Eyes of the Dragon Tutorials
Part 23
Level Editor

I'm writing these tutorials for the MonoGame 3.8 framework using Visual Studio 2019. The tutorials will make more sense if they are read in order. You can find the list of tutorials on the Eyes of the Dragon page of my web blog. I will be making each version of the project available on GitHub here. It will be included on the page that links to the tutorials.

You all knew that this was coming at some point. You will want to be able to create your world at design time and read it in at run time. For that you will need a world, or level, editor. To accomplish that I'm going to add a new project that will be used to create your levels and other content that is MonoGame related. The **RpgEditor** is great for creating data to be used with the game but you are going to want to associated MonoGame related content to NPCs as an example. You will want a sprite to represent the NPC in the game.

To get started right click the **EyesOfTheDragon** solution, not the project, in the solution explorer. Select **Add** and then **New Project**. Choose a **Windows Forms App (.NET Framework)** from the list of projects. Name this new project **XLevelEditor**.

Now, right click the **Form1.cs** file and select Rename. Name the form **FormMain.cs**. When prompted if you wish to rename code elements select **Yes**.

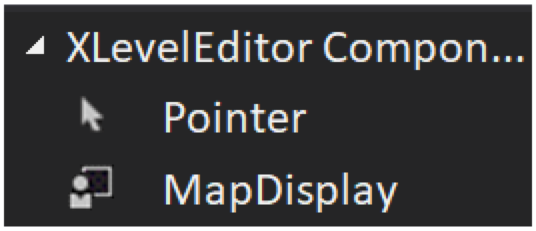
Right click the **XLevelEditor** project in the **Solution Explorer** and choose **Manage NuGet Packages**. Click the **Browse** tab on the left and enter **MonoGame.Forms.DX** in the search box. On the right click the **Install** button to install the package. Repeat the process but this time you will want to search for **MonoGame.Framework.Content.Pipeline** to give us access to **IntermediateSerializer** for serializing our content. The last step is to reference the **RpgLibrary** in the new project. Right click the **RpgLibrary** in the **Solution Explorer** and select **Add** and then **Reference**. Select **Project Reference** from the list on the left then **RpgLibrary**.

The next step is to create a class for drawing the map in. Right click the **XRpgEditor** project, select **Add** and then **Class**. Name this new class **MapDisplay**. This is the code for that class so far.

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

namespace XLevelEditor
{
    public class MapDisplay : MonoGame.Forms.Controls.MonoGameControl
    {
        protected override void Initialize()
        {
            base.Initialize();
        }

        protected override void Update(GameTime gameTime)
        {
            base.Update(gameTime);
        }

        protected override void Draw()
        {
            base.Draw();
        }
    }
}
```

The first step is that I inherit the class from **MonoGame.Forms.Controls.MonoGameControl**. That is from the **MonGame.Forms.DX NuGet** package and can be used to render MonoGame. This is an abstract class and defines **Initialize**, **Update** and **Draw** methods. In the all three overrides I call the base method associated with them.

Build your project now. After building it bring up the design view of **FormMain** by right clicking it and selecting **View Designer**. If you expand the Toolbox you should see something similar to the following at the top of the Toolbox.



The next step is going to be to design some forms. The main form will be the actual editor. The child forms will be for adding items to a level. I found copy/paste much more effective and efficient when designing forms that doing it manually through text. Expand the **FormMain.cs** node in the project and open the **FormMain.Designer.cs** file. Place the following code in that file. Because it is generate code I am not going to explain it. It's just easier than trying to go over dropping over 40 controls on the form and adjusting their properties.

```
namespace XLevelEditor
{
    partial class FormMain
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.menuStrip1 = new System.Windows.Forms.MenuStrip();
            this.levelToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.newLevelToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.openLevelToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.saveLevelToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.toolStripMenuItem1 = new System.Windows.Forms.ToolStripSeparator();
            this.exitEditorToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.tilesetToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.newTilesetToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.openTilesetToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.loadTilesetToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.removeTilesetToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.mapLayerToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.newLayerToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.openLayerToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.saveLayerToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.charactersToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.chestsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.keysToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.splitContainer1 = new System.Windows.Forms.SplitContainer();
            this.mapDisplay = new XLevelEditor.MapDisplay();
            this.tabProperties = new System.Windows.Forms.TabControl();
            this.tabTilesets = new System.Windows.Forms.TabPage();
            this.lblTilesets = new System.Windows.Forms.Label();
            this.lbTileset = new System.Windows.Forms.ListBox();
            this.pbTilesetPreview = new System.Windows.Forms.PictureBox();
            this.lblCurrentTileset = new System.Windows.Forms.Label();
            this.nudCurrentTile = new System.Windows.Forms.NumericUpDown();
            this.gbDrawMode = new System.Windows.Forms.GroupBox();
            this.rbErase = new System.Windows.Forms.RadioButton();
            this.rbDraw = new System.Windows.Forms.RadioButton();
            this.lblTile = new System.Windows.Forms.Label();
            this.pbTilePreview = new System.Windows.Forms.PictureBox();
            this.tabLayers = new System.Windows.Forms.TabPage();
            this.clbLayers = new System.Windows.Forms.CheckedListBox();
            this.tabCharacters = new System.Windows.Forms.TabPage();
            this.tabChests = new System.Windows.Forms.TabPage();
            this.tabKeys = new System.Windows.Forms.TabPage();
            this.controlTimer = new System.Windows.Forms.Timer(this.components);
            this.menuStrip1.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.splitContainer1)).BeginInit();
            this.splitContainer1.Panel1.SuspendLayout();
            this.splitContainer1.Panel2.SuspendLayout();
            this.splitContainer1.SuspendLayout();
            this.tabProperties.SuspendLayout();
            this.tabTilesets.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.pbTilesetPreview)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.nudCurrentTile)).BeginInit();
            this.gbDrawMode.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.pbTilePreview)).BeginInit();
            this.tabLayers.SuspendLayout();
            this.SuspendLayout();
            //
            // menuStrip1
            //
            this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.levelToolStripMenuItem,
            this.tilesetToolStripMenuItem,
            this.mapLayerToolStripMenuItem,
            this.charactersToolStripMenuItem,
            this.chestsToolStripMenuItem,
            this.keysToolStripMenuItem});
            this.menuStrip1.Location = new System.Drawing.Point(0, 0);
            this.menuStrip1.Name = "menuStrip1";
            this.menuStrip1.Size = new System.Drawing.Size(1006, 28);
            this.menuStrip1.TabIndex = 0;
            this.menuStrip1.Text = "menuStrip1";
            //
            // levelToolStripMenuItem
            //
            this.levelToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.newLevelToolStripMenuItem,
            this.openLevelToolStripMenuItem,
            this.saveLevelToolStripMenuItem,
            this.toolStripMenuItem1,
            this.exitEditorToolStripMenuItem});
            this.levelToolStripMenuItem.Name = "levelToolStripMenuItem";
            this.levelToolStripMenuItem.Size = new System.Drawing.Size(55, 24);
            this.levelToolStripMenuItem.Text = "&Level";
            //
            // newLevelToolStripMenuItem
            //
            this.newLevelToolStripMenuItem.Name = "newLevelToolStripMenuItem";
            this.newLevelToolStripMenuItem.Size = new System.Drawing.Size(152, 24);
            this.newLevelToolStripMenuItem.Text = "&New Level";
            //
            // openLevelToolStripMenuItem
            //
            this.openLevelToolStripMenuItem.Name = "openLevelToolStripMenuItem";
            this.openLevelToolStripMenuItem.Size = new System.Drawing.Size(152, 24);
            this.openLevelToolStripMenuItem.Text = "&Open Level";
            //
            // saveLevelToolStripMenuItem
            //
            this.saveLevelToolStripMenuItem.Name = "saveLevelToolStripMenuItem";
            this.saveLevelToolStripMenuItem.Size = new System.Drawing.Size(152, 24);
            this.saveLevelToolStripMenuItem.Text = "&Save Level";
            //
            // toolStripMenuItem1
            //
            this.toolStripMenuItem1.Name = "toolStripMenuItem1";
            this.toolStripMenuItem1.Size = new System.Drawing.Size(149, 6);
            //
            // exitEditorToolStripMenuItem
            //
            this.exitEditorToolStripMenuItem.Name = "exitEditorToolStripMenuItem";
            this.exitEditorToolStripMenuItem.Size = new System.Drawing.Size(152, 24);
            this.exitEditorToolStripMenuItem.Text = "E&xit Editor";
            //
            // tilesetToolStripMenuItem
            //
            this.tilesetToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.newTilesetToolStripMenuItem,
            this.openTilesetToolStripMenuItem,
            this.loadTilesetToolStripMenuItem,
            this.removeTilesetToolStripMenuItem});
            this.tilesetToolStripMenuItem.Name = "tilesetToolStripMenuItem";
            this.tilesetToolStripMenuItem.Size = new System.Drawing.Size(64, 24);
            this.tilesetToolStripMenuItem.Text = "&Tileset";
            //
            // newTilesetToolStripMenuItem
            //
            this.newTilesetToolStripMenuItem.Name = "newTilesetToolStripMenuItem";
            this.newTilesetToolStripMenuItem.Size = new System.Drawing.Size(179, 24);
            this.newTilesetToolStripMenuItem.Text = "&New Tileset";
            //
            // openTilesetToolStripMenuItem
            //
            this.openTilesetToolStripMenuItem.Name = "openTilesetToolStripMenuItem";
            this.openTilesetToolStripMenuItem.Size = new System.Drawing.Size(179, 24);
            this.openTilesetToolStripMenuItem.Text = "&Open Tileset";
            //
            // loadTilesetToolStripMenuItem
            //
            this.loadTilesetToolStripMenuItem.Name = "loadTilesetToolStripMenuItem";
            this.loadTilesetToolStripMenuItem.Size = new System.Drawing.Size(179, 24);
            this.loadTilesetToolStripMenuItem.Text = "&Load Tileset";
            //
            // removeTilesetToolStripMenuItem
            //
            this.removeTilesetToolStripMenuItem.Name = "removeTilesetToolStripMenuItem";
            this.removeTilesetToolStripMenuItem.Size = new System.Drawing.Size(179, 24);
            this.removeTilesetToolStripMenuItem.Text = "&Remove Tileset";
            //
            // mapLayerToolStripMenuItem
            //
            this.mapLayerToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.newLayerToolStripMenuItem,
            this.openLayerToolStripMenuItem,
            this.saveLayerToolStripMenuItem});
            this.mapLayerToolStripMenuItem.Name = "mapLayerToolStripMenuItem";
            this.mapLayerToolStripMenuItem.Size = new System.Drawing.Size(90, 24);
            this.mapLayerToolStripMenuItem.Text = "&Map Layer";
            //
            // newLayerToolStripMenuItem
            //
            this.newLayerToolStripMenuItem.Name = "newLayerToolStripMenuItem";
            this.newLayerToolStripMenuItem.Size = new System.Drawing.Size(153, 24);
            this.newLayerToolStripMenuItem.Text = "&New Layer";
            //
            // openLayerToolStripMenuItem
```

```
//
this.openLayerToolStripMenuItem.Name = "openLayerToolStripMenuItem";
this.openLayerToolStripMenuItem.Size = new System.Drawing.Size(153, 24);
this.openLayerToolStripMenuItem.Text = "&Open Layer";
//
// saveLayerToolStripMenuItem
//
this.saveLayerToolStripMenuItem.Name = "saveLayerToolStripMenuItem";
this.saveLayerToolStripMenuItem.Size = new System.Drawing.Size(153, 24);
this.saveLayerToolStripMenuItem.Text = "&Save Layer";
//
// charactersToolStripMenuItem
//
this.charactersToolStripMenuItem.Name = "charactersToolStripMenuItem";
this.charactersToolStripMenuItem.Size = new System.Drawing.Size(90, 24);
this.charactersToolStripMenuItem.Text = "&Characters";
//
// chestsToolStripMenuItem
//
this.chestsToolStripMenuItem.Name = "chestsToolStripMenuItem";
this.chestsToolStripMenuItem.Size = new System.Drawing.Size(63, 24);
this.chestsToolStripMenuItem.Text = "C&hests";
//
// keysToolStripMenuItem
//
this.keysToolStripMenuItem.Name = "keysToolStripMenuItem";
this.keysToolStripMenuItem.Size = new System.Drawing.Size(51, 24);
this.keysToolStripMenuItem.Text = "&Keys";
//
// splitContainer1
//
this.splitContainer1.Dock = System.Windows.Forms.DockStyle.Fill;
this.splitContainer1.Location = new System.Drawing.Point(0, 28);
this.splitContainer1.Name = "splitContainer1";
//
// splitContainer1.Panel1
//
this.splitContainer1.Panel1.Controls.Add(this.mapDisplay);
//
// splitContainer1.Panel2
//
this.splitContainer1.Panel2.Controls.Add(this.tabProperties);
this.splitContainer1.Size = new System.Drawing.Size(1006, 647);
this.splitContainer1.SplitterDistance = 800;
this.splitContainer1.TabIndex = 1;
//
// mapDisplay
//
this.mapDisplay.Dock = System.Windows.Forms.DockStyle.Fill;
this.mapDisplay.Location = new System.Drawing.Point(0, 0);
this.mapDisplay.Name = "mapDisplay";
this.mapDisplay.Size = new System.Drawing.Size(800, 647);
this.mapDisplay.TabIndex = 0;
this.mapDisplay.Text = "mapDisplay1";
//
// tabProperties
//
this.tabProperties.Controls.Add(this.tabTilesets);
this.tabProperties.Controls.Add(this.tabLayers);
this.tabProperties.Controls.Add(this.tabCharacters);
```

```
            this.tabProperties.Controls.Add(this.tabChests);
            this.tabProperties.Controls.Add(this.tabKeys);
            this.tabProperties.Dock = System.Windows.Forms.DockStyle.Fill;
            this.tabProperties.Location = new System.Drawing.Point(0, 0);
            this.tabProperties.Name = "tabProperties";
            this.tabProperties.SelectedIndex = 0;
            this.tabProperties.Size = new System.Drawing.Size(202, 647);
            this.tabProperties.TabIndex = 1;
            //
            // tabTilesets
            //
            this.tabTilesets.Controls.Add(this.lblTilesets);
            this.tabTilesets.Controls.Add(this.lbTileset);
            this.tabTilesets.Controls.Add(this.pbTilesetPreview);
            this.tabTilesets.Controls.Add(this.lblCurrentTileset);
            this.tabTilesets.Controls.Add(this.nudCurrentTile);
            this.tabTilesets.Controls.Add(this.gbDrawMode);
            this.tabTilesets.Controls.Add(this.lblTile);
            this.tabTilesets.Controls.Add(this.pbTilePreview);
            this.tabTilesets.Location = new System.Drawing.Point(4, 25);
            this.tabTilesets.Name = "tabTilesets";
            this.tabTilesets.Padding = new System.Windows.Forms.Padding(3);
            this.tabTilesets.Size = new System.Drawing.Size(194, 618);
            this.tabTilesets.TabIndex = 0;
            this.tabTilesets.Text = "Tiles";
            this.tabTilesets.UseVisualStyleBackColor = true;
            //
            // lblTilesets
            //
            this.lblTilesets.Location = new System.Drawing.Point(7, 321);
            this.lblTilesets.Name = "lblTilesets";
            this.lblTilesets.Size = new System.Drawing.Size(180, 23);
            this.lblTilesets.TabIndex = 7;
            this.lblTilesets.Text = "Tilesets";
            this.lblTilesets.TextAlign = System.Drawing.ContentAlignment.TopCenter;
            //
            // lbTileset
            //
            this.lbTileset.FormattingEnabled = true;
            this.lbTileset.ItemHeight = 16;
            this.lbTileset.Location = new System.Drawing.Point(7, 352);
            this.lbTileset.Name = "lbTileset";
            this.lbTileset.Size = new System.Drawing.Size(180, 260);
            this.lbTileset.TabIndex = 6;
            //
            // pbTilesetPreview
            //
            this.pbTilesetPreview.Location = new System.Drawing.Point(7, 138);
            this.pbTilesetPreview.Name = "pbTilesetPreview";
            this.pbTilesetPreview.Size = new System.Drawing.Size(180, 180);
            this.pbTilesetPreview.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.StretchImage;
            this.pbTilesetPreview.TabIndex = 5;
            this.pbTilesetPreview.TabStop = false;
            //
            // lblCurrentTileset
            //
            this.lblCurrentTileset.Location = new System.Drawing.Point(7, 112);
            this.lblCurrentTileset.Name = "lblCurrentTileset";
            this.lblCurrentTileset.Size = new System.Drawing.Size(180, 23);
            this.lblCurrentTileset.TabIndex = 4;
            this.lblCurrentTileset.Text = "Current Tileset";
            this.lblCurrentTileset.TextAlign = System.Drawing.ContentAlignment.TopCenter;
            //
            // nudCurrentTile
            //
            this.nudCurrentTile.Location = new System.Drawing.Point(7, 83);
            this.nudCurrentTile.Name = "nudCurrentTile";
            this.nudCurrentTile.Size = new System.Drawing.Size(180, 22);
            this.nudCurrentTile.TabIndex = 3;
            //
            // gbDrawMode
            //
            this.gbDrawMode.Controls.Add(this.rbErase);
            this.gbDrawMode.Controls.Add(this.rbDraw);
            this.gbDrawMode.Location = new System.Drawing.Point(63, 7);
            this.gbDrawMode.Name = "gbDrawMode";
            this.gbDrawMode.Size = new System.Drawing.Size(128, 70);
            this.gbDrawMode.TabIndex = 2;
            this.gbDrawMode.TabStop = false;
            this.gbDrawMode.Text = "Draw Mode";
            //
            // rbErase
            //
            this.rbErase.AutoSize = true;
            this.rbErase.Location = new System.Drawing.Point(7, 43);
            this.rbErase.Name = "rbErase";
            this.rbErase.Size = new System.Drawing.Size(66, 21);
            this.rbErase.TabIndex = 1;
            this.rbErase.Text = "Erase";
            this.rbErase.UseVisualStyleBackColor = true;
            //
            // rbDraw
            //
            this.rbDraw.AutoSize = true;
            this.rbDraw.Checked = true;
            this.rbDraw.Location = new System.Drawing.Point(7, 20);
            this.rbDraw.Name = "rbDraw";
            this.rbDraw.Size = new System.Drawing.Size(61, 21);
            this.rbDraw.TabIndex = 0;
            this.rbDraw.TabStop = true;
            this.rbDraw.Text = "Draw";
            this.rbDraw.UseVisualStyleBackColor = true;
            //
            // lblTile
            //
            this.lblTile.Location = new System.Drawing.Point(7, 7);
            this.lblTile.Name = "lblTile";
            this.lblTile.Size = new System.Drawing.Size(50, 17);
            this.lblTile.TabIndex = 1;
            this.lblTile.Text = "Tile";
            this.lblTile.TextAlign = System.Drawing.ContentAlignment.TopCenter;
            //
            // pbTilePreview
            //
            this.pbTilePreview.Location = new System.Drawing.Point(7, 27);
            this.pbTilePreview.Name = "pbTilePreview";
            this.pbTilePreview.Size = new System.Drawing.Size(50, 50);
            this.pbTilePreview.TabIndex = 0;
            this.pbTilePreview.TabStop = false;
```

```
//
// tabLayers
//
this.tabLayers.Controls.Add(this.clbLayers);
this.tabLayers.Location = new System.Drawing.Point(4, 25);
this.tabLayers.Name = "tabLayers";
this.tabLayers.Padding = new System.Windows.Forms.Padding(3);
this.tabLayers.Size = new System.Drawing.Size(194, 618);
this.tabLayers.TabIndex = 1;
this.tabLayers.Text = "Map Layers";
this.tabLayers.UseVisualStyleBackColor = true;
//
// clbLayers
//
this.clbLayers.Dock = System.Windows.Forms.DockStyle.Fill;
this.clbLayers.FormattingEnabled = true;
this.clbLayers.Location = new System.Drawing.Point(3, 3);
this.clbLayers.Name = "clbLayers";
this.clbLayers.Size = new System.Drawing.Size(188, 612);
this.clbLayers.TabIndex = 0;
//
// tabCharacters
//
this.tabCharacters.Location = new System.Drawing.Point(4, 25);
this.tabCharacters.Name = "tabCharacters";
this.tabCharacters.Size = new System.Drawing.Size(194, 618);
this.tabCharacters.TabIndex = 2;
this.tabCharacters.Text = "Characters";
this.tabCharacters.UseVisualStyleBackColor = true;
//
// tabChests
//
this.tabChests.Location = new System.Drawing.Point(4, 25);
this.tabChests.Name = "tabChests";
this.tabChests.Size = new System.Drawing.Size(194, 618);
this.tabChests.TabIndex = 3;
this.tabChests.Text = "Chests";
this.tabChests.UseVisualStyleBackColor = true;
//
// tabKeys
//
this.tabKeys.Location = new System.Drawing.Point(4, 25);
this.tabKeys.Name = "tabKeys";
this.tabKeys.Size = new System.Drawing.Size(194, 618);
this.tabKeys.TabIndex = 4;
this.tabKeys.Text = "Keys";
this.tabKeys.UseVisualStyleBackColor = true;
//
// FormMain
//
this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(1006, 675);
this.Controls.Add(this.splitContainer1);
this.Controls.Add(this.menuStrip1);
this.MainMenuStrip = this.menuStrip1;
this.Name = "FormMain";
this.StartPosition = System.Windows.Forms.FormStartPosition.CenterParent;
this.Text = "Level Editor";
this.menuStrip1.ResumeLayout(false);
```

```
            this.menuStrip1.PerformLayout();
            this.splitContainer1.Panel1.ResumeLayout(false);
            this.splitContainer1.Panel2.ResumeLayout(false);
            ((System.ComponentModel.ISupportInitialize)(this.splitContainer1)).EndInit();
            this.splitContainer1.ResumeLayout(false);
            this.tabProperties.ResumeLayout(false);
            this.tabTilesets.ResumeLayout(false);
            ((System.ComponentModel.ISupportInitialize)(this.pbTilesetPreview)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.nudCurrentTile)).EndInit();
            this.gbDrawMode.ResumeLayout(false);
            this.gbDrawMode.PerformLayout();
            ((System.ComponentModel.ISupportInitialize)(this.pbTilePreview)).EndInit();
            this.tabLayers.ResumeLayout(false);
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.MenuStrip menuStrip1;
        private System.Windows.Forms.ToolStripMenuItem levelToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem newLevelToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem openLevelToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem saveLevelToolStripMenuItem;
        private System.Windows.Forms.ToolStripSeparator toolStripMenuItem1;
        private System.Windows.Forms.ToolStripMenuItem exitEditorToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem tilesetToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem newTilesetToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem openTilesetToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem loadTilesetToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem mapLayerToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem newLayerToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem openLayerToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem saveLayerToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem charactersToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem chestsToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem keysToolStripMenuItem;
        private System.Windows.Forms.SplitContainer splitContainer1;
        private MapDisplay mapDisplay;
        private System.Windows.Forms.TabControl tabProperties;
        private System.Windows.Forms.TabPage tabTilesets;
        private System.Windows.Forms.TabPage tabLayers;
        private System.Windows.Forms.TabPage tabCharacters;
        private System.Windows.Forms.TabPage tabChests;
        private System.Windows.Forms.TabPage tabKeys;
        private System.Windows.Forms.Label lblTilesets;
        private System.Windows.Forms.ListBox lbTileset;
        private System.Windows.Forms.PictureBox pbTilesetPreview;
        private System.Windows.Forms.Label lblCurrentTileset;
        private System.Windows.Forms.NumericUpDown nudCurrentTile;
        private System.Windows.Forms.GroupBox gbDrawMode;
        private System.Windows.Forms.RadioButton rbErase;
        private System.Windows.Forms.RadioButton rbDraw;
        private System.Windows.Forms.Label lblTile;
        private System.Windows.Forms.PictureBox pbTilePreview;
        private System.Windows.Forms.ToolStripMenuItem removeTilesetToolStripMenuItem;
        private System.Windows.Forms.CheckedListBox clbLayers;
        private System.Windows.Forms.Timer controlTimer;
    }
```

```
}
```

Rather than reading and writing the classes in the **XRpgLibrary** directly, I'm going to add some data classes to the **RpgLibrary**. You read and write the data classes from the editor. When the player unlocks an area of your world you read in the data using the **Content Pipeline**. Since they are changing the world as they explore it you will be using a different mechanism for saving and loading games.

To get started, right click the **RpgLibrary** and select **New Folder**. Name the new folder **WorldClasses**. Right click the **WorldClasses** folder, select **Add** and then **Class**. Name the new class **TilesetData**. The code for that class follows next.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RpgLibrary.WorldClasses
{
    public class TilesetData
    {
        public string TilesetName;
        public string TilesetImageName;
        public int TileWidthInPixels;
        public int TileHeightInPixels;
        public int TilesWide;
        public int TilesHigh;
    }
}
```

Just a basic public class that has enough data to create a **Tileset** from the **XRpgLibrary**. There are two string fields, **TilesetName** and **TilesetImageName**. **TilesetName** is the name of the tileset and will be used in a manager class. **TilesetImageName** is the name of the file that holds the image for the tileset. There are then four integer fields that describe the tiles in a tileset. There are fields for the width and height of the tiles in the tileset, **TileWidthInPixels** and **TileHeightInPixels**. There are also fields for the number of tiles wide the tile set is and how many tiles high, **TilesWide** and **TilesHigh**. With those values you have enough information to pass to the constructor of the **Tileset** class to create a **Tileset**.

The next class to add is a class to represent a map layer as maps are made up of lists of **Tileset** and **MapLayer** objects. Right click the **WorldClasses** folder in the **RpgLibrary** folder, select **Add** and then **Class**. Name this new class **MapLayerData**. This is the code for that class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RpgLibrary.WorldClasses
{
    public struct Tile
    {
        public int TileIndex;
        public int TileSetIndex;
        public Tile(int tileIndex, int tileSetIndex)
        {
            TileIndex = tileIndex;
            TileSetIndex = tileSetIndex;
        }
    }

    public class MapLayerData
    {
        public string MapLayerName;
        public int Width;
        public int Height;
        public Tile[] Layer;

        private MapLayerData()
        {
        }

        public MapLayerData(string mapLayerName, int width, int height)
        {
            MapLayerName = mapLayerName;
            Width = width;
            Height = height;
            Layer = new Tile[height * width];
        }

        public void SetTile(int x, int y, Tile tile)
        {
            Layer[y * Width + x] = tile;
        }

        public Tile GetTile(int x, int y)
        {
            return Layer[y * Width + x];
        }
    }
}
```

I added a structure to this class for tiles. It has public integer fields for the index of the tile and the index of the tileset. I included a constructor that requires two parameters: **tileIndex** and **tileSetIndex**. They are for the index of the tile and the index of the tileset respectively. Making it a structure rather than a class makes it a value type rather than a reference type. So, when I create the array for tiles in the **MapLayerData** the tiles automatically have values.

The **MapLayerData** class has four public fields: **Layer** that is a single dimension array of **Tile**, **MapLayerName** that is a string for the name of the layer, **Width** and **Height** that are integers and are the width and height of the map. Why did I choose a single dimension array rather than a two dimension array like in the tile engine? You can't serialize arrays with more than one dimension. You can simulate a multidimension array in a one dimension array though.

There are two constructors for the **MapLayerData** class. There is a private constructor that takes no parameters that will be used in serializing and deserializing the class. The public constructor takes three paramaters: **mapLayerName** that is the name of the map layer, **width** and **height** that are the width and height of the map. The public constructor first sets the **MapLayerName**, **Width**, and **Height** fields to the values passed in. I then create a single dimension array of **Tile** that is **height** times **width**.

To find out how many elements are in a 2D array you multiple the width of the array by the height of the array so to represent a 2D array in a 1D array you need an array that is that size.

There are two public methods in this class. The first, **SetTile**, sets that tile that is represented at coordinates **x** and **y** to the **Tile** passed in. To set the element, and retrieve the element, you need to be consistent in determining the index. I used the calculation **y * Width + x**. Lets think about this a bit. Take a small 2D array, [4, 3]. You can represent it using a 1D array [12]. You have **y** between 0 and 3 and **x** between 0 and 2 in loops like this.

```
       for (y = 0; y < 4; y++)
           for (x = 0; x < 3; x++)
```

When your **y** is at 0 the calculation for the 1D array will be 0 * 3 + 0 = 0, 0 * 3 + 1 = 1, and 0 * 3 + 2 = 2. Then if you move to **y** at 1 you will get 1 * 3 + 0 = 3, 1 * 3 + 1 = 4, and 1 * 3 + 2 = 5. When you then move to **y** at 2 you will get 2 * 3 + 0 = 6, 2 * 3 + 1 = 7, and 2 * 3 + 2 = 8. Finally, when you move **y** to 3 you will get 3 * 3 + 0 = 9, 3 * 3 + 1 = 10, and 3 * 3 + 2 = 11. You can see that using the calculation **y * WidthOfArray + x** you can determine the position of a 2D element at (x, y) in 1D array. This is pretty much what the computer does when you declare a 2D array. It sets aside an area of memory the size of the array. It uses calculations like above to decide where the element resides in memory.

Now that you have classes that represent a tileset and a map layer you can create a class that can represent an entire map. Right click the **WorldClasses** in the **RpgLibrary** project, select **Add** and then **Class**. Name this new class **MapData**. The code for that class follows next.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace RpgLibrary.WorldClasses
{
    public class MapData
    {
        public string MapName;
        public MapLayerData[] Layers;
        public TilesetData[] Tilesets;

        private MapData()
        {
        }

        public MapData(string mapName, List<TilesetData> tilesets, List<MapLayerData> layers)
        {
            MapName = mapName;
            Tilesets = tilesets.ToArray();
            Layers = layers.ToArray();
        }
    }
}
```

Not a very complex class. There are just three fields. A string for the name of the map, an array of **MapLayerData** for the layers in the map, and an array of **TilesetData** for the tilesets in the map. There is a private constructor that will be used in serializing and deserializing the map and a public constructor that takes three parameters: the name of the map, a **List<TilesetData>** for the tilesets the maps uses, and a **List<MapLayerData>** for the layers in the map. It sets the fields using the values that are passed to the constructor. I use the **ToArray** method of the **List<T>** class to convert the lists to arrays.

I want to add another data class, for levels. Right click the **WorldClasses** in the **RpgLibrary** project, select **Add** and then **Class**. Name this new class **LevelData**. This is the code for that class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RpgLibrary.WorldClasses
{
    public class LevelData
    {
        public string LevelName;
        public string MapName;
        public int MapWidth;
        public int MapHeight;
        public string[] CharacterNames;
        public string[] ChestNames;
        public string[] TrapNames;

        private LevelData()
        {
        }

        public LevelData(
            string levelName,
            string mapName,
            int mapWidth,
            int mapHeight,
            List<string> characterNames,
            List<string> chestNames,
            List<string> trapNames)
        {
            LevelName = levelName;
            MapName = mapName;
            MapWidth = mapWidth;
            MapHeight = mapHeight;
            CharacterNames = characterNames.ToArray();
            ChestNames = chestNames.ToArray();
            TrapNames = trapNames.ToArray();
        }
    }
}
```

Another rather straight forward class. There are fields for the name of the level, **LevelName**, the map for the level, **MapName**, the width and height of the map**, MapWidth** and **MapHeight**, the names of characters on the map, **CharacterNames**, the name of chests on the map**, ChestNames**, and the names of traps on the map, **TrapNames**. The private constructor will be used in serializing and deserializing objects. The public constructor takes a parameter for each of the fields. For the arrays of strings I pass in **List<string>** and use the **ToArray** method of **List<T>**.

I'm going to move back to the editor now. I want to add in a form for creating a new level. Right click the **XLevelEditor** project in the solution explorer, select **Add** and then **Windows Form**. Name this new form **FormNewLevel**. Here is the code for the designer.

```
namespace XLevelEditor
{
    partial class FormNewLevel
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.lblLevelName = new System.Windows.Forms.Label();
            this.tbLevelName = new System.Windows.Forms.TextBox();
            this.lblMapWidth = new System.Windows.Forms.Label();
            this.mtbWidth = new System.Windows.Forms.MaskedTextBox();
            this.lblMapHeight = new System.Windows.Forms.Label();
            this.mtbHeight = new System.Windows.Forms.MaskedTextBox();
            this.btnOK = new System.Windows.Forms.Button();
            this.btnCancel = new System.Windows.Forms.Button();
            this.lblMapName = new System.Windows.Forms.Label();
            this.tbMapName = new System.Windows.Forms.TextBox();
            this.SuspendLayout();
            //
            // lblLevelName
            //
            this.lblLevelName.AutoSize = true;
            this.lblLevelName.Location = new System.Drawing.Point(13, 15);
            this.lblLevelName.Name = "lblLevelName";
            this.lblLevelName.Size = new System.Drawing.Size(87, 17);
            this.lblLevelName.TabIndex = 0;
            this.lblLevelName.Text = "Level Name:";
            //
            // tbLevelName
            //
            this.tbLevelName.Location = new System.Drawing.Point(106, 12);
            this.tbLevelName.Name = "tbLevelName";
            this.tbLevelName.Size = new System.Drawing.Size(100, 22);
            this.tbLevelName.TabIndex = 1;
            //
            // lblMapWidth
            //
            this.lblMapWidth.AutoSize = true;
            this.lblMapWidth.Location = new System.Drawing.Point(21, 75);
            this.lblMapWidth.Name = "lblMapWidth";
            this.lblMapWidth.Size = new System.Drawing.Size(79, 17);
            this.lblMapWidth.TabIndex = 4;
            this.lblMapWidth.Text = "Map Width:";
            //
            // mtbWidth
            //
            this.mtbWidth.Location = new System.Drawing.Point(106, 72);
            this.mtbWidth.Mask = "0000";
            this.mtbWidth.Name = "mtbWidth";
            this.mtbWidth.Size = new System.Drawing.Size(45, 22);
            this.mtbWidth.TabIndex = 5;
            //
            // lblMapHeight
            //
            this.lblMapHeight.AutoSize = true;
            this.lblMapHeight.Location = new System.Drawing.Point(13, 106);
            this.lblMapHeight.Name = "lblMapHeight";
            this.lblMapHeight.Size = new System.Drawing.Size(84, 17);
            this.lblMapHeight.TabIndex = 6;
            this.lblMapHeight.Text = "Map Height:";
            //
            // mtbHeight
            //
            this.mtbHeight.Location = new System.Drawing.Point(106, 103);
            this.mtbHeight.Mask = "0000";
            this.mtbHeight.Name = "mtbHeight";
            this.mtbHeight.Size = new System.Drawing.Size(45, 22);
            this.mtbHeight.TabIndex = 7;
            //
            // btnOK
            //
            this.btnOK.Location = new System.Drawing.Point(13, 142);
            this.btnOK.Name = "btnOK";
            this.btnOK.Size = new System.Drawing.Size(75, 23);
            this.btnOK.TabIndex = 8;
            this.btnOK.Text = "OK";
            this.btnOK.UseVisualStyleBackColor = true;
            //
            // btnCancel
            //
            this.btnCancel.Location = new System.Drawing.Point(106, 142);
            this.btnCancel.Name = "btnCancel";
            this.btnCancel.Size = new System.Drawing.Size(75, 23);
            this.btnCancel.TabIndex = 9;
            this.btnCancel.Text = "Cancel";
            this.btnCancel.UseVisualStyleBackColor = true;
            //
            // lblMapName
            //
            this.lblMapName.AutoSize = true;
            this.lblMapName.Location = new System.Drawing.Point(17, 42);
```

```
            this.lblMapName.Name = "lblMapName";
            this.lblMapName.Size = new System.Drawing.Size(80, 17);
            this.lblMapName.TabIndex = 2;
            this.lblMapName.Text = "Map Name:";
            //
            // tbMapName
            //
            this.tbMapName.Location = new System.Drawing.Point(106, 42);
            this.tbMapName.Name = "tbMapName";
            this.tbMapName.Size = new System.Drawing.Size(100, 22);
            this.tbMapName.TabIndex = 3;
            //
            // FormNewLevel
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(250, 225);
            this.ControlBox = false;
            this.Controls.Add(this.tbMapName);
            this.Controls.Add(this.lblMapName);
            this.Controls.Add(this.btnCancel);
            this.Controls.Add(this.btnOK);
            this.Controls.Add(this.mtbHeight);
            this.Controls.Add(this.lblMapHeight);
            this.Controls.Add(this.mtbWidth);
            this.Controls.Add(this.lblMapWidth);
            this.Controls.Add(this.tbLevelName);
            this.Controls.Add(this.lblLevelName);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Name = "FormNewLevel";
            this.StartPosition = System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "New Level";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label lblLevelName;
        private System.Windows.Forms.TextBox tbLevelName;
        private System.Windows.Forms.Label lblMapWidth;
        private System.Windows.Forms.MaskedTextBox mtbWidth;
        private System.Windows.Forms.Label lblMapHeight;
        private System.Windows.Forms.MaskedTextBox mtbHeight;
        private System.Windows.Forms.Button btnOK;
        private System.Windows.Forms.Button btnCancel;
        private System.Windows.Forms.Label lblMapName;
        private System.Windows.Forms.TextBox tbMapName;
    }
}
```

Now that the form has been designed it is time to code its logic. Right click **FormNewLevel** in the **XLevelEditor** project and select **View Code**. Change the code for the form to the following.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using RpgLibrary.WorldClasses;

namespace XLevelEditor
{
    public partial class FormNewLevel : Form
    {
        #region Field Region

        bool okPressed;
        LevelData levelData;

        #endregion

        #region Property Region

        public bool OKPressed
        {
            get { return okPressed; }
        }

        public LevelData LevelData
        {
            get { return levelData; }
        }

        #endregion

        #region Constructor Region

        public FormNewLevel()
        {
            InitializeComponent();
            btnOK.Click += new EventHandler(btnOK_Click);
            btnCancel.Click += new EventHandler(btnCancel_Click);
        }

        #endregion

        #region Button Event Handler Region

        void btnOK_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(tbLevelName.Text))
            {
                MessageBox.Show("You must enter a name for the level.", "Missing Level Name");
                return;
            }

            if (string.IsNullOrEmpty(tbMapName.Text))
            {
                MessageBox.Show("You must enter a name for the map of the level.", "Missing Map 
Name");
                return;
            }

            int mapWidth = 0;
            int mapHeight = 0;

            if (!int.TryParse(mtbWidth.Text, out mapWidth) || mapWidth < 1)
            {
                MessageBox.Show("The width of the map must be greater than or equal to one.",
                    "Map Size Error");
                return;
            }

            if (!int.TryParse(mtbHeight.Text, out mapHeight) || mapHeight < 1)
            {
                MessageBox.Show("The height of the map must be greater than or equal to one.",
                    "Map Size Error");
                return;
            }

            levelData = new RpgLibrary.WorldClasses.LevelData(
                tbLevelName.Text,
                tbMapName.Text,
                mapWidth,
                mapHeight,
                new List<string>(),
                new List<string>(),
                new List<string>());

            okPressed = true;
            this.Close();
        }

        void btnCancel_Click(object sender, EventArgs e)
        {
            okPressed = false;
            this.Close();
        }

        #endregion
    }
}
```

There is a using statement to bring the **LevelData** class that I added into scope. There are two fields in this class and properties to read their values, but not write them. The first, **okPressed**, determines how the form was closed, by the OK button or the Cancel button. The second, **LevelData**, holds the information off the form if the form was closed successfully. The constructor wires event handlers for the **Click** events of the buttons.

In the handler for the **Click** event for **btnOK** I validate the form. If the **Text** property of **tbLevelName** is null or empty I display a message box stating that a name must be entered for the level and exit the method. Similarly, I display a message and exit if the **Text** property of **tbMapName** is null or empty. There are then two local variables that will hold the conversion of the **Text** property of the **Masked Text Boxes** into integers. I use the **TrParse** method to attempt to parse the **Text** property of **mtbWidth**. If the return value is false the second half of the expression will not be evaluated. If the attempt was successful I then check if the result is less than 1. If either evaluate to true I display a message and exit the method. I do something similar for **mtbHeight** but work with height instead of width. You should really do better evaluation than the map width and height less than 1. If the form passed validation I create a new **LevelData** object using the **Text** properties of the two **Text Boxes**, the **mapWidth** and

mapHeight local variables, and new **List<string>** instance for the **List<string>** parameters. I then set **okPressed** to true and close the form.

The handler for the **Click** event for **btnCancel** is much simpler. It just sets the **okPressed** field to false and then closes the form.

I want to create a form for creating **TilesetData** objects. Right click the **XLevelEditor**, select **Add** and then **Windows Form**. Name this new form **FormNewTileset**. The code for the designer follows next.

```
namespace XLevelEditor
{
    partial class FormNewTileset
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, 
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.lblTilesetName = new System.Windows.Forms.Label();
            this.tbTilesetName = new System.Windows.Forms.TextBox();
            this.lblTilesetImageName = new System.Windows.Forms.Label();
            this.tbTilesetImage = new System.Windows.Forms.TextBox();
            this.btnSelectImage = new System.Windows.Forms.Button();
            this.lblTileWidth = new System.Windows.Forms.Label();
            this.mtbTileWidth = new System.Windows.Forms.MaskedTextBox();
            this.lblTileHeight = new System.Windows.Forms.Label();
            this.mtbTileHeight = new System.Windows.Forms.MaskedTextBox();
            this.lblTilesWide = new System.Windows.Forms.Label();
            this.mtbTilesWide = new System.Windows.Forms.MaskedTextBox();
            this.lblTilesHigh = new System.Windows.Forms.Label();
            this.mtbTilesHigh = new System.Windows.Forms.MaskedTextBox();
            this.btnOK = new System.Windows.Forms.Button();
            this.btnCancel = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // lblTilesetName
```

```
//
this.lblTilesetName.AutoSize = true;
this.lblTilesetName.Location = new System.Drawing.Point(54, 10);
this.lblTilesetName.Name = "lblTilesetName";
this.lblTilesetName.Size = new System.Drawing.Size(95, 17);
this.lblTilesetName.TabIndex = 0;
this.lblTilesetName.Text = "Tileset Name:";
//
// tbTilesetName
//
this.tbTilesetName.Location = new System.Drawing.Point(155, 5);
this.tbTilesetName.Name = "tbTilesetName";
this.tbTilesetName.Size = new System.Drawing.Size(100, 22);
this.tbTilesetName.TabIndex = 1;
//
// lblTilesetImageName
//
this.lblTilesetImageName.AutoSize = true;
this.lblTilesetImageName.Location = new System.Drawing.Point(12, 40);
this.lblTilesetImageName.Name = "lblTilesetImageName";
this.lblTilesetImageName.Size = new System.Drawing.Size(137, 17);
this.lblTilesetImageName.TabIndex = 2;
this.lblTilesetImageName.Text = "Tileset Image Name:";
//
// tbTilesetImage
//
this.tbTilesetImage.Enabled = false;
this.tbTilesetImage.Location = new System.Drawing.Point(155, 34);
this.tbTilesetImage.Name = "tbTilesetImage";
this.tbTilesetImage.Size = new System.Drawing.Size(100, 22);
this.tbTilesetImage.TabIndex = 3;
//
// btnSelectImage
//
this.btnSelectImage.Location = new System.Drawing.Point(261, 34);
this.btnSelectImage.Name = "btnSelectImage";
this.btnSelectImage.Size = new System.Drawing.Size(28, 23);
this.btnSelectImage.TabIndex = 4;
this.btnSelectImage.Tag = "";
this.btnSelectImage.Text = "...";
this.btnSelectImage.UseVisualStyleBackColor = true;
//
// lblTileWidth
//
this.lblTileWidth.AutoSize = true;
this.lblTileWidth.Location = new System.Drawing.Point(74, 69);
this.lblTileWidth.Name = "lblTileWidth";
this.lblTileWidth.Size = new System.Drawing.Size(75, 17);
this.lblTileWidth.TabIndex = 5;
this.lblTileWidth.Text = "Tile Width:";
//
// mtbTileWidth
//
this.mtbTileWidth.Location = new System.Drawing.Point(155, 62);
this.mtbTileWidth.Mask = "000";
this.mtbTileWidth.Name = "mtbTileWidth";
this.mtbTileWidth.Size = new System.Drawing.Size(34, 22);
this.mtbTileWidth.TabIndex = 6;
//
// lblTileHeight
```

```
//
this.lblTileHeight.AutoSize = true;
this.lblTileHeight.Location = new System.Drawing.Point(69, 93);
this.lblTileHeight.Name = "lblTileHeight";
this.lblTileHeight.Size = new System.Drawing.Size(80, 17);
this.lblTileHeight.TabIndex = 7;
this.lblTileHeight.Text = "Tile Height:";
//
// mtbTileHeight
//
this.mtbTileHeight.Location = new System.Drawing.Point(155, 90);
this.mtbTileHeight.Mask = "000";
this.mtbTileHeight.Name = "mtbTileHeight";
this.mtbTileHeight.Size = new System.Drawing.Size(34, 22);
this.mtbTileHeight.TabIndex = 8;
//
// lblTilesWide
//
this.lblTilesWide.AutoSize = true;
this.lblTilesWide.Location = new System.Drawing.Point(17, 121);
this.lblTilesWide.Name = "lblTilesWide";
this.lblTilesWide.Size = new System.Drawing.Size(132, 17);
this.lblTilesWide.TabIndex = 9;
this.lblTilesWide.Text = "Number Tiles Wide:";
//
// mtbTilesWide
//
this.mtbTilesWide.Location = new System.Drawing.Point(155, 118);
this.mtbTilesWide.Mask = "000";
this.mtbTilesWide.Name = "mtbTilesWide";
this.mtbTilesWide.Size = new System.Drawing.Size(34, 22);
this.mtbTilesWide.TabIndex = 10;
//
// lblTilesHigh
//
this.lblTilesHigh.AutoSize = true;
this.lblTilesHigh.Location = new System.Drawing.Point(20, 150);
this.lblTilesHigh.Name = "lblTilesHigh";
this.lblTilesHigh.Size = new System.Drawing.Size(129, 17);
this.lblTilesHigh.TabIndex = 11;
this.lblTilesHigh.Text = "Number Tiles High:";
//
// mtbTilesHigh
//
this.mtbTilesHigh.Location = new System.Drawing.Point(155, 147);
this.mtbTilesHigh.Mask = "000";
this.mtbTilesHigh.Name = "mtbTilesHigh";
this.mtbTilesHigh.Size = new System.Drawing.Size(33, 22);
this.mtbTilesHigh.TabIndex = 12;
//
// btnOK
//
this.btnOK.Location = new System.Drawing.Point(57, 175);
this.btnOK.Name = "btnOK";
this.btnOK.Size = new System.Drawing.Size(75, 23);
this.btnOK.TabIndex = 13;
this.btnOK.Text = "OK";
this.btnOK.UseVisualStyleBackColor = true;
//
// btnCancel
```

```
            //
            this.btnCancel.Location = new System.Drawing.Point(155, 175);
            this.btnCancel.Name = "btnCancel";
            this.btnCancel.Size = new System.Drawing.Size(75, 23);
            this.btnCancel.TabIndex = 14;
            this.btnCancel.Text = "Cancel";
            this.btnCancel.UseVisualStyleBackColor = true;
            //
            // FormNewTileset
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(325, 250);
            this.ControlBox = false;
            this.Controls.Add(this.btnCancel);
            this.Controls.Add(this.btnOK);
            this.Controls.Add(this.mtbTilesHigh);
            this.Controls.Add(this.lblTilesHigh);
            this.Controls.Add(this.mtbTilesWide);
            this.Controls.Add(this.lblTilesWide);
            this.Controls.Add(this.mtbTileHeight);
            this.Controls.Add(this.lblTileHeight);
            this.Controls.Add(this.mtbTileWidth);
            this.Controls.Add(this.lblTileWidth);
            this.Controls.Add(this.btnSelectImage);
            this.Controls.Add(this.tbTilesetImage);
            this.Controls.Add(this.lblTilesetImageName);
            this.Controls.Add(this.tbTilesetName);
            this.Controls.Add(this.lblTilesetName);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Name = "FormNewTileset";
            this.StartPosition = System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "New Tileset";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label lblTilesetName;
        private System.Windows.Forms.TextBox tbTilesetName;
        private System.Windows.Forms.Label lblTilesetImageName;
        private System.Windows.Forms.TextBox tbTilesetImage;
        private System.Windows.Forms.Button btnSelectImage;
        private System.Windows.Forms.Label lblTileWidth;
        private System.Windows.Forms.MaskedTextBox mtbTileWidth;
        private System.Windows.Forms.Label lblTileHeight;
        private System.Windows.Forms.MaskedTextBox mtbTileHeight;
        private System.Windows.Forms.Label lblTilesWide;
        private System.Windows.Forms.MaskedTextBox mtbTilesWide;
        private System.Windows.Forms.Label lblTilesHigh;
        private System.Windows.Forms.MaskedTextBox mtbTilesHigh;
        private System.Windows.Forms.Button btnOK;
        private System.Windows.Forms.Button btnCancel;
    }
}
```

Now I'm going to add some code to **FormNewTileset**. Right click **FormNewTileset** in the solution explorer and select **View Code**. Change the code to the following.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using RpgLibrary.WorldClasses;

namespace XLevelEditor
{
    public partial class FormNewTileset : Form
    {
        #region Field Region

        bool okPressed;
        TilesetData tilesetData;

        #endregion

        #region Property Region

        public TilesetData TilesetData
        {
            get { return tilesetData; }
        }

        public bool OKPressed
        {
            get { return okPressed; }
        }

        #endregion

        #region Constructor Region

        public FormNewTileset()
        {
            InitializeComponent();

            btnSelectImage.Click += new EventHandler(btnSelectImage_Click);
            btnOK.Click += new EventHandler(btnOK_Click);
            btnCancel.Click += new EventHandler(btnCancel_Click);
        }

        #endregion

        #region Button Event Handler Region

        void btnSelectImage_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofDialog = new OpenFileDialog();

            ofDialog.Filter = "Image Files|*.BMP;*.GIF;*.JPG;*.TGA;*.PNG";
            ofDialog.CheckFileExists = true;
            ofDialog.CheckPathExists = true;
            ofDialog.Multiselect = false;
            DialogResult result = ofDialog.ShowDialog();

            if (result == DialogResult.OK)
            {
                tbTilesetImage.Text = ofDialog.FileName;
            }
        }

        void btnOK_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(tbTilesetName.Text))
            {
                MessageBox.Show("You must enter a name for the tileset.");
                return;
            }

            if (string.IsNullOrEmpty(tbTilesetImage.Text))
            {
                MessageBox.Show("You must select an image for the tileset.");
                return;
            }

            int tileWidth = 0;
            int tileHeight = 0;
            int tilesWide = 0;
            int tilesHigh = 0;

            if (!int.TryParse(mtbTileWidth.Text, out tileWidth))
            {
                MessageBox.Show("Tile width must be an integer value.");
                return;
            }
            else if (tileWidth < 0)
            {
                MessageBox.Show("Tile width must me greater than zero.");
                return;
            }

            if (!int.TryParse(mtbTileHeight.Text, out tileHeight))
            {
                MessageBox.Show("Tile height must be an integer value.");
                return;
            }
            else if (tileHeight < 0)
            {
                MessageBox.Show("Tile height must be greater than zero.");
                return;
            }

            if (!int.TryParse(mtbTilesWide.Text, out tilesWide))
            {
                MessageBox.Show("Tiles wide must be an integer value.");
                return;
            }
            else if (tilesWide < 0)
            {
                MessageBox.Show("Tiles wide must me greater than zero.");
                return;
            }
```

```
            if (!int.TryParse(mtbTilesHigh.Text, out tilesHigh))
            {
                MessageBox.Show("Tiles high must be an integer value.");
                return;
            }
            else if (tilesHigh < 0)
            {
                MessageBox.Show("Tiles high must be greater than zero.");
                return;
            }

            tilesetData = new TilesetData();
            tilesetData.TilesetName = tbTilesetName.Text;
            tilesetData.TilesetImageName = tbTilesetImage.Text;
            tilesetData.TileWidthInPixels = tileWidth;
            tilesetData.TileHeightInPixels = tileHeight;
            tilesetData.TilesWide = tilesWide;
            tilesetData.TilesHigh = tilesHigh;

            okPressed = true;

            this.Close();
        }

        void btnCancel_Click(object sender, EventArgs e)
        {
            okPressed = false;
            this.Close();
        }

        #endregion
    }
}
```

There is a using statement to bring the **TilesetData** class into scope. The first field, **okPressed**, and is a bool that will hold how the form was closed. The other field, **tilesetData**, is of type **TilesetData** will hold an object if the form is successfully closed using the OK button. There are properties to expose the values of the fields as read only, or get only. The constructor wires event handlers for the **Click** event of all three buttons.

In the handler for **btnSelectImage** I create a **OpenFileDialog** object. I set the **Filter** property so that the dialog will filter will show common image files. I set the **CheckFileExsists** and **CheckPathExists** properties to true so that we are sure the file exists. I also set **Multiselect** to false so that the user can only select one file. I capture the result of the **ShowDialog** method and compare it to **OK**. If the dialog was closed using the **OK** button I set the **Text** property of **tbTilesetImage** to the **FileName** property of the dialog.

The handler for **btnOK** does a lot of validation on the values on the form. If the **Text** property of either **tbTilesetName** or **tbTilesetImage** is null or empty I display a message box and exit the method. There are four local variables to hold the result of converting the **Text** property of the **Masked Text Boxes** to integers. I use the **TryParse** method of the **int** class try and convert the **Text** properties. If the conversion fails I display a message saying the value must be an integer value and exit the method. If it succeeded I check to see if the value is less than 1. If it is less than 1 I display a message saying the value must be greater than zero and exit the method. If the data on the form is valid I create a **TilesetData** object and set the fields to the values from the form. I set the **okPressed** field to true and then close the form. The handler for **btnCancel** sets the **okPressed** field to false and closes the form.

The next form that I want to create is a form for making new layers of the map. Right click the **XLevelEditor** project in the solution explorer, select **Add** and then **Windows Form**. Name this new form **FormNewLayer**. This is the code for the designer.

```
namespace XLevelEditor
{
    partial class FormNewLayer
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.lblLayerName = new System.Windows.Forms.Label();
            this.tbLayerName = new System.Windows.Forms.TextBox();
            this.gbFillLayer = new System.Windows.Forms.GroupBox();
            this.cbFill = new System.Windows.Forms.CheckBox();
            this.lblTilesetIndex = new System.Windows.Forms.Label();
            this.lblTileIndex = new System.Windows.Forms.Label();
            this.btnOK = new System.Windows.Forms.Button();
            this.btnCancel = new System.Windows.Forms.Button();
            this.nudTile = new System.Windows.Forms.NumericUpDown();
            this.nudTileset = new System.Windows.Forms.NumericUpDown();
            this.gbFillLayer.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.nudTile)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.nudTileset)).BeginInit();
            this.SuspendLayout();
            //
            // lblLayerName
            //
            this.lblLayerName.AutoSize = true;
            this.lblLayerName.Location = new System.Drawing.Point(27, 15);
            this.lblLayerName.Name = "lblLayerName";
            this.lblLayerName.Size = new System.Drawing.Size(89, 17);
            this.lblLayerName.TabIndex = 0;
            this.lblLayerName.Text = "Layer Name:";
            //
            // tbLayerName
            //
            this.tbLayerName.Location = new System.Drawing.Point(122, 12);
            this.tbLayerName.Name = "tbLayerName";
            this.tbLayerName.Size = new System.Drawing.Size(100, 22);
            this.tbLayerName.TabIndex = 1;
            //
            // gbFillLayer
            //
            this.gbFillLayer.Controls.Add(this.nudTileset);
            this.gbFillLayer.Controls.Add(this.nudTile);
            this.gbFillLayer.Controls.Add(this.lblTileIndex);
            this.gbFillLayer.Controls.Add(this.lblTilesetIndex);
            this.gbFillLayer.Location = new System.Drawing.Point(12, 71);
            this.gbFillLayer.Name = "gbFillLayer";
            this.gbFillLayer.Size = new System.Drawing.Size(210, 79);
            this.gbFillLayer.TabIndex = 3;
            this.gbFillLayer.TabStop = false;
            this.gbFillLayer.Text = "Fill With";
            //
            // cbFill
            //
            this.cbFill.AutoSize = true;
            this.cbFill.Checked = true;
            this.cbFill.CheckState = System.Windows.Forms.CheckState.Checked;
            this.cbFill.Location = new System.Drawing.Point(12, 44);
            this.cbFill.Name = "cbFill";
            this.cbFill.Size = new System.Drawing.Size(95, 21);
            this.cbFill.TabIndex = 2;
            this.cbFill.Text = "Fill Layer?";
            this.cbFill.UseVisualStyleBackColor = true;
            //
            // lblTilesetIndex
            //
            this.lblTilesetIndex.AutoSize = true;
            this.lblTilesetIndex.Location = new System.Drawing.Point(4, 50);
            this.lblTilesetIndex.Name = "lblTilesetIndex";
            this.lblTilesetIndex.Size = new System.Drawing.Size(91, 17);
            this.lblTilesetIndex.TabIndex = 2;
            this.lblTilesetIndex.Text = "Tileset Index:";
            //
            // lblTileIndex
            //
            this.lblTileIndex.AutoSize = true;
            this.lblTileIndex.Location = new System.Drawing.Point(23, 22);
            this.lblTileIndex.Name = "lblTileIndex";
            this.lblTileIndex.Size = new System.Drawing.Size(72, 17);
            this.lblTileIndex.TabIndex = 0;
            this.lblTileIndex.Text = "Tile Index:";
            //
            // btnOK
            //
            this.btnOK.Location = new System.Drawing.Point(30, 169);
            this.btnOK.Name = "btnOK";
            this.btnOK.Size = new System.Drawing.Size(75, 23);
            this.btnOK.TabIndex = 4;
            this.btnOK.Text = "OK";
            this.btnOK.UseVisualStyleBackColor = true;
            //
            // btnCancel
            //
            this.btnCancel.Location = new System.Drawing.Point(122, 169);
            this.btnCancel.Name = "btnCancel";
            this.btnCancel.Size = new System.Drawing.Size(75, 23);
            this.btnCancel.TabIndex = 5;
            this.btnCancel.Text = "Cancel";
            this.btnCancel.UseVisualStyleBackColor = true;
            //
            // nudTile
            //
            this.nudTile.Location = new System.Drawing.Point(101, 20);
            this.nudTile.Maximum = new decimal(new int[] {
            512,
            0,
            0,
            0});
            this.nudTile.Minimum = new decimal(new int[] {
            1,
            0,
            0,
            -2147483648});
            this.nudTile.Name = "nudTile";
            this.nudTile.Size = new System.Drawing.Size(84, 22);
            this.nudTile.TabIndex = 1;
            this.nudTile.Value = new decimal(new int[] {
            1,
            0,
            0,
            -2147483648});
            //
            // nudTileset
            //
            this.nudTileset.Location = new System.Drawing.Point(101, 48);
            this.nudTileset.Maximum = new decimal(new int[] {
            512,
            0,
            0,
            0});
            this.nudTileset.Minimum = new decimal(new int[] {
            1,
            0,
            0,
            -2147483648});
            this.nudTileset.Name = "nudTileset";
            this.nudTileset.Size = new System.Drawing.Size(84, 22);
            this.nudTileset.TabIndex = 3;
            this.nudTileset.Value = new decimal(new int[] {
            1,
            0,
            0,
            -2147483648});
            //
            // FormNewLayer
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
```

```
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(275, 225);
            this.ControlBox = false;
            this.Controls.Add(this.btnCancel);
            this.Controls.Add(this.btnOK);
            this.Controls.Add(this.cbFill);
            this.Controls.Add(this.gbFillLayer);
            this.Controls.Add(this.tbLayerName);
            this.Controls.Add(this.lblLayerName);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Name = "FormNewLayer";
            this.StartPosition = System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "New Layer";
            this.gbFillLayer.ResumeLayout(false);
            this.gbFillLayer.PerformLayout();
            ((System.ComponentModel.ISupportInitialize)(this.nudTile)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.nudTileset)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label lblLayerName;
        private System.Windows.Forms.TextBox tbLayerName;
        private System.Windows.Forms.GroupBox gbFillLayer;
        private System.Windows.Forms.NumericUpDown nudTileset;
        private System.Windows.Forms.NumericUpDown nudTile;
        private System.Windows.Forms.Label lblTileIndex;
        private System.Windows.Forms.Label lblTilesetIndex;
        private System.Windows.Forms.CheckBox cbFill;
        private System.Windows.Forms.Button btnOK;
        private System.Windows.Forms.Button btnCancel;
    }
}
```

Before I get to the code for this form I want to add another public constructor to the **MapLayerData** class. I will be adding in two more parameters. One will be a tile index and the other will be a tilset index. In that constructor I will fill the layer with tiles using the tile index and tileset index. Add the following constructor to the **MapLayerData** class.

```
        public MapLayerData(string mapLayerName, int width, int height, int tileIndex, int
tileSet)
        {
            MapLayerName = mapLayerName;
            Width = width;
            Height = height;
            Layer = new Tile[height * width];
            Tile tile = new Tile(tileIndex, tileSet);
            for (int y = 0; y < height; y++)
                for (int x = 0; x < width; x++)
                    SetTile(x, y, tile);
        }
```

There is a **Tile** variable called **tile** that is created using the parameters passed in. It is safe to do this because the **Tile** is a structure and not a class. That makes it a value type and not a reference type. So if you modify one of the **Tile** objects in the array it will not affect the others. There is then a set of nested loops added to this constructor. The outer loop is for the Y coordinate and the inner loops is for the X coordinate. I call the **SetTile** method passing in **x**, **y**, and the **Tile** I created before.

I want to extend the **MapLayer** class in the **XRpgLibrary** to include a static method that will return a **MapLayer** from a **MapLayerData**. Add the following using statement and method to the **MapLayer** class in the **MGRpgLibrary**.

```
using RpgLibrary.WorldClasses;

public static MapLayer FromMapLayerData(MapLayerData data)
{
    MapLayer layer = new MapLayer(data.Width, data.Height);

    for (int y = 0; y < data.Height; y++)
        for (int x = 0; x < data.Width; x++)
        {
            layer.SetTile(
                x,
                y,
                data.GetTile(x, y).TileIndex,
                data.GetTile(x, y).TileSetIndex);
        }

    return layer;
}
```

The new method takes a **MapLayerData** parameter that is the object you want to convert. Inside the method I create a new **MapLayer** instance using the **Height** and **Width** properties of the object passed in. There is then a set of nested for loops. The outer loop will loop through the height of the layer and the inner array will loop through the width of the layer. Inside the loop I use the **SetTile** method of the **MapLayer** class to set the tile passing in values from the **GetTile** method of **MapLayerData** to get the tile at the (x, y) coordinates from the loop variables.

I'm now going to add the code for **FormNewLayer**. Right click **FormNewLayer** in the **XLevelEditor** and select **View Code**. Change the code to the following.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using RpgLibrary.WorldClasses;

namespace XLevelEditor
{
    public partial class FormNewLayer : Form
    {
        #region Field Region

        bool okPressed;
        int LayerWidth;
        int LayerHeight;
        MapLayerData mapLayerData;

        #endregion
        #region Property Region

        public bool OKPressed
        {
            get { return okPressed; }
        }

        public MapLayerData MapLayerData
        {
            get { return mapLayerData; }
        }

        #endregion

        #region Constructor Region

        public FormNewLayer(int width, int height)
        {
            InitializeComponent();

            LayerWidth = width;
            LayerHeight = height;

            btnOK.Click += new EventHandler(btnOK_Click);
            btnCancel.Click += new EventHandler(btnCancel_Click);
        }

        #endregion

        #region Button Event Handler Region

        void btnOK_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(tbLayerName.Text))
            {
                MessageBox.Show("You must enter a name for the layer.", "Missing Layer Name");
                return;
            }

            if (cbFill.Checked)
            {
                mapLayerData = new MapLayerData(
                    tbLayerName.Text,
                    LayerWidth,
                    LayerHeight,
                    (int)nudTile.Value,
                    (int)nudTileset.Value);
            }
            else
            {
                mapLayerData = new MapLayerData(
                    tbLayerName.Text,
                    LayerWidth,
                    LayerHeight);
            }
```

```
            okPressed = true;

            this.Close();
        }

        void btnCancel_Click(object sender, EventArgs e)
        {
            okPressed = false;

            this.Close();
        }

        #endregion
    }
}
```

There is a using statement to bring the **WorldClasses** from the **RpgLibrary** into scope, namely the **MapLayerData** class. There are four fields in the class. The first, **okPressed**, will be used to determine how the form closed. Then next two, **LayerWidth** and **LayerHeight**, are the width and height of the map. I'm not allowing layers of varying sizes so all layers should have the same height and width. The last field is a **MapLayerData** object. If the **OK** button is pressed I will set it to a new **MapLayerData** object. There are properties to expose the **okPressed** and **mapLayerData** fields as get only.

Forms are just classes and because they are just classes you can create new constructors for them to get values needed by the form to the form. I changed the constructor to take two integer parameters, the width and height of the map. In the constructor I set the **LayerWidth** and **LayerHeight** fields with the values passed in. I then wire the **Click** event for both of the buttons.

In the event handler for the **Click** event of **btnOK** I check to see if the **Text** property of **tbLayerName** is null or empty. If it is I display a message box saying that a name for the layer is required and exit the method. I then check the **Checked** property of **cbFill** to see if the map should be filled with a specific tile. If that is true I create a new map using the **Text** property of **tbLayerName**, the **LayerWidth** and **LayerHeight** fields, and the **Value** property of **nudTile** and **nudTileset** cast to an integer. The **Value** property of the **Numeric Up and Down** control is a decimal so you need to convert it to an integer. If it wasn't clicked I create a new layer using the **Text** property of **tbLayerName** and the **LayerWidth** and **LayerHeight** fields. I then set **okPressed** to true and close the form.

The code for the **Click** event of **btnCancel** you've seen several times now. I set **okPressed** to false and close the form.

For the rest of the tutorial I will be coding the actual level editor now that all of the basic pieces we need are in place. In this tutorial we will be working mostly in the code for **FormMain**, the actual level editor. Right click **FormMain** in the **XLevelEditor** project and select **View Code**. A lot of code is going to go into the code for the form so I'm going to do it in stages. To start with I'm going to set up the using statements, a few fields and properties, wire some event handlers, and set some properties of controls. Change the code for **FormMain** to the following.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using GDIBitmap = System.Drawing.Bitmap;
using GDIColor = System.Drawing.Color;
using GDIImage = System.Drawing.Image;
using GDIGraphics = System.Drawing.Graphics;
using GDIGraphicsUnit = System.Drawing.GraphicsUnit;
using GDIRectangle = System.Drawing.Rectangle;
using RpgLibrary.WorldClasses;
using MGRpgLibrary.TileEngine;

namespace XLevelEditor
{
    public partial class FormMain : Form
    {
        #region Field Region

        LevelData levelData;
        public static TileMap map;
        public static List<Tileset> tileSets = new List<Tileset>();
        public static List<MapLayer> layers = new List<MapLayer>();

        #endregion
        #region Property Region

        public GraphicsDevice GraphicsDevice
        {
            get { return mapDisplay.GraphicsDevice; }
        }

        #endregion

        #region Constructor Region

        public FormMain()
        {
            InitializeComponent();

            this.Load += new EventHandler(FormMain_Load);
            this.FormClosing += new FormClosingEventHandler(FormMain_FormClosing);

            tilesetToolStripMenuItem.Enabled = false;
            mapLayerToolStripMenuItem.Enabled = false;
            charactersToolStripMenuItem.Enabled = false;
            chestsToolStripMenuItem.Enabled = false;
            keysToolStripMenuItem.Enabled = false;

            newLevelToolStripMenuItem.Click +=
                new EventHandler(newLevelToolStripMenuItem_Click);
            newTilesetToolStripMenuItem.Click += new
                EventHandler(newTilesetToolStripMenuItem_Click);
            newLayerToolStripMenuItem.Click += new
                EventHandler(newLayerToolStripMenuItem_Click);

            mapDisplay.MouseEnter += new EventHandler(mapDisplay_MouseEnter);
            mapDisplay.MouseLeave += new EventHandler(mapDisplay_MouseLeave);
            mapDisplay.MouseMove += new MouseEventHandler(mapDisplay_MouseMove);
            mapDisplay.MouseDown += new MouseEventHandler(mapDisplay_MouseDown);
            mapDisplay.MouseUp += new MouseEventHandler(mapDisplay_MouseUp);
        }

        #endregion

        #region Form Event Handler Region

        void FormMain_Load(object sender, EventArgs e)
        {
            Application.Idle += new EventHandler(Application_Idle);
        }

        void FormMain_FormClosing(object sender, FormClosingEventArgs e)
        {
        }

        void Application_Idle(object sender, EventArgs e)
        {
            mapDisplay.Invalidate();
        }

        #endregion

        #region New Menu Item Event Handler Region

        void newLevelToolStripMenuItem_Click(object sender, EventArgs e)
        {
            using (FormNewLevel frmNewLevel = new FormNewLevel())
            {
                frmNewLevel.ShowDialog();

                if (frmNewLevel.OKPressed)
                {
                    levelData = frmNewLevel.LevelData;
                    tilesetToolStripMenuItem.Enabled = true;
                }
            }
        }

        void newTilesetToolStripMenuItem_Click(object sender, EventArgs e)
        {
            using (FormNewTileset frmNewTileset = new FormNewTileset())
            {
                frmNewTileset.ShowDialog();

                if (frmNewTileset.OKPressed)
                {
                    TilesetData data = frmNewTileset.TilesetData;
                    mapLayerToolStripMenuItem.Enabled = true;
                }
            }
        }

        void newLayerToolStripMenuItem_Click(object sender, EventArgs e)
        {
            using (FormNewLayer frmNewLayer = new FormNewLayer(levelData.MapWidth,
                levelData.MapHeight))
            {
```

```
                frmNewLayer.ShowDialog();
                if (frmNewLayer.OKPressed)
                {
                    MapLayerData data = frmNewLayer.MapLayerData;

                    charactersToolStripMenuItem.Enabled = true;
                    chestsToolStripMenuItem.Enabled = true;
                    keysToolStripMenuItem.Enabled = true;
                }
            }
        }

        #endregion

        #region Map Display Event Handler Region

        void mapDisplay_MouseUp(object sender, MouseEventArgs e)
        {
        }
        void mapDisplay_MouseDown(object sender, MouseEventArgs e)
        {
        }
        void mapDisplay_MouseMove(object sender, MouseEventArgs e)
        {
        }
        void mapDisplay_MouseLeave(object sender, EventArgs e)
        {
        }
        void mapDisplay_MouseEnter(object sender, EventArgs e)
        {
        }

        #endregion
    }
}
```

There was going to be confusion between some of the classes in the **System.Drawing** name space and the MonoGame Framework so I removed the using statement for **System.Drawing**. I added in a using statement for **System.IO** as we will be doing some file input and output. I also added in using statements for the basic MonoGame framework and the MonoGame framework **Graphics** classes. There is then several qualified using statements. A qualified using statement allows you to swap a fully qualified class with the qualifier. So, when you need to use the **Bitmap** class from **System.Drawing** you can use **GDIBitmap**. You may also hear it called aliasing, giving a class an alternative name.

I added in a few fields that I'm sure we are going to want to work with. There is a **LevelData** field that will hold information about the level currently being edited. I added in a **TileMap** field, **map**, to hold the map. I also have two **List<T>** fields. The first is for the tile sets and the second is the layers of the map.

At the moment there is just the one property, **GraphicsDevice**. I included that because you will need access to the **GraphicsDevice** associated with the **MapDisplay**. You need it to read in content.

The constructor first wires event handlers for the **Load** event of the form and the **FormClosing** event. In the **Load** event I will do a little initialization. In the **FormClosing** event I will be adding checks that if the level has changed the user gets a warning that they haven't saved the level. I then set the **Enabled** property of most of the main menu items to false. I do that because you don't want to add items until there is a level to work with. I then wire the handlers for the **Click** event of the new level, new tile set, and new map layer menu items. I also wire event handlers for the mouse and the map display.

The event handler for the **Load** event for the form wires an event handler for the **Idle** event of the application. The **Idle** event will be fired when your form isn't doing anything, or idle. When it is idle is a good time to let the map display know that it can redraw itself. The **FormClosing** event handler does nothing at the moment but will be used in the future so I added it in now. Next is the event handler for the **Idle** event of the application. All it does is call the **Invalidate** method of the map display letting it know it is time to redraw itself.

The next region of code is for the new menu items. The first is for the new level menu item. Inside of a using statement I create a new instance of **FormNewLevel**. Inside that statement I call the **ShowDialog** method. I then check the **OKPressed** property of the form. If it was true I set the **levelData** field to be the **LevelData** property of **FormNewLevel**. I also set the **Enabled** property of the tileset menu item to true so tilesets can be added.

The handler for the **Click** event for the new tile set menu item is similar. It creates an instance of the form inside of a using statement. Inside that it displays the form using the **ShowDialog** method. It checks to see if the **OKPressed** property of the form is true as well. It is I set a **TilesetData** local variable to the **TilesetData** property. I also set the **Enabled** property of the map layer menu item to true.

The event handler for the **Click** event for a new layer has a similar form as the others. In a using statement I create a new **FormNewLayer**. I pass in the **MapWidth** and **MapHeight** from the **levelData** field to the constructor. Inside the using statement I call the **ShowDialog** method to display the form. I check to see if the **OKPressed** property of the form is true. If so, I create a **MapLayerData** object. I also set the **Enabled** property of the other main menu items to true.

There is a region of code that is dedicated to event handlers related to the **MapDisplay** control. They handle the mouse down, mouse up, move move, mouse enter and mouse leave events.

That is the basic framework for the level editor. The next step will be reading in the image for a tileset so it can be used. To handle that I'm going to extend the event handler for the new tileset menu item. First, you are going to want a list of images you can use for the form. Add the following field and change the **newTilesetToolStripMenuItem_Click** method to the following. I also added a new method to the **TileMap** class called **AddTileset** that will add a new tileset to the map.

```
List<GDIImage> tileSetImages = new List<GDIImage>();

void newTilesetToolStripMenuItem_Click(object sender, EventArgs e)
{
    using (FormNewTileset frmNewTileset = new FormNewTileset())
    {
        frmNewTileset.ShowDialog();
```

```
            if (frmNewTileset.OKPressed)
            {
                TilesetData data = frmNewTileset.TilesetData;

                try
                {
                    GDIImage image = (GDIImage)GDIBitmap.FromFile(data.TilesetImageName);
                    tileSetImages.Add(image);

                    Stream stream = new FileStream(data.TilesetImageName, FileMode.Open,
                    FileAccess.Read);

                    Texture2D texture = Texture2D.FromStream(GraphicsDevice, stream);
                    Tileset tileset = new Tileset(
                        texture,
                        data.TilesWide,
                        data.TilesHigh,
                        data.TileWidthInPixels,
                        data.TileHeightInPixels);

                    tileSets.Add(tileset);

                    if (map != null)
                        map.AddTileset(tileset);

                    stream.Close();
                    stream.Dispose();
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error reading file.\n" + ex.Message, "Error reading image");
                    return;
                }

                lbTileset.Items.Add(data.TilesetName);

                if (lbTileset.SelectedItem == null)
                    lbTileset.SelectedIndex = 0;

                mapLayerToolStripMenuItem.Enabled = true;
            }
        }
    }
```

TileMap Class

```
public void AddTileset(Tileset tileset)
{
    tilesets.Add(tileset);
}
```

So, what is the new code doing. First I used a try-catch block when I try to read in the image for the tileset. Here is a spot where an exception could be thrown if there is a problem reading the the image so it is best to do this in a try-catch block. That way you can recover from an exception. I first try to read in the image in a format that can be assigned to the **Picture Box** control that will preview the tileset, an **Image** from GDI+. To do that I cast the return value of the **FromFile** method of the **Bitmap** class from GDI+ as well. I then add the image to the list of tileset images. I create a **FileStream** using the **TilesetImageName** property from the **TilesetData** object for the name of the file and use

FileMode.Open and **FileAccess.Read** to read the file. If you don't specify you want to read the file you may get an access violation stating the file is being used by another process. I then use the **FromStream** method of the **Texture2D** class to read in the image. I then create a new **Tileset** object and add it to the list of tilesets using the **Texture2D** I just read in and the fields from the **TilesetData** object. If that **map** field is not null I call the new **AddTileset** method to add it to the map. I then close the stream and dispose it. I the catch for the try I display a message box stating there was an error and the error then exit the method. I then add the **TilesetName** property of the **TilesetData** object to **lbTileset**, the **List Box** that holds the names of the tile sets. If the **SelectedItem** of **lbTileset** is null I set the **SelectedIndex** property of **lbTileset** to zero so that the tileset will be selected. I also set the **Enabled** property of map layer menu item to true.

When the user clicks a tileset name in **lbTileset** you are going to want to perform a few actions. The best way to do that is to handle the **SelectedIndexChanged** event of the **List Box**. I wired the event in the **Load** event of the form and placed the handler in the same region. Change the **FormMain_Load** method to the following and add this new region below the **Form Event Handler Region**.

```
        void FormMain_Load(object sender, EventArgs e)
        {
            Application.Idle += new EventHandler(Application_Idle);
            lbTileset.SelectedIndexChanged += new EventHandler(lbTileset_SelectedIndexChanged);
        }

        void lbTileset_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (lbTileset.SelectedItem != null)
            {
                nudCurrentTile.Value = 0;
                nudCurrentTile.Maximum = 
tileSets[lbTileset.SelectedIndex].SourceRectangles.Length - 1;

                FillPreviews();
            }
        }
```

The new event handler for the **Load** event of the form wires a handler for the **SelectedIndexChanged** event of **lbTileset** that will be fired when a different tileset is selected from the list. The new region I added is called **Tile Tab Event Handler Region** and will house event handlers related to the tile tab in the left pane. The event handler makes sure that the **SelectedItem** is not null. If it is not I set the **Value** property of **nudCurrentTile** to 0, the first tile in the tile set. I also set the **Maximum** property to be the number of source rectangles in the tileset minus 1 because the source rectangles are zero based so the last rectangle is the length minus 1. I then call a method I wrote that will fill the **Picture Boxes** in the tab called **FillPreviews**.

The **FillPreviews** does a little GDI+ manipulation to get the selected tile in **nudCurrentTile** into the tile preview **Picture Box**. You first need to know what tileset is selected, the **SelectedIndex** property of **lbTileset**, and which tile is selected, the **Value** property of **nudCurrentTile**. It needs to be cast to an integer because it is a decimal. I first create a **GDIImage** that is the width and height of the preview **Picture Box** for the tile. I need a destination rectangle to draw the image to. I use zero for the **X** and **Y** coordinates and the **Width** and **Height** of the image for the width and height. I also need the source rectangle of the tile in the tile set. I get that using the **SelectedIndex** and the **Value** properties that I captured and the **tileSets** collection. I then create a **Graphics** object using **FromImage** and the preview image that I created. I then using the **DrawImage** method passing in the image from **tileSetImages**, the destination rectangle, source rectangle, and **GDIGraphicsUnit.Pixel**. I then set the **Image** properties of the **Picture Boxes**.

```
void FormMain_Load(object sender, EventArgs e)
{
    Application.Idle += new EventHandler(Application_Idle);
    lbTileset.SelectedIndexChanged += new EventHandler(lbTileset_SelectedIndexChanged);
}

#region Tile Tab Event Handler Region

void lbTileset_SelectedIndexChanged(object sender, EventArgs e)
{
    if (lbTileset.SelectedItem != null)
    {
        nudCurrentTile.Value = 0;
        nudCurrentTile.Maximum = tileSets[lbTileset.SelectedIndex].SourceRectangles.Length - 1;

        FillPreviews();
    }
}

private void FillPreviews()
{
    int selected = lbTileset.SelectedIndex;
    int tile = (int)nudCurrentTile.Value;

    GDIImage preview = (GDIImage)new GDIBitmap(pbTilePreview.Width, pbTilePreview.Height);

    GDIRectangle dest = new GDIRectangle(0, 0, preview.Width, preview.Height);
    GDIRectangle source = new GDIRectangle(
        tileSets[selected].SourceRectangles[tile].X,
        tileSets[selected].SourceRectangles[tile].Y,
        tileSets[selected].SourceRectangles[tile].Width,
        tileSets[selected].SourceRectangles[tile].Height);

    GDIGraphics g = GDIGraphics.FromImage(preview);

    g.DrawImage(tileSetImages[selected], dest, source, GDIGraphicsUnit.Pixel);

    pbTilesetPreview.Image = tileSetImages[selected];
    pbTilePreview.Image = preview;
}

#endregion
```

You are also going to want to change the tile preview if the **Value** property of **nudCurrentTile** changes. I will wire that in the **Load** event handler of the form as well. Change that method to the following and add this handler to the **Tile Tab Event Handler Region**.

```
void FormMain_Load(object sender, EventArgs e)
{
    Application.Idle += new EventHandler(Application_Idle);
    lbTileset.SelectedIndexChanged += new EventHandler(lbTileset_SelectedIndexChanged);
    nudCurrentTile.ValueChanged += new EventHandler(nudCurrentTile_ValueChanged);
}

void nudCurrentTile_ValueChanged(object sender, EventArgs e)
{
    if (lbTileset.SelectedItem != null)
    {
        FillPreviews();
    }
}
```

The new handler checks to see if the **SelectedItem** property is not null. If it is null you don't want a preview. If it is not null I call the **FillPreviews** method to update the previews.

The next step it to handle creating a new layer. That takes place in the handler for the new layer menu item. Change that handler to the following.

```
void newLayerToolStripMenuItem_Click(object sender, EventArgs e)
{
    using (FormNewLayer frmNewLayer = new FormNewLayer(levelData.MapWidth,
levelData.MapHeight))
    {
        frmNewLayer.ShowDialog();
        if (frmNewLayer.OKPressed)
        {
            MapLayerData data = frmNewLayer.MapLayerData;

            if (clbLayers.Items.Contains(data.MapLayerName))
            {
                MessageBox.Show("Layer with name " + data.MapLayerName + " exists.", "Existing
layer");
                return;
            }

            MapLayer layer = MapLayer.FromMapLayerData(data);

            clbLayers.Items.Add(data.MapLayerName, true);
            clbLayers.SelectedIndex = clbLayers.Items.Count - 1;
            layers.Add(layer);

            if (map == null)
                map = new TileMap(tileSets, layers);

            charactersToolStripMenuItem.Enabled = true;
            chestsToolStripMenuItem.Enabled = true;
            keysToolStripMenuItem.Enabled = true;
        }
    }
}
```

What the new code is doing is checking to see if an entry in **clbLayers**, the **Checked List Box**, with the name **data.MapLayername** already exists. If it does I display an error message that a layer with that name already exists and exit the method. Otherwise I use the **FromMapLayerData** method I added to the **MapLayer** class to create a new layer and add it to the **List<MapLayer>**. I also add the **MapLayerName** to **clbLayers**. When adding the item I have it set to be checked initially, and thus drawn by default. I also set the **SelectedIndex** property of **clbLayers** to be the last layer that was added. If the **map** field is null I create a new **TileMap** using the **List<Tileset>** and **List<MapLayer>**.

This is where you can see what the dangers of passing reference types around. The **map** field now has references to the **tileSets** and **layers** fields of the form. Changing one of them changes the other. I had originally included an else to the if statement that checked to see if **map** was null where I'd the new layer to **map** and suddenly there were 3 layers in the **layers** field, not 2. So, be careful when you are passing reference types around that this sort of thing does not happen.

The next step will be to actually draw the layers. For that we need two more things, a **Camera** and an **Engine**. Add the following fields to the **Fields** region of **FormMain**. Also, change the **Load** event handler of the form to the following to actually create the camera and the engine.

```
public static Camera camera;

Engine engine;
        void FormMain_Load(object sender, EventArgs e)
        {
            Application.Idle += new EventHandler(Application_Idle);

            lbTileset.SelectedIndexChanged += new EventHandler(lbTileset_SelectedIndexChanged);
            nudCurrentTile.ValueChanged += new EventHandler(nudCurrentTile_ValueChanged);

            Rectangle viewPort = new Rectangle(0, 0, mapDisplay.Width, mapDisplay.Height);

            camera = new Camera(viewPort);
            engine = new Engine(32, 32);
        }
```

What I did was create a **Rectangle** that describes the **MapDisplay** control because the **Camera** class needs a rectangle that describes the view port it is using. I then create an **Engine** instance with pixel width and height of 32 pixels. Something I will be handling later, possibly not in this tutorial, is what to do if the size of the **MapDisplay** changes and how to recover from that. I will also add in options that you can set to change the width and height of the tiles on the screen.

So, the next step is to start drawing. That will take place in the **Draw** method of the **MapDisplay** class. Change that **Draw** method to the following.

```
protected override void Draw()
{
    base.Draw();

    if (FormMain.map == null)
        return;

    Editor.spriteBatch.Begin(
        SpriteSortMode.Deferred,
        BlendState.AlphaBlend,
        SamplerState.PointClamp,
        null,
        null,
        null,
        FormMain.camera.Transformation);
    for (int i = 0; i < FormMain.layers.Count; i++)
        FormMain.layers[i].Draw(Editor.spriteBatch, FormMain.camera, FormMain.tileSets);

    Editor.spriteBatch.End();
}
```

I first check to see if the **map** field of **FormMain** is null. If it is null I exit the method. I then call the **Begin** method of **SpriteBatch** passing in parameters like I did in the game. It is a property of the **Editor** property from the control we inherited from. Next I loop over the number of layers in a for loop. I use a for loop instead of a foreach loop because eventually I will want to get the check state of the item in the **clbLayers** checked list box. I call the **Draw** method of the layer in the list of layers passing in the **SpriteBatch** object, the **Camera** object and the **List<Tileset>** for the tile sets. I then call the **End** method of **SpriteBatch**.

We need the **Draw** method of a **MapLayer** that takes those three parameters. Add the following **Draw** method to the **MapLayer** class.

```
public void Draw(SpriteBatch spriteBatch, Camera camera, List<Tileset> tilesets)
{
    Point cameraPoint = Engine.VectorToCell(camera.Position * (1 / camera.Zoom));
    Point viewPoint = Engine.VectorToCell(
        new Vector2(
            (camera.Position.X + camera.ViewportRectangle.Width) * (1 / camera.Zoom),
            (camera.Position.Y + camera.ViewportRectangle.Height) * (1 / camera.Zoom)));

    Point min = new Point();
    Point max = new Point();

    min.X = Math.Max(0, cameraPoint.X - 1);
    min.Y = Math.Max(0, cameraPoint.Y - 1);
    max.X = Math.Min(viewPoint.X + 1, Width);
    max.Y = Math.Min(viewPoint.Y + 1, Height);

    Rectangle destination = new Rectangle(0, 0, Engine.TileWidth, Engine.TileHeight);
    Tile tile;

    for (int y = min.Y; y < max.Y; y++)
    {
        destination.Y = y * Engine.TileHeight;

        for (int x = min.X; x < max.X; x++)
        {
            tile = GetTile(x, y);

            if (tile.TileIndex == -1 || tile.Tileset == -1)
                continue;

            destination.X = x * Engine.TileWidth;

            spriteBatch.Draw(
                tilesets[tile.Tileset].Texture,
                destination,
                tilesets[tile.Tileset].SourceRectangles[tile.TileIndex],
                Color.White);
        }
    }
}
```

The next thing I want to do is get the map scrolling and adding in tiles with the editor. My computer was scrolling my maps really fast so I ended up adding a **Timer** control to the form. Since it was part of the form when I copied the designer code it is already on the form, we just need to activate it.

I added the handler to the **Form Event Handler Region**. Change the **FormMain_Load** method to the following and add this handler to the region. I also removed the handler for **Application.Idle**. I will call the **Invalidate** method from the handler for the timer's tick event.

```
void FormMain_Load(object sender, EventArgs e)
{
    lbTileset.SelectedIndexChanged += new EventHandler(lbTileset_SelectedIndexChanged);
    nudCurrentTile.ValueChanged += new EventHandler(nudCurrentTile_ValueChanged);

    Rectangle viewPort = new Rectangle(0, 0, mapDisplay.Width, mapDisplay.Height);

    camera = new Camera(viewPort);
    engine = new Engine(32, 32);

    controlTimer.Tick += new EventHandler(controlTimer_Tick);
    controlTimer.Enabled = true;
    controlTimer.Interval = 200;
}

void controlTimer_Tick(object sender, EventArgs e)
{
    mapDisplay.Invalidate();
    Logic();
}
```

So, all I did was wire the **Tick** event handler and set the **Enabled** property to true so that the event will be fired and I set the **Interval** value to 200 milliseconds, a fifth of a second. You may want to make the **Interval** smaller if your logic is sluggish or higher if it is too fast. Even where I set it my map scrolled rather quickly. Part of the reason being that I'm scrolling the map 1 tile at a time. The handler for the **Tick** event calls **Invalidate** on **mapDisplay** to redraw the display and it calls the **Logic** method.
For handling the logic of the editor I needed to add a few fields. A **Point** field for the position of the mouse over the **MapDisplay** and two bool fields that determines if the left mouse button is down and the other determines if we are interested in tracking the mouse. Add the following fields to **FormMain**.

```
Point mouse = new Point();
bool isMouseDown = false;
bool trackMouse = false;
```

Now I'm going to change the code for the mouse event handlers for the **MapDisplay**. Change them to the following.

```
        void mapDisplay_MouseUp(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
                isMouseDown = false;
        }

        void mapDisplay_MouseDown(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
                isMouseDown = true;
        }

        void mapDisplay_MouseMove(object sender, MouseEventArgs e)
        {
            mouse.X = e.X;
            mouse.Y = e.Y;
        }

        void mapDisplay_MouseLeave(object sender, EventArgs e)
        {
            trackMouse = false;
        }

        void mapDisplay_MouseEnter(object sender, EventArgs e)
        {
            trackMouse = true;
        }
```

The handler for **MouseUp** checks to see if the left mouse button triggered the event. If it did I set **isMouseDown** to false. The **MouseDown** handler works in reverse. The **MouseMove** handler sets the **X** and **Y** properties of **mouse** to the **X** and **Y** properties of the mouse. The handler for **MouseLeave** sets **trackMouse** to false because the mouse has moved outside of the **MapDisplay** and we aren't interested in processing it. The **MouseEnter** handler does the reverse, sets **trackMouse** to true because we are now interested in the mouse again.

Before I get to the logic of scrolling the map and drawing tiles I need to make two changes to the **Camera** class. I need to make the set part of the **Position** property public. I also need to make the **LockCamera** method public. Change the **Position** property and **LockCamera** method to the following.

```
public Vector2 Position
{
    get { return position; }
    set { position = value; }
}

public void LockCamera()
{
    position.X = MathHelper.Clamp(position.X,
        0,
        TileMap.WidthInPixels * zoom - viewportRectangle.Width);

    position.Y = MathHelper.Clamp(position.Y,
        0,
        TileMap.HeightInPixels * zoom - viewportRectangle.Height);
}
```

The next thing to do is to add the **Logic** method as that is where I will be handling scrolling the map and drawing the map. Add the following method.

```
private void Logic()
{
    if (layers.Count == 0)
        return;

    Vector2 position = camera.Position;

    if (trackMouse)
    {
        if (mouse.X < Engine.TileWidth)
            position.X -= Engine.TileWidth;

        if (mouse.X > mapDisplay.Width - Engine.TileWidth)
            position.X += Engine.TileWidth;

        if (mouse.Y < Engine.TileHeight)
            position.Y -= Engine.TileHeight;

        if (mouse.Y > mapDisplay.Height - Engine.TileHeight)
            position.Y += Engine.TileHeight;

        camera.Position = position;
        camera.LockCamera();

        position.X = mouse.X + camera.Position.X;
        position.Y = mouse.Y + camera.Position.Y;

        Point tile = Engine.VectorToCell(position);

        if (isMouseDown)
        {
            if (rbDraw.Checked)
            {
                layers[clbLayers.SelectedIndex].SetTile(
                    tile.X,
                    tile.Y,
                    (int)nudCurrentTile.Value,
                    lbTileset.SelectedIndex);
            }

            if (rbErase.Checked)
            {
                layers[clbLayers.SelectedIndex].SetTile(
                    tile.X,
                    tile.Y,
                    -1,
                    -1);
            }
        }
    }
}
```

The first thing I do is check to see if the **Count** property of the **layers** field is zero. If it is you don't want to try and edit anything so I exit the method. I then save the **Position** property of the camera in a local variable **position**. I then check the **trackMouse** property. If it is true the mouse is in the map display. I check to see if the **X** value of the mouse's position is less than the width of a tile on the screen. If it is I subtract the width of a tile from the position of the camera, scrolling the map left. Then if the **X** value of the mouse's position is greater than or equal to the width of the map display minus the width of a tile I scroll the map one tile to the right. I do something similar for the **Y** component of the mouse's position. If it is less than the height of tile I scroll up and if it is greater than the height of the display minus a tile I scroll the map down. Since **Position** is a property in the **Camera** class and a structure you can't modify its **X** and **Y** value directly so I set the **Position** property of the camera to the **position** variable and call the **LockCamera** method to lock the camera. I then set the **position** variable to the position of the mouse plus the position of the camera. This tells us which tile the mouse is in on the map. I capture what tile the mouse is in using the **VectorToCell** method of the **Engine** class. I then check if the **isMouseDown** field is true. If it is I check if **rbDraw**'s **Checked** property is true. If it is true you want to draw the tile. I call the **SetTile** method of the **MapLayer** class that takes the x and y coordinates of the tile and the tile index and tileset the tile belongs to. Similarly if **rbErase**'s **Checked** property is true you want to erase the tile. That is done by setting its tile index and tileset to -1.

So, we have a working basic level editor. I'm going to stop this tutorial here though as I think you've had more than enough to digest. In a future tutorial I'm going to add writing out and reading in maps and adding some more options into the editor. The plan for this tutorial was to get started with the level editor and we are well under way.

That's it for this tutorial. It covers the original tutorial with many modifications. So, I encourage you to visit my blog at, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials and other goodness.

Good luck in your Game Programming Adventures!

Cynthia