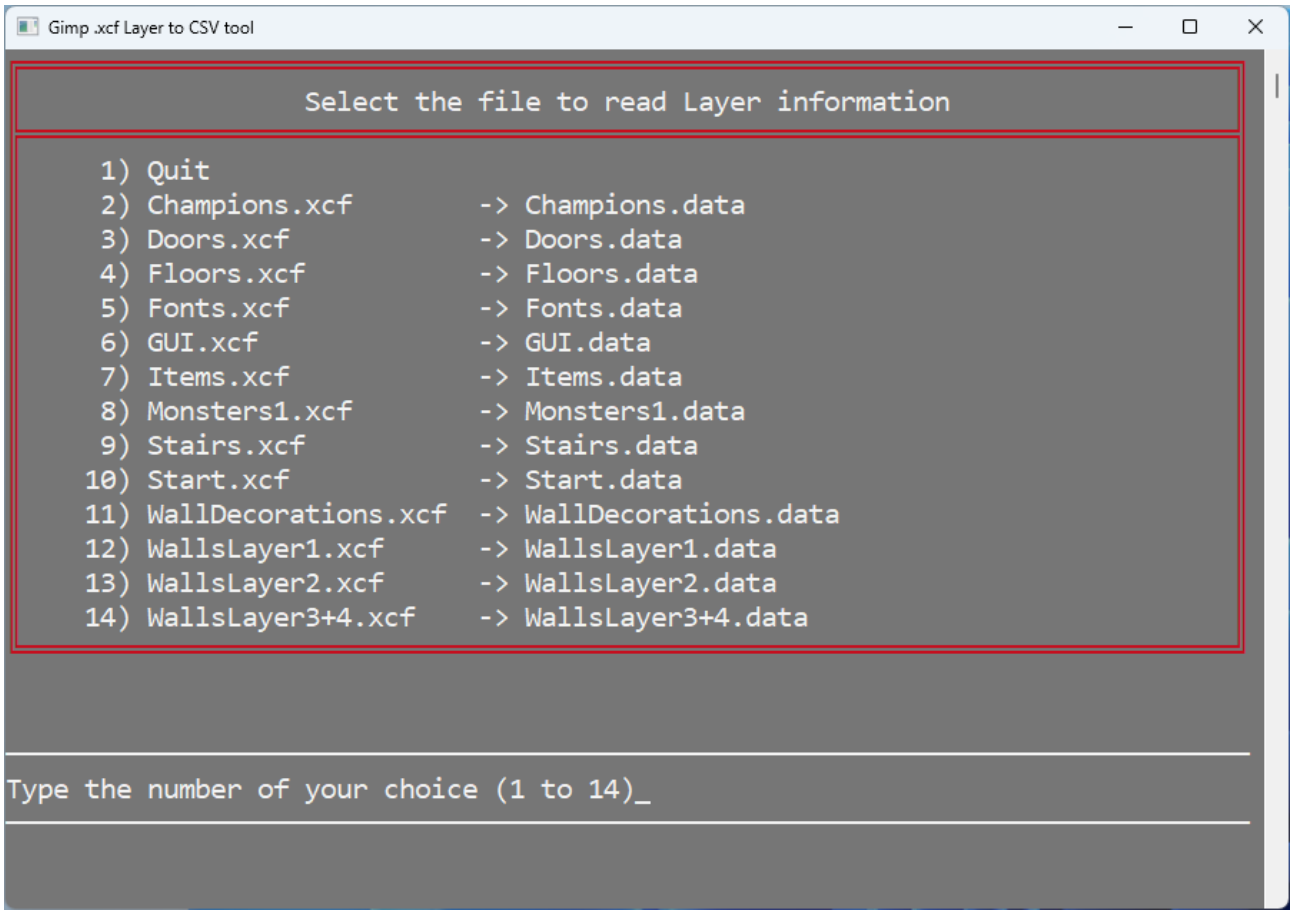
```
if (!fileExtension.StartsWith("."))
     fileExtension = $".{fileExtension}";
```

The code then contines to draw the File List inside a While loop to enable you to process any files required

```
while (!quit)
{
    row = XConsole6.Clear(foreColor: FG, backColor: BG, width: 80, height: 22);
    row += 2;
    choice = XConsole6.Menu("Select the file to read Layer information", fileList, row, 0, FG, "DarkRed", BG);
    if (choice == 0) quit = true;
    else
    {
        string inputFile = files[choice - 1].ToString();
        string saveFile = Path.Combine(imagePath, Path.GetFileNameWithoutExtension(files[choice - 1]) +
fileExtension);

        XConsole6.Clear();
        if (File.Exists(saveFile)) File.Delete(saveFile); // if file already exists, delete it

        using (MagickImageCollection images = new MagickImageCollection(inputFile))
        {
            using (StreamWriter f = new StreamWriter(saveFile))
            {
                for (int index = images.Count - 1; index > -1; index--)
                {
                    // lines starting with "#", or (Legacy from previous version) "ImageBackground" or "Group:" NOT
processed
                    if (!images[index].Label.StartsWith("#") && images[index].Label != "ImageBackground" && !
images[index].Label.StartsWith("Group:"))
                    {
                        Console.WriteLine($"{images[index].Label},{images[index].Page.X},{images[index].Page.Y},
{images[index].Width},{images[index].Height}");
                        if (index > 1) // terminate line with newline character
                            f.WriteLine($"{images[index].Label},{images[index].Page.X},{images[index].Page.Y},
{images[index].Width},{images[index].Height}");
                        else // do not add newline character
                            f.Write($"{images[index].Label},{images[index].Page.X},{images[index].Page.Y},
{images[index].Width},{images[index].Height}");
                    }
                }
            }
        }
    }
}
```

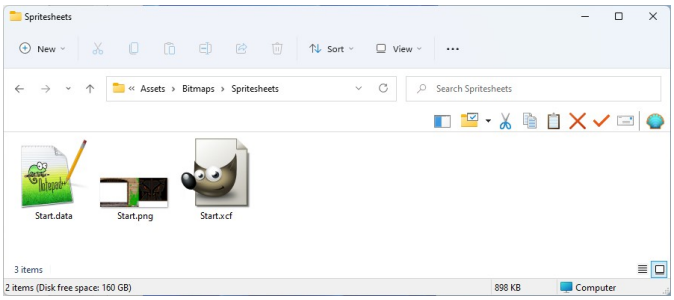


This is the full list of all Atlases needed in the project. Yours will only have 1 entry “Start.xcf”
Make sure it is output to “Start.data” as above

Type the number next to “Start.xcf”
and press Enter

Type 1 and Enter to quit

The interface will rapidly redraw
but it will have written a file called
Start.data in the Spritesheets directory:



I have associated .data to be opened with Notepad++ by default, hence the icon.

The important code that actually writes the file and uses the Magick package is:

```
if (File.Exists(saveFile)) File.Delete(saveFile); // if file already exists, delete it

using (MagickImageCollection images = new MagickImageCollection(inputFile))
{
    using (StreamWriter f = new StreamWriter(saveFile))
    {
        for (int index = images.Count - 1; index > -1; index--)
```

```
        {

// lines starting with "#", or (Legacy from previous version)  "ImageBackground" or
"Group:" NOT processed

if (!images[index].Label.StartsWith("#") && images[index].Label != "ImageBackground"
&& !images[index].Label.StartsWith("Group:"))
{
    Console.WriteLine($"{images[index].Label},{images[index].Page.X},
{images[index].Page.Y},{images[index].Width},{images[index].Height}");


if (index > 1) // terminate line with newline character
    f.WriteLine($"{images[index].Label},{images[index].Page.X},{images[index].Page.Y},
{images[index].Width},{images[index].Height}");


else // do not add newline character
    f.Write($"{images[index].Label},{images[index].Page.X},{images[index].Page.Y},
{images[index].Width},{images[index].Height}");

            }
        }
    }
}
```

The Magick package gets a lot of information from the .xcf file, but the parts required for each layer are:

```
.Label, .Page.X, .Page.Y, .Width, .Height
```