Creating a Role Playing Game with XNA Game Studio 3.0
Part 21
Adding a Custom Content Pipeline Processor and Importer

To follow along with this tutorial you will have to have read the previous tutorials to understand much of what it going on. You can find a list of tutorials here: XNA 3.0 Role Playing Game Tutorials You will also find the latest version of the project on the web site on that page. If you want to follow along and type in the code from this PDF as you go, you can find the previous project at this link: Eyes of the Dragon - Version 20 You can download the graphics from this link: Graphics.zip

In this tutorial I will be adding a custom content pipeline processor and importer for the game to read in **Tileset** objects. I have been debating if I should go over creating the **Tile Set Generator** or if I should just go over the format of the XML document that it creates. I decide that in this tutorial I will just go over the XML document that the generator creates. I will write a separate tutorial on creating the generator.

I didn't have to create an XML document for this I could have used a different file format. Creating an XML document is fairly easy with C# and the same is true for reading one in. There is one thing that is important. I had to give the document a unique file extension. I settled on **tset**. It seemed fairly unique and described the file. This is what a **tset** file looks like.

```
<Tileset>
  <TextureElement TextureName="tileset1" />
  <TilesetDefinitions TileWidth="128" TileHeight="128" />
  <TilesetRectangles>
    <Rectangle X="0" Y="0" Width="128" Height="128" />
    <Rectangle X="128" Y="0" Width="128" Height="128" />
    <Rectangle X="256" Y="0" Width="128" Height="128" />
    <Rectangle X="384" Y="0" Width="128" Height="128" />
    <Rectangle X="0" Y="128" Width="128" Height="128" />
    <Rectangle X="128" Y="128" Width="128" Height="128" />
    <Rectangle X="256" Y="128" Width="128" Height="128" />
    <Rectangle X="384" Y="128" Width="128" Height="128" />
    <Rectangle X="0" Y="256" Width="128" Height="128" />
    <Rectangle X="128" Y="256" Width="128" Height="128" />
    <Rectangle X="256" Y="256" Width="128" Height="128" />
    <Rectangle X="384" Y="256" Width="128" Height="128" />
    <Rectangle X="0" Y="384" Width="128" Height="128" />
    <Rectangle X="128" Y="384" Width="128" Height="128" />
    <Rectangle X="256" Y="384" Width="128" Height="128" />
    <Rectangle X="384" Y="384" Width="128" Height="128" />
  </TilesetRectangles>
</Tileset>
```

The root node for the **tset** file is called **Tileset**. The first child node is **TextureElement**. This node has an attribute called **TextureName**. This attribute holds the name of the **Texture2D** for the tile set. The second child node is **TilesetDefinitions**. This node has two attributes. **TileWidth** and **TileHeight**. These are in order the width of each tile in the tile set and the height of the tile in the tile set. The last child node is **TilesetRectangles**. This node has many child nodes called **Rectangle**. These children hold information about rectangles in the tile set rectangles. Like in the **Tileset** class, the rectangles are generated by starting in the upper left hand corner and going across and then down. I noticed that in the constructor for the **Tileset** class I was doing it the other way around. Starting at the upper left hand corner and going down then across. The **Rectangle** nodes have four attributes **X, Y, Width** and **Height** . The **X** attribute holds the **X** value of each rectangle and **Y** holds the **Y** value of

each rectangle. The **Width** and **Height** attributes hold the width and height of the rectangles.

Before I go much farther there are few things that you should do. First go to the **Tileset** folder in the **Content** folder and rename **tileset1.png** to **tilesetTexture1.png**. I have added a new file to my web site Content.zip. This file will hold the non-graphic conent for the game. Download the file and extract the **tileset1.tset** file and add it to the **Tileset** folder in the **Content** folder.

To create custom content processers and importers you have to create a new project for the processers and importers. The first thing you will need to do is right click **solution name**, not the project, and select **Add** and then **New project**. From the dialog box that pops up you need to navigate to the **XNA Game Studio** node. Next you need to select **Content Pipeline Extension Library** and call it **New2DRPGContent**. There will be a class in the new project called **ContentProcessor1.cs**. You do not need this file so right click it and select **Delete**. Now a reference to this project must be added to the **Content** folder of the game. Right click the **Content** folder and select **Add reference**. Click the **Projects** tab of the dialog box that pops up and select **New2DRPGContent**.

To the **New2DRPGContent** project add a new folder called **Tileset**. This folder will hold the processor, importer and writer for the **Tileset** class. This folder will hold the **Content Importer, Conent Processor** and **Content Writer** for the **Tileset** class.

First I will add the **Content Importer.** The job of the importer is to read in the **Content** from the **Content** folder. Right click the new folder and select **Add new item**. In the dialog that pops up navigate to the **XNA Game Studio** node and select **Content Importer** call it **TilesetImporter**. As I always do when presenting new code I will give you the code and then explain it. This is the code for the **TilsetImporter**.

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Content.Pipeline;
using Microsoft.Xna.Framework.Content.Pipeline.Graphics;
using System.Xml;

using TImport = System.Xml.XmlDocument;

namespace New2DRPGContent
{
    [ContentImporter(".tset", DisplayName = "Tileset Importer",
            DefaultProcessor = "TilesetProcessor")]
    public class ContentImporter1 : ContentImporter<TImport>
    {
        public override TImport Import(string filename,
            ContentImporterContext context)
        {
            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.Load(filename);
            return xmlDoc;
        }
    }
}
```

When you create a new importer there is a using statement that says **TImport = System.String**.

TImport is the return type of the importer. You don't have to use this but I decide to use it. Our importer reads in an XML document so I changed **System.String** to **System.Xml.XmlDocument**. I also added in a using statement of **System.Xml**.

The class uses the attribute **ContentImporterAttribute**. This attribute is applied to the class. It associates files with a **.tset** file extension with the importer. The **DisplayName** parameter give the processor a friendly name. The **DefaultProcessor** parameter says that when a **tset** file is found to use the **TilsetProcessor**.

The constructor takes as a parameter the **filename** of the file to be imported. The second parameter provides properties that define logging behavior for the importer. Inside the constuctor I create an instance of **XmlDocument**. I call the **Load** method of the document passing the **filename** parameter of the constructor. Finally I return the **XmlDocument**. That is all for the importer.

Before I get to the **Content Processor** I want to create a small class to hold the values the **Content Processor** creates. Right click the **Tileset** folder in the **New2DRPGContent** folder and select **Add class**. Name the class **TilesetContent**. This class is a public class that holds the name of the texture for the tileset, the tile height and width, how many tiles high and wide the tileset is and the list of rectangles for the tileset. I did have to add a using statement for the **XNA Framework**. This is the code for the class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;

namespace New2DRPGContent.Tileset
{
    public class TilesetContent
    {
        public string textureName;
        public int tileWidth, tileHeight;
        public List<Rectangle> tileRectangles;
    }
}
```

Now it is time to write the **Content Processor** . The job of the processor is to process the return type of the importer. Right click the **Tileset** folder in the **New2DRPGContent** folder. Select the **XNA Game Studio** node and then **Content Processor**. Name it **TilesetProcessor**. You will have to give the processor the same name that you used in the importer. This is the code for the processor.

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Content.Pipeline;
using Microsoft.Xna.Framework.Content.Pipeline.Graphics;
using Microsoft.Xna.Framework.Content.Pipeline.Processors;
using System.Xml;
using TInput = System.Xml.XmlDocument;
using TOutput = New2DRPGContent.Tileset.TilesetContent;
```

```
namespace New2DRPGContent.Tileset
{
    [ContentProcessor(DisplayName = "Tileset Processor")]
    public class TilesetProcessor : ContentProcessor<TInput, TOutput>
    {
        public override TOutput Process(TInput input,
            ContentProcessorContext context)
        {
            TilesetContent tilesetContent = new TilesetContent();

            foreach (XmlNode node in input.DocumentElement.ChildNodes)
            {
                if (node.Name == "TextureElement")
                {
                    tilesetContent.textureName =
                        node.Attributes["TextureName"].Value;
                }
                if (node.Name == "TilesetDefinitions")
                {
                    tilesetContent.tileWidth =
                        Int32.Parse(node.Attributes["TileWidth"].Value);
                    tilesetContent.tileHeight =
                        Int32.Parse(node.Attributes["TileHeight"].Value);
                }

                if (node.Name == "TilesetRectangles")
                {
                    List<Rectangle> rectangles = new List<Rectangle>();

                    foreach (XmlNode rectNode in node.ChildNodes)
                    {
                        if (rectNode.Name == "Rectangle")
                        {
                            Rectangle rect;
                            rect = new Rectangle(
                                Int32.Parse(rectNode.Attributes["X"].Value),
                                Int32.Parse(rectNode.Attributes["Y"].Value),
                                Int32.Parse(rectNode.Attributes["Width"].Value),
                                Int32.Parse(rectNode.Attributes["Height"].Value));
                            rectangles.Add(rect);
                        }
                    }

                    tilesetContent.tileRectangles = rectangles;
                }
            }

            return tilesetContent;
        }
    }
}
```

I needed a using statement for XML. Like when you create a **Content Importer** the **Content Processor** creates a few using statements. **TInput = System.String** and **TOutput = System.String**. These are for the input to the processor and the output of the processor. I change the first one to **System.Xml.Xmldocument** because that is what the importer returns. For the second I used **New2DRPGContent.Tileset.TilesetContent** the class that I just wrote. The class has an attribute applied to it **ContentProcessor** this attribute has a parameter **DisplayName** which holds the friendly

name for the processor.

There is an override of the **Process** method. The **Processor** method processes the input, an XML document in this case, and returns the result, in this case a **TilesetContent** object. The method creates an instance of the **TilesetContent** object and then parses the XML documents. I am not sure how familiar you are with XML documents so I will try and explain it.

Inside a **foreach** loop I loop through all of the child nodes of the root element. You can get the root element of an XML document using the **DocumentElement** property. The **DocumentElement** property has a collection called **ChildNodes**. This collection has all of the children of the root element. I used a foreach loop instead of reading them in order in case you want to create your own XML file and don't put the elements in order.

There are three if statements in side the foreach loop. The first one checks to see if the current node is the **TextureElement**. If it is the **TextureElement** I get the value of the **TextureName** attribute using the Value property of the Attributes property for the node. The Attributes property is a collection of attributes for the node.

In the second if statement I check to see if the current node is **TilesetDefinitions**. I then use the **Parse** method the **Int32** to convert the sting values of the attributes to integers. I set **tileWidth** and **tileHeight** to the value of the attributes **TileWidth** and **TileHeight.**

In the third if statement checks to see if the current node is **TilesetRectangles**. Inside that if statement I create a **List<Rectangle>** to hold all of the rectangles. Then I have another foreach loop. That loop will loop through all of the children of the current node. There is an if statement inside that loop to make sure that the name of the child is **Rectangle**. If the child is a rectangle I create using the attribute of the node.

Now it is time to write the **Content Type Writer**. The job of the **Content Type Writer** is to write a binary **XNB** file. This will help protect your custom content from being hacked. For example if you have script files that are in XML format somebody can easily change the text of the XML files. That is one of the reasons why I chose to use these classes. So right click the **Tileset** folder in the **New2DRPGContent** project. Make sure to select the **XNA Game Studio** node and then choose **Content Type Writer**. Call it **TilesetTypeWriter**. This is the code for the **TilesetTypeWriter** class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Content.Pipeline;
using Microsoft.Xna.Framework.Content.Pipeline.Graphics;
using Microsoft.Xna.Framework.Content.Pipeline.Processors;
using Microsoft.Xna.Framework.Content.Pipeline.Serialization.Compiler;

using TWrite = New2DRPGContent.Tileset.TilesetContent;

namespace New2DRPGContent.Tileset
{
    [ContentTypeWriter]
    public class TilesetTypeWriter : ContentTypeWriter<TWrite>
```

```
    {
        protected override void Write(ContentWriter output, TWrite value)
        {
            output.Write(value.textureName);
            output.Write(value.tileWidth);
            output.Write(value.tileHeight);
            output.Write(value.tileRectangles.Count);
            foreach (Rectangle rect in value.tileRectangles)
            {
                output.Write(rect.X);
                output.Write(rect.Y);
                output.Write(rect.Width);
                output.Write(rect.Height);
            }
        }

        public override string GetRuntimeReader(TargetPlatform targetPlatform)
        {
            return "New2DRPG.TilesetReader, EyesOfTheDragon";
        }
    }
}
```

There is yet another using statement in this class. This one is called **TWrite**. It is the type that the **Content Processor** returned. There is an attribute applied to this class. You must apply the attribute as well as extend the **ContentTypeWriter** class. There are two methods in this class. The first one is an override of the **Write** method. This is the method you will use to write out the **XNB** file. It takes as a parameter a **ContentWriter** object that is used to write the **XNB** file and a parameter **TWrite**. This parameter is the values from the **Content Processor**. The second method is called **GetRuntimeReader**. This method has a parameter an enum of type **TargetPlatform**. This paremeter holds the platform the game is targeting.

In the **Write** method I write out the values from the **value** parameter. It is important to note the order that you write things out because in the **Content Type Reader** you will need to read them in the same order. I first write out the name of the texture for the **Tileset**. Then I write out the width and height of the tiles. To help the **Content Reader** I decided to write out the number of rectangles there are in the list of rectangles. Finally in a foreach loop I write out the **X**, **Y**, **Width** and **Height** of each of the rectangles.

The **GetRuntimeReader** method returns a string that describes the reader for the class. The first part is the name of the **Content Type Reader** and the second is the name of the assembly the reader is in. In this case **New2DRPG.TilesetReader** is the name of the **Content Type Reader** and the name of the assembly is **EyesOfTheDragon**.

Before I get to the **Content Type Reader** I want to make a few changes to the game. The first change will be the **Tileset** class. I am going to change this class to have the name of the texture for the tile set, the width and height of the tiles and a List<Rectangle> for the tiles. This is the new class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
```

```
namespace New2DRPG
{
    public class Tileset
    {
        List<Rectangle> tiles = new List<Rectangle>();
        static int tileWidth;
        static int tileHeight;
        string textureName;

        public Tileset(string textureName, int tileWidth, int tileHeight,
            List<Rectangle> tiles)
        {
            this.textureName = textureName;
            Tileset.tileWidth = tileWidth;
            Tileset.tileHeight = tileHeight;
            this.tiles = tiles;
        }

        public List<Rectangle> Tiles
        {
            get { return tiles; }
        }

        public static int TileWidth
        {
            get { return tileWidth; }
        }

        public static int TileHeight
        {
            get { return tileHeight; }
        }

        public string TextureName
        {
            get { return textureName; }
        }
    }
}
```

You will see that there have been a lot of changes to this class. One is that I removed the static **Texture2D** field **tilesetTexture** and the two integer fields **tilesWide** and **tilesHigh**. I added in a new **string** field **textureName**. This field will hold the asset name for the associated tile set texture. There are public get only properties for all of the fields. I no longer need the **CreateRectangle** method as well. The parameters to the constructor are a **string textureName**, an int **tileWidth**, an int **tileHeight** and a **List<Rectangle>** tiles. In the constructor I assign the parameters to the corrosponding fields.

There were changes to the construtor and **LoadContent** methods of the **ActionScreen**. In the constructor I no longer need to create the **Tileset** object as I can load it in using the **ContentManager** object. In the **LoadContent** method I load in the **Tileset** object. These are the new methods.

```
public ActionScreen(Game game, SpriteFont gameFont, string tilesetName)
    : base(game)
{
    playerCharacter = new PlayerCharacter(game);
    this.gameFont = gameFont;
```

```
    this.tilesetName = tilesetName;
    LoadContent();

    tileEngine = new TileEngine(game, this.tileset, 50, 50);
    Components.Add(tileEngine);
    tileEngine.Show();

    viewportWidth = TileEngine.ViewPortWidth;
    viewportHeight = TileEngine.ViewPortHeight;
    screenWidth = game.Window.ClientBounds.Width;
    screenHeight = game.Window.ClientBounds.Height;
}

protected override void LoadContent()
{
    base.LoadContent();
    tileset = Content.Load<Tileset>(@"TileSets\tileset1");
    chest = Content.Load<Texture2D>(@"Items\chest");
    characterHUDTexture = Content.Load<Texture2D>(@"Backgrounds\characterhud");
    this.interfaceFont = Content.Load<SpriteFont>("smallFont");
}
```

I also had to change the **TileEngine**. The first thing that I did was add in a static **Texture2D** field **tilesetTexture**. Then I changed the **TileSet** property to return the new static field. These are the changes.

```
static Texture2D tilesetTexture;

public TileEngine(Game game, Tileset tileset,
                  int tileMapWidth, int tileMapHeight)
    : base(game)
{
    spriteBatch =
        (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
    Content =
        (ContentManager)Game.Services.GetService(typeof(ContentManager));

    TileEngine.tilesetTexture =
        Content.Load<Texture2D>(@"TileSets\" + tileset.TextureName);

    TileEngine.tileset = tileset;
    TileEngine.tileMapWidth = tileMapWidth;
    TileEngine.tileMapHeight = tileMapHeight;
    TileEngine.tiles = tileset.Tiles;

    map = new TileMap(tileMapWidth, tileMapHeight, game);
    childComponents.Add(map);

    Texture2D chest = Content.Load<Texture2D>(@"Items\chest");

    ItemSprite tempItemSprite;
    Vector2 position;
    Random random = new Random();

    for (int i = 0; i < 3; i++)
    {
        position = new Vector2(
            random.Next(0, 10),
            random.Next(0, 10));
```

```
            tempItemSprite = new ItemSprite(game,
                chest, position);

            childComponents.Add(tempItemSprite);
        }

        screenWidth = game.Window.ClientBounds.Width;
        screenHeight = game.Window.ClientBounds.Height;
    }

    public static Texture2D TileSet
    {
        get { return TileEngine.tilesetTexture; }
    }
```

Now it is time to write the **Content Type Reader**. Go back to the game project and right click the **TileEngine** folder. To this folder add in a new class called **TilesetReader**. This is the code for that class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using System.Reflection;

namespace New2DRPG
{
    class TilesetReader : ContentTypeReader<Tileset>
    {
        protected override Tileset Read(
            ContentReader input, Tileset existingInstance)
        {
            string textureName = input.ReadString();
            int tileWidth = input.ReadInt32();
            int tileHeight = input.ReadInt32();
            int tileCount = input.ReadInt32();
            List<Rectangle> tiles = new List<Rectangle>();

            for (int i = 0; i < tileCount; i++)
            {
                int xValue = input.ReadInt32();
                int yValue = input.ReadInt32();
                int width = input.ReadInt32();
                int height = input.ReadInt32();
                Rectangle rect = new Rectangle(xValue,
                    yValue,
                    width,
                    height);

                tiles.Add(rect);
            }

            return new Tileset(textureName, tileWidth, tileHeight, tiles);
        }
```

```
        }
}
```

There is just one method in this class **Read**. This method is responsible for reading in your custom content and returning a new object. The parameters to the **Read** method are a **ContentReader input** that will be used to read the content. The second parameter is a little more complicated. The documentation on the **ContentTypeReader** class says this: "**The object receiving the data, or null if a new instance of the object should be created."** I just return a new instance instead of using this parameter.

In the **Read** method I just read in each of the items in order. The **ContentReader** class has methods for reading different types. I use the **ReadString** method to read in the **textureName**. I use **ReadInt32** to read in t**ileWidth**, **tileHeight** and **tileCount**. Next I create a **List<Rectangle>** to hold the rectangles. Then inside a for loop I use **ReadInt32** to read in **xValue**, **yValue**, **width** and **height**. I create a new **Rectangle** object using those variables and add it to the **List<Rectangle>**. Then I return a new **Tileset** object.

Well that is it for this tutorial. I am already working on coding the next part of Eyes of the Dragon so I encourage you to keep either visiting my site http://xna.jtmbooks.com or my blog, http://xna-rpg.blogspot.com for the latest news on these tutorials.