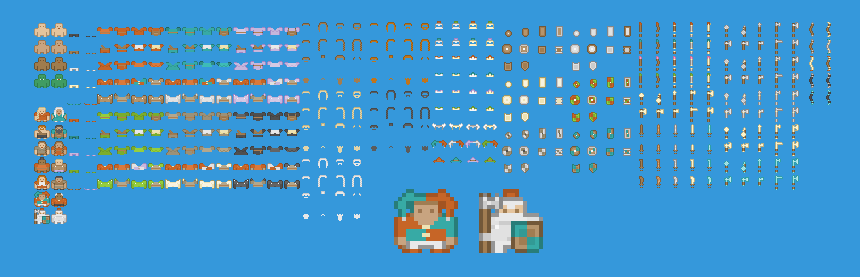
Tutorial – Introduction to C#: Character Creator part 1

Introduction:

This tutorial is the first in a series of tutorials in which we will cover creating a character customization editor.

This editor will allow a game designer to create different characters using a modular character spritesheet. The character settings will then be saved in a text file which could then be loaded into a game.



In this tutorial, we will be building the foundation for such a program. We’ll define a few classes within a console application and add the functionality to allow the user to define the properties of a simple character.

Later tutorials will build upon this one, each one adding more functionality.

Prerequisites:

Before beginning this tutorial, you should have completed the following prerequisite tutorials:

* *Introduction to Managed Languages Tutorial*. Available in the session on *Managed Languages*.  
  You will only need to follow the steps on installing C# and .NET Components if the C# project templates are not available in Visual Studio. Otherwise completion of this tutorial is optional.

Creating a Console Application:

Open Visual Studio and create a C# Console application.

If this option isn’t available to you, follow the steps in the tutorial for the session on Managed Languages to download the C# language components and project templates.

Name your Project and Solution *CharacterCreator*.

Your project should contain the following empty *Program.cs* file:

namespace CharacterCreator

{

class Program

{

static void Main(string[] args)

{

}

}

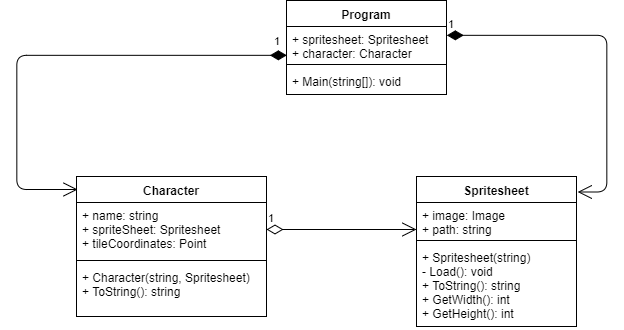
}

Before we start modifying the *Main()* function, we should create the other classes we will need for our character customization application.

To start with, we will need a class that holds information about the sprite sheet being used to generate the characters. This class will need to store the path of the image file (which will then get loaded into an *Image* member variable), and the dimensions of individual tiles.

We will also need a class that defines a character that will be generated using our program. This character class will store a reference to the sprite sheet, along with all the modular sprites that will combine to make the completed character sprite (body, shirt, hat, moustache, etc).

The following class diagram describes our program structure for this tutorial:



The Spritesheet Class:

To simplify the program, we will assume that all images in the sprite sheet are of the same dimensions.

For this tutorial we will use the *Roguelike Characters* sprite sheet available from *Kenny* (<http://www.kenney.nl/assets/roguelike-characters>). If you have an alternate sprite sheet you wish to use (that matches these constraints), you may use that instead.

Although we are making a console application (so our program will have no visual output) we will still load the image into memory. This will be useful in future tutorials when we want to query with width and height of the image. So, you will need to download and extract this sprite sheet, placing the *roguelikeChar\_transparent.png* file in your project directory.

Create the *Spreadsheet* class and fill in the code as follows:

using System.Drawing;

namespace CharacterCreator

{

class Spritesheet

{

private Image image = null;

public string path;

public int GetWidth()

{

// add your implementation here

}

public int GetHeight()

{

// add your implementation here

}

public Spritesheet(string path)

{

this.path = path;

Load();

}

public void Load()

{

image = Image.FromFile(path);

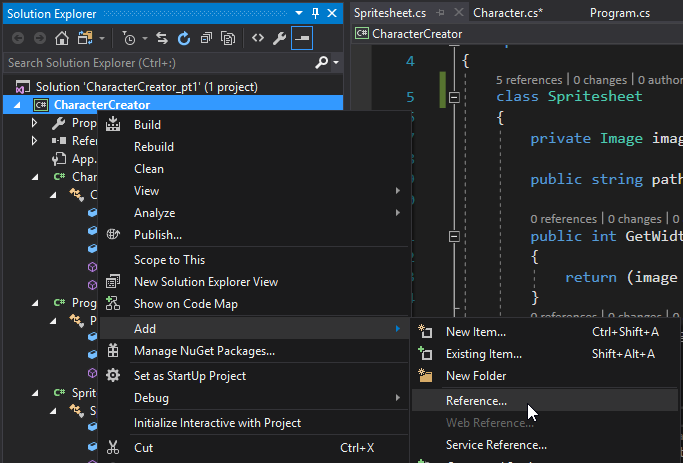
}

}

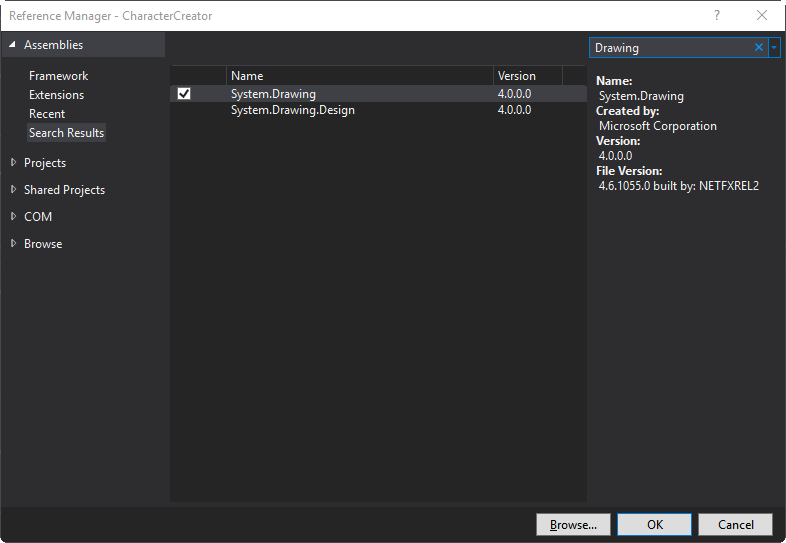
}

We’re using the *System.Drawing* library to load the image file into memory. This library isn’t included by default in a console application, so we’ll need to add a reference to this library manually.

Right-click on the project name in *Solution Explorer* and select *Add* -> *Reference*.



From the dialog box that appears, open the *Assemblies* tab. The *System.Drawing* library will be listed somewhere in here, but the fastest way to find it is to use the *Search Assemblies* box in the upper-right of this window.



Search for “*Drawing*” and select *System.Drawing* from the search results. Ensure the checkbox is ticked and press *OK*.

You will notice in the constructor for the *Spritesheet* class we’re calling the *Load()* function. This will load the image using the file path passed into the constructor.

You should note that if the file path is invalid, then the program will throw an exception (and crash). We will look at what exceptions are, and recovering gracefully from them, in a future tutorial.

Finally, the *GetWidth()* and *GetHeight()* functions have not been completed. Add the processing for these functions. You should check to see if the image has been loaded (i.e., is it not null) before returning the *Width* or *Height* property from the image.

The Character Class:

The character class will simply store the name of the character, a reference to the sprite sheet, and the tile coordinates of a sprite in the sheet.

For now, we will only use one sprite per character. In future tutorials, we’ll add the ability to build a character by layering multiple sprites.

In C#, classes are reference types (as opposed to structures and primitive types, which are value types). When passing reference types as function arguments, a reference to the object is passed (akin to pointers in C++). When passing value types as arguments, a copy of the value is made.

We will dive into reference and value types in more detail in a future session. For now, realise that when we pass the *Spritesheet* class object to the *Character* object, it is storing a reference to the *Spritesheet* (that we’ll create in the *Program* class) and not a copy of it.

Add the *Character* class to your program:  
  
The *System.Drawing* library is necessary in this class as we use a *Point* type to store the x,y tile coordinates. A point is akin to a Vector, except the coordinates are stored as integers and not floats.

using System;

using System.Drawing;

namespace CharacterCreator

{

class Character

{

public string name;

public Spritesheet spritesheet;

public Point tileCoordinates = new Point(0, 0);

public Character(string name, Spritesheet spritesheet)

{

this.name = name;

this.spritesheet = spritesheet;

}

}

}

The Program Class:

C++ is a true Object-Oriented programming language. By that we mean that it strictly enforces the requirement that everything is an object.

This is why the *Main* function must be a member function of a class. There is no such thing as a non-member function as there is in C/C++.

In the *Program* class we’ll create member variables to store one instance each of the *Spritesheet* and *Character* classes. Because the *Main()* function is static, these variables will also need to be static. As we add more features and functionality to our program we’ll come up with a better design, but for now this will be adequate.

In the *Main()* function itself we want to ask the user to input the path of the sprite sheet and the property values of the character to create via the command line. We’ll then create and initialize these objects.

The final task will be to output some information about the created objects to the command line.

Add the following code to your *Program* class:

using System;

using System.IO;

namespace CharacterCreator

{

class Program

{

static Spritesheet spritesheet = null;

static Character character = null;

static void Main(string[] args)

{

string path;

string name;

Console.WriteLine("Working directory is " + Directory.GetCurrentDirectory());

Console.WriteLine("Enter the relative path of the sprite sheet texture: ");

path = Console.ReadLine();

Console.WriteLine("Enter the name of the character to create: ");

name = Console.ReadLine();

spritesheet = new Spritesheet(Directory.GetCurrentDirectory() + "\\" + path);

if (spritesheet == null)

{

Console.WriteLine("Error loading sprite sheet. Exiting program.");

return;

}

character = new Character(name, spritesheet);

Console.WriteLine("Enter the x tile index: ");

int x = int.Parse(Console.ReadLine());

(continued on next page)

int y = int.Parse(Console.ReadLine());

character.tileCoordinates = new System.Drawing.Point(x, y);

Console.WriteLine(character.ToString());

Console.ReadKey();

}

}

}

ToString():

You will notice in the *Main()* function above we are using the *ToString()* function when outputting the created character to the command line.

*ToString()* is defined in *System.Object*. Because all objects in C# derive from *System.Object*, you can call this function on every object. Unless overrided, this function will simply print the name of the class.

We can override *ToString()* in both our *Character* and *Spritesheet* class to customise the string that is returned.

In your *Character* class, override this function as follows:

public override string ToString()

{

return base.ToString() + "\n\tpath: \t" + spritesheet.path +

"\n\ttile coordinates: \t" + tileCoordinates.ToString();

}

Although it isn’t used, you could also override *ToString()* for the *Spreadsheet* class:

public override string ToString()

{

return base.ToString() + ": " + path.ToString();

}

Execute your program.

If the spritesheet image is in your project directory, then the path you need to enter at the command line should be:

..\..\roguelikeChar\_transparent.png

If the path is incorrect, your program will throw an assertion and execution will stop in the *Spritesheet::Load()* function.

Your console should look similar to the following image if the program ran successfully:

