# Eyes of the Dragon Tutorials
# Part 18
# Finding Loot Part 2

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.

In the last tutorial I worked on adding chests for the player to interact with to find loot. In this tutorial I am going to continue on with that. The first thing I want to do is to add in two classes for keys for unlocking locks. Like you can find on chests, doors, and other objects. Keys are just another type of item so they belong in the **ItemClasses** folder of the **RpgLibrary**. Like other types of items you will want a data class for use in editors and reading in data and a class for the keys themselves that inherits from the **BaseItem** class. Right click the **ItemClasses** folder, select **Add** and then **Class**. Name this new class **KeyData**. Repeat the process and name the new class **Key**. The code for both of those classes follows next.

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

namespace RpgLibrary.ItemClasses
{
    public class KeyData
    {
        #region Field Region

        public string Name;
        public string Type;

        #endregion

        #region Constructor Region

        public KeyData()
        {
        }

        #endregion

        #region Method Region

        public override string ToString()
        {
            string toString = Name + ", ";
            toString += Type;
            return toString;
        }

        #endregion
    }
```

```
}

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

namespace RpgLibrary.ItemClasses
{
    public class Key : BaseItem
    {
        #region Field region
        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public Key(string name, string type)
            : base(name, type, 0, 0, null)
        {
        }

        #endregion

        #region Virtual Method Region

        public override object Clone()
        {
            Key key = new Key(this.Name, this.Type);
            return key;
        }

        #endregion
    }
}
```

Not much you haven't seen before. The **KeyData** class has two public fields. The **Name** of the key and the **Type** of the key. I debated about including other fields like a price and a weight. I didn't think they were necessary. The **Type** field can be of great use. If you've played the **Fable** games they have special silver keys that can be used to open special chests. Each chest requires a certain number of these keys. Using the **Type** field you can easily add this functionality into your games. It will require a minor tweak that I will make later. The **ToString** method of the **KeyData** class just combines the name of the key and the type. The **Key** class is also simplistic. It inherits from the **BaseItem** class. It takes just two string parameters for the name and the type. In the call to the base constructor I pass in the parameters passed to the **Key** class and 0 for the price and weight and null for **allowableClasses**. The **Clone** method just creates a new key and returns it.

The first tweak for handling multiple keys of the same type is to add in two fields to the **ChestData** class. They are **KeyType** which is a string and **KeysRequired** which is an integer. I'm going to also associate a **DifficultyLevel** with a chest from the **SkillClasses** folder for the lock on the chest. In the **ToString** method you add in these three new fields, calling their **ToString** methods. I also removed the **TextureName** field from the class. I will be handling associating an image with a chest when I get to

the level editor as it applies more to levels. Change the **ChestData** method to the following.

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

namespace RpgLibrary.ItemClasses
{
    public class ChestData
    {

        public string Name;
        public DifficultyLevel DifficultyLevel;
        public bool IsLocked;
        public bool IsTrapped;
        public string TrapName;
        public string KeyName;
        public string KeyType;
        public int KeysRequired;
        public int MinGold;
        public int MaxGold;
        public Dictionary<string, string> ItemCollection;

        public ChestData()
        {
            ItemCollection = new Dictionary<string, string>();
        }

        public override string ToString()
        {
            string toString = Name + ", ";

            toString += DifficultyLevel.ToString() + ", ";
            toString += IsLocked.ToString() + ", ";
            toString += IsTrapped.ToString() + ", ";
            toString += TrapName + ", ";
            toString += KeyName + ", ";
            toString += KeyType + ", ";
            toString += KeysRequired.ToString() + ", ";
            toString += MinGold.ToString() + ", ";
            toString += MaxGold.ToString();

            foreach (KeyValuePair<string, string> pair in ItemCollection)
            {
                toString += ", " + pair.Key + "+" + pair.Value;
            }

            return toString;
        }
    }
}
```

You also need update the **Chest** class, specifically the **Clone** method. Change the **Clone** of the **Chest** class to the following.

```
        public override object Clone()
```

```
        {
            ChestData data = new ChestData();

            data.Name = chestData.Name;
            data.DifficultyLevel = chestData.DifficultyLevel;
            data.IsLocked = chestData.IsLocked;
            data.IsTrapped = chestData.IsTrapped;
            data.TrapName = chestData.TrapName;
            data.KeyName = chestData.KeyName;
            data.KeyType = chestData.KeyType;
            data.KeysRequired = chestData.KeysRequired;
            data.MinGold = chestData.MinGold;
            data.MaxGold = chestData.MaxGold;

            foreach (KeyValuePair<string, string> pair in chestData.ItemCollection)
                data.ItemCollection.Add(pair.Key, pair.Value);

            Chest chest = new Chest(data);

            return chest;
        }
```

I will handle opening chests with keys in a future tutorial. I just wanted to add in the functionality to this tutorial so it will be available later when it is needed. I'm going to work on the editor a bit on creating keys and chests. To do that I'm first going to update the **ItemDataManager** class to hold dictionaries of **<string, ItemDataType>** where **ItemDataType** is the data class associated with the item. Change the **ItemDataManager** to the following.

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

namespace RpgLibrary.ItemClasses
{
    public class ItemDataManager
    {
        #region Field Region

        readonly Dictionary<string, ArmorData> armorData = new
            Dictionary<string, ArmorData>();

        readonly Dictionary<string, ShieldData> shieldData = new
            Dictionary<string, ShieldData>();

        readonly Dictionary<string, WeaponData> weaponData = new
            Dictionary<string, WeaponData>();

        readonly Dictionary<string, ReagentData> reagentData = new
            Dictionary<string, ReagentData>();

        readonly Dictionary<string, KeyData> keyData = new
            Dictionary<string, KeyData>();

        readonly Dictionary<string, ChestData> chestData = new
            Dictionary<string, ChestData>();

        #endregion
```

```
        #region Property Region

        public Dictionary<string, ArmorData> ArmorData
        {
            get { return armorData; }
        }

        public Dictionary<string, ShieldData> ShieldData
        {
            get { return shieldData; }
        }

        public Dictionary<string, WeaponData> WeaponData
        {
            get { return weaponData; }
        }

        public Dictionary<string, ReagentData> ReagentData
        {
            get { return reagentData; }
        }

        public Dictionary<string, KeyData> KeyData
        {
            get { return keyData; }
        }

        public Dictionary<string, ChestData> ChestData
        {
            get { return chestData; }
        }

        #endregion

        #region Constructor Region
        #endregion

        #region Method Region
        #endregion
    }
}
```

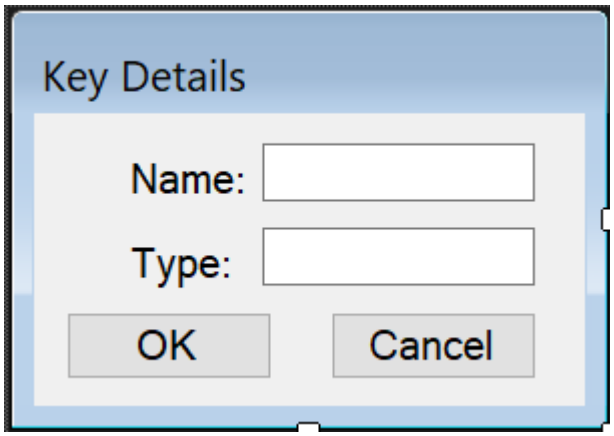
Nothing there that you haven't seen before. Build your project to make sure it builds correctly. Right click the **RpgEditor** project in the solution explorer, select **Add** and then **Windows Form**. Name this new form **FormKey**. Set the **Size** property of the form to be the **Size** property of your **FormDetails**. Set the **MinimizeBox** property to false and the **Text** property to **Keys**. You will now want to have **FormKey** inherit from **FormDetails** instead of **Form**. Right click **FormKey** 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.Threading.Tasks;
```

```
using System.Windows.Forms;

namespace RpgEditor
{
    public partial class FormKey : FormDetails
    {
        public FormKey()
        {
            InitializeComponent();
        }
    }
}
```

I'm going to add another form before I get to the code for **FormKey**. Right click **RpgEditor** in the solution explorer, select **Add** and then **Windows Form**. Name this new form **FormKeyDetails**. I'm going to add a couple controls and set some properties for **FormKeyDetails**. Your finished form in the designer should resemble the one below.




Start by changing the size of the form so it is a bigger than what you need. Drag a **Label** onto the form near the top then a **Text Box** and position it to the right of the **Label**. Drag another **Label** onto the form positioning it below the first. Drag a second **Text Box** onto the form and position it below the first **Text Box**. Now drag two **Buttons** onto the form. Position the first to the left and the second to the right. Set the **Text** property of the form to **Key Details**, the **StartPosition** to **CenterParent**, the **ControlBox** to **False**, and the **FormBorderStyle** to **FixedDialog**. Set the **Name** property of the first **Text Box** to **tbName** and the second to **tbType**. Set the **Text** property of the first **Label** to **Name:** and the second to **Type:**. For the buttons the one of the left has **Name** and **Text** properties of **btnOK** and **OK**. The second**btnCancel** and **Cancel** for the **Name** and **Text** properties.

Alternatively, you can copy and paste this code into the FormKeyDetails.Designer.cs file.

```
namespace RpgEditor
{
    partial class FormKeyDetails
    {
        /// <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.label1 = new System.Windows.Forms.Label();
            this.tbName = new System.Windows.Forms.TextBox();
            this.label2 = new System.Windows.Forms.Label();
            this.tbType = new System.Windows.Forms.TextBox();
            this.btnOK = new System.Windows.Forms.Button();
            this.btnCancel = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(31, 13);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(49, 17);
            this.label1.TabIndex = 0;
            this.label1.Text = "Name:";
            //
            // tbName
            //
            this.tbName.Location = new System.Drawing.Point(83, 10);
            this.tbName.Name = "tbName";
            this.tbName.Size = new System.Drawing.Size(100, 22);
            this.tbName.TabIndex = 1;
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Location = new System.Drawing.Point(31, 41);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(44, 17);
            this.label2.TabIndex = 2;
            this.label2.Text = "Type:";
            //
            // tbType
            //
            this.tbType.Location = new System.Drawing.Point(83, 38);
            this.tbType.Name = "tbType";
            this.tbType.Size = new System.Drawing.Size(100, 22);
```

```
            this.tbType.TabIndex = 3;
            //
            // btnOK
            //
            this.btnOK.Location = new System.Drawing.Point(12, 66);
            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(108, 66);
            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;
            //
            // FormKeyDetails
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(200, 97);
            this.ControlBox = false;
            this.Controls.Add(this.btnCancel);
            this.Controls.Add(this.btnOK);
            this.Controls.Add(this.tbType);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.tbName);
            this.Controls.Add(this.label1);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Name = "FormKeyDetails";
            this.StartPosition = System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "Key Details";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox tbName;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.TextBox tbType;
        private System.Windows.Forms.Button btnOK;
        private System.Windows.Forms.Button btnCancel;
    }
}
```

Now lets add the logic to **FormKeyDetails**. Right click **FormKeyDetails** 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.ItemClasses;

namespace RpgEditor
{
    public partial class FormKeyDetails : Form
    {
        #region Field Region

        KeyData key;

        #endregion

        #region Property Region

        public KeyData Key
        {
            get { return key; }
            set { key = value; }
        }

        #endregion

        #region Constructor Region

        public FormKeyDetails()
        {
            InitializeComponent();

            this.Load += new EventHandler(FormKeyDetails_Load);
            this.FormClosing += new FormClosingEventHandler(FormKeyDetails_FormClosing);

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

        #endregion

        #region Event Handler Region

        void FormKeyDetails_Load(object sender, EventArgs e)
        {
            if (key != null)
            {
                tbName.Text = key.Name;
                tbType.Text = key.Type;
            }
        }

        void FormKeyDetails_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (e.CloseReason == CloseReason.UserClosing)
            {
                e.Cancel = true;
            }
        }
```

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

            key = new KeyData();

            key.Name = tbName.Text;
            key.Type = tbType.Text;

            this.FormClosing -= FormKeyDetails_FormClosing;
            this.Close();
        }

        void btnCancel_Click(object sender, EventArgs e)
        {
            key = null;

            this.FormClosing -= FormKeyDetails_FormClosing;
            this.Close();
        }

        #endregion
    }
}
```

This should look familiar to you. It follows the other detail forms that I created. There is a using statement to bring the **ItemClasses** name space from the **RpgLibrary** into scope. There is a field of type **KeyData** for the key that is being created or edited. If the key is being edited you can use the property to set the fields of the key. The constructor wires handlers for the **Load** and **FormClosing** events of the form. It also wires handlers for the **Click** event of **btnOK** and **btnCancel**.

The event handler for the **Load** event of the form checks to see if the field **key** is not null. If it isn't it sets the **Text** properties of **tbName** and **tbType** to be the **Name** and **Type** fields of **key**. The event handler for **FormClosing** is the same as before. If the reason for closing the form is the user is trying to close the form the event is cancelled.

In the **Click** event handler for **btnOK** I check to see if the **Text** property of **tbName** is null or empty. If it is I display a message box stating that you need to give the key a name and then exit the method. I'm not enforcing that a key have a type associated with it. If you want to add the functionality in you can do the same as I did for **tbName** with **tbType**. If **tbName** had a value I assign the **key** field a new instance of **KeyData**. I then set the **Name** and **Type** fields to be the **Text** property of **tbName** and **tbType** respectively. To allow the form to close I unsubscribe the **FormClosing** event handler and call the **Close** method to close the form.

The **Click** event handler for **btnCancel** sets the **key** field to null. It then unsubscribes the **FormClosing** event and calls the **Close** method of the form. Just like in the other forms for a specific type of item. Time to code the logic for **FormKey**. Right click **FormKey** in the solution explorer and select **View Code**. Change the code for **FormKey** 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 System.IO;
using RpgLibrary.CharacterClasses;
using RpgLibrary.ItemClasses;

namespace RpgEditor
{
    public partial class FormKey : FormDetails
    {
        #region Field Region
        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public FormKey()
        {
            InitializeComponent();

            btnAdd.Click += new EventHandler(btnAdd_Click);
            btnEdit.Click += new EventHandler(btnEdit_Click);
            btnDelete.Click += new EventHandler(btnDelete_Click);
        }

        #endregion

        #region Event Handler Region

        void btnAdd_Click(object sender, EventArgs e)
        {
            using (FormKeyDetails frmKeyDetails = new FormKeyDetails())
            {
                frmKeyDetails.ShowDialog();

                if (frmKeyDetails.Key != null)
                {
                    AddKey(frmKeyDetails.Key);
                }
            }
        }
        void btnEdit_Click(object sender, EventArgs e)
        {
            if (lbDetails.SelectedItem != null)
            {
                string detail = lbDetails.SelectedItem.ToString();
                string[] parts = detail.Split(',');
                string entity = parts[0].Trim();

                KeyData data = itemManager.KeyData[entity];
                KeyData newData = null;
```

```
        using (FormKeyDetails frmKeyData = new FormKeyDetails())
        {
            frmKeyData.Key = data;
            frmKeyData.ShowDialog();

            if (frmKeyData.Key == null)
                return;

            if (frmKeyData.Key.Name == entity)
            {
                itemManager.KeyData[entity] = frmKeyData.Key;
                FillListBox();
                return;
            }

            newData = frmKeyData.Key;
        }

        DialogResult result = MessageBox.Show(
            "Name has changed. Do you want to add a new entry?",
            "New Entry",
            MessageBoxButtons.YesNo);

        if (result == DialogResult.No)
            return;

        if (itemManager.KeyData.ContainsKey(newData.Name))
        {
            MessageBox.Show("Entry already exists. Use Edit to modify the entry.");
            return;
        }

        lbDetails.Items.Add(newData);
        itemManager.KeyData.Add(newData.Name, newData);
    }
}
void btnDelete_Click(object sender, EventArgs e)
{
    if (lbDetails.SelectedItem != null)
    {
        string detail = (string)lbDetails.SelectedItem;
        string[] parts = detail.Split(',');
        string entity = parts[0].Trim();

        DialogResult result = MessageBox.Show(
            "Are you sure you want to delete " + entity + "?",
            "Delete",
            MessageBoxButtons.YesNo);

        if (result == DialogResult.Yes)
        {
            lbDetails.Items.RemoveAt(lbDetails.SelectedIndex);
            itemManager.KeyData.Remove(entity);

            if (File.Exists(FormMain.ItemPath + @"\Key\" + entity + ".xml"))
                File.Delete(FormMain.ItemPath + @"\Key\" + entity + ".xml");
        }
    }
}
#endregion
```

```
        #region Method Region

        public void FillListBox()
        {
            lbDetails.Items.Clear();

            foreach (string s in FormDetails.ItemManager.KeyData.Keys)
                lbDetails.Items.Add(FormDetails.ItemManager.KeyData[s]);
        }

        private void AddKey(KeyData keyData)
        {
            if (FormDetails.ItemManager.KeyData.ContainsKey(keyData.Name))
            {
                DialogResult result = MessageBox.Show(
                    keyData.Name + " already exists. Overwrite it?",
                    "Existing key",
                    MessageBoxButtons.YesNo);

                if (result == DialogResult.No)
                    return;

                itemManager.KeyData[keyData.Name] = keyData;

                FillListBox();

                return;
            }

            itemManager.KeyData.Add(keyData.Name, keyData);
            lbDetails.Items.Add(keyData);
        }

        #endregion
    }
}
```

Again, this should look familiar as other forms use the same code. The difference is that instead of working with armor, shields, or weapons, I'm working with keys. There is the using statement for the **System.IO** name space. Event handlers for the click event of the buttons are wired in the constructors. The **Click** event handler of **btnAdd** creates a form in a using block. I show the form. If the **Key** property of the form is not null I call the **AddKey** method passing in the **Key** property. The **Click** event handler of **btnEdit** checks to see if the **SelectedItem** of **lbDetails** is not null. It parses the string to get the name of the key. It then gets the **KeyData** for the selected item and sets **newData** to null. In a using statement a form is created. The **Key** property of the form is set to the **KeyData** of **SelectedItem.** I call the **ShowDialog** method to display the form. If the **Key** property of form is null I exit the method. If the name is the same as before I assign the entry in the item manager to be the new key, call **FillListBox** to update the key and exit the method. I then set **newData** to be the **Key** property of the form. The name of the key changed so I display a message box asking if the new key should be added. If the result is no I exit the method. If there is a key with that name already I display a message box and exit. If there wasn't I add the new key to the list box and the item manager. The **Click** event handler for **btnDelete** checks to make sure that the **SelectedItem** of the list box is not null. It parses the selected item and displays a message box asking if the key should be deleted. If the result of the message box is Yes I remove the key from the list box and I remove if from the item manager as well. I

then delete the file, if it exists.

I noticed something in my code that really should be fixed. In the code of **FormWeapon** I have as the class name **Weapons** instead of **FormWeapon**. If you have that problem as well this is a good time to fix it. Right click **FormWeapon** in the solution explorer and select **View Code**. Place your cursor over **Weapons** and press **F2** to rename it. In the dialog box that pops up replace **Weapons** with **FormWeapon** and press OK. In the second dialog box that pops up just select **OK**.

Before I get to adding these forms to the editor I want to add in forms that work with **ChestData**. Right click **RpgEditor** in the solution explorer, select **Add** and then **Windows Form**. Name this new form **FormChest**. Set the **Size** property of the form to be the **Size** property of your **FormDetails**. Set the **MinimizeBox** property to false and the **Text** property to **Chests**. You will now want to have **FormChest** inherit from **FormDetails** instead of **Form**. Right click **FormChest** 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;

namespace RpgEditor
{
    public partial class FormChest : FormDetails
    {
        public FormChest()
        {
            InitializeComponent();
        }
    }
}
```

You will also want a form for creating chests. Right click the **RpgEditor** project in the solution explorer, select **Add** and then **Windows Form**. Name this new form **FormChestDetails**. My finished form in the designer is next.

I set a few properties for the form. I set **ControlBox** to **False**, **FormBorderStyle** to **FixedDialog** and **Text** to **Chest**. Set **StartPosition** to **CenterParent** as well.

There are a lot of controls on the form and I tried to group them logically. First, make your form a lot bigger to house all of the controls. The first control I dragged onto the form was a **Label** and set its **Text** property to **Chest Name:**. I then dragged a **Text Box** beside the **Label** and set its **Name** property to **tbName**. I also made the **Text Box** a little wider. Below the **Text Box** I dragged a **Group Box** to group the items related to the lock on the chest together. I set that **Group Box**'s **Text** property to **Lock Properties**. I dragged a **Check Box** onto that **Group Box** and set its **Name** to **cbLock** and its **Text** to **Locked**. Under the **Check Box** I dragged a **Label** and **Combo Box**. Set the **Label**'s **Text** property to **Lock Difficulty:** and the **Combo Box**'s **Name** property to **cboDifficulty**. I then dragged a **Label** and

set its **Text** property to **Key Name:** and a **Text Box** beside that and set its **Name** property to **tbKeyName**. I dragged another **Label** and **Text Box** under those two. The **Label**'s **Text** property was set to **Key Type:** and the **Text Box**'s **Name** was set to **tbKeyType**. I then dragged a **Label** and **Numeric Up Down** onto the **Group Box**. I set the **Text** property of the **Label** to **Keys Needed:** and the **Name** property of the **Numeric Up Down** to **nudKeys**. I also set the **Enabled** property of the **Combo Box**, **Text Boxes**, and **Numeric Up Down** to **False** initially. The will be enabled if the user checks the **Check Box** to be checked.

I dragged a second **Group Box** onto the form below the **Lock Properties Group Box** and set the width to be the same width. I set the **Text** property of the second **Group Box** to **Trap Properties**. I dragged a **Check Box** onto that **Group Box** and set its **Name** property to be **cbTrap**. I set then dragged a **Label** and **Text Box** onto the second **Group Box**. I set the **Text** property of the **Label** to **Trap Name:** and the **Name** property of the **Text Box** to **tbTrap**. I set the **Enabled** property of **tbTrap** to **False** as well.
I then dragged a third **Group Box** onto the form and sized the width to be the same as the other two. I set its **Text** property to **Gold Properties**. I dragged a **Label** and **Numeric Up Down** onto this **Group Box**. I set the **Text Property** of the **Label** to **Minimum Gold:** and the **Name** property of the **Numeric Up Down** to **nudMinGold**. I dragged another **Label** and **Numeric Up Down** onto this **Group Box**. I set the **Text** property of the **Label** to **Maximum Gold:** and the **Name** property of the **Numeric Up Down** to **nudMaxGold**. I set the **Maximum** property of **nudMinGold** and **nudMaxGold** to be 10000. You will want to tweak this number as you test your game to make sure that it is not unbalanced. If the player accumulates too much gold and can buy really expensive and powerful items then your balance goes out the window. The same is true if you give the player too little gold. They won't be able to defend themselves against more powerful monsters.

I then dragged on another **Group Box** and set its **Text** property to **Item Properties**. I dragged a **List Box** onto the **Group Box** and made it wider and taller. I set its **Name** property to **lbItems**. I then dragged two **Buttons** below the **List Box**. The one on the left I named **btnAdd** and set its **Text** property to **Add**. The one on the right I named **btnRemove** and set its **Text** property to **Remove**.
The last two controls I dragged onto the form were two **Buttons**. The first button, the one on the left, I set its **Name** to **btnOK** and its **Text** to **OK**. The second I set its **Name** property to **btnCancel** and its **Text** property to **Cancel**.

Alternatively, you can add this code to the **FormChestDetails.Designer.cs** file.

```
namespace RpgEditor
{
    partial class FormChestDetails
    {
        /// <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.tbName = new System.Windows.Forms.TextBox();
        this.label1 = new System.Windows.Forms.Label();
        this.label2 = new System.Windows.Forms.Label();
        this.cboDifficulty = new System.Windows.Forms.ComboBox();
        this.cbLock = new System.Windows.Forms.CheckBox();
        this.cbTrap = new System.Windows.Forms.CheckBox();
        this.label3 = new System.Windows.Forms.Label();
        this.tbTrap = new System.Windows.Forms.TextBox();
        this.label4 = new System.Windows.Forms.Label();
        this.label5 = new System.Windows.Forms.Label();
        this.tbKeyName = new System.Windows.Forms.TextBox();
        this.tbKeyType = new System.Windows.Forms.TextBox();
        this.label6 = new System.Windows.Forms.Label();
        this.nudKeys = new System.Windows.Forms.NumericUpDown();
        this.lbItems = new System.Windows.Forms.ListBox();
        this.btnAdd = new System.Windows.Forms.Button();
        this.btnRemove = new System.Windows.Forms.Button();
        this.label8 = new System.Windows.Forms.Label();
        this.nudMinGold = new System.Windows.Forms.NumericUpDown();
        this.nudMaxGold = new System.Windows.Forms.NumericUpDown();
        this.label9 = new System.Windows.Forms.Label();
        this.groupBox1 = new System.Windows.Forms.GroupBox();
        this.groupBox2 = new System.Windows.Forms.GroupBox();
        this.groupBox3 = new System.Windows.Forms.GroupBox();
        this.groupBox4 = new System.Windows.Forms.GroupBox();
        this.btnOK = new System.Windows.Forms.Button();
        this.btnCancel = new System.Windows.Forms.Button();
        ((System.ComponentModel.ISupportInitialize)(this.nudKeys)).BeginInit();
        ((System.ComponentModel.ISupportInitialize)(this.nudMinGold)).BeginInit();
        ((System.ComponentModel.ISupportInitialize)(this.nudMaxGold)).BeginInit();
        this.groupBox1.SuspendLayout();
        this.groupBox2.SuspendLayout();
        this.groupBox3.SuspendLayout();
        this.groupBox4.SuspendLayout();
        this.SuspendLayout();
        //
        // tbName
        //
        this.tbName.Location = new System.Drawing.Point(113, 17);
        this.tbName.Margin = new System.Windows.Forms.Padding(4);
        this.tbName.Name = "tbName";
        this.tbName.Size = new System.Drawing.Size(169, 22);
        this.tbName.TabIndex = 1;
        //
        // label1
```

```
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(16, 20);
            this.label1.Margin = new System.Windows.Forms.Padding(4, 0, 4, 0);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(89, 17);
            this.label1.TabIndex = 0;
            this.label1.Text = "Chest Name:";
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Location = new System.Drawing.Point(11, 44);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(99, 17);
            this.label2.TabIndex = 1;
            this.label2.Text = "Lock Difficulty:";
            //
            // cboDifficulty
            //
            this.cboDifficulty.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDownList;
            this.cboDifficulty.Enabled = false;
            this.cboDifficulty.FormattingEnabled = true;
            this.cboDifficulty.Location = new System.Drawing.Point(117, 41);
            this.cboDifficulty.Name = "cboDifficulty";
            this.cboDifficulty.Size = new System.Drawing.Size(132, 24);
            this.cboDifficulty.TabIndex = 2;
            //
            // cbLock
            //
            this.cbLock.AutoSize = true;
            this.cbLock.Location = new System.Drawing.Point(30, 20);
            this.cbLock.Name = "cbLock";
            this.cbLock.Size = new System.Drawing.Size(76, 21);
            this.cbLock.TabIndex = 0;
            this.cbLock.Text = "Locked";
            this.cbLock.UseVisualStyleBackColor = true;
            //
            // cbTrap
            //
            this.cbTrap.AutoSize = true;
            this.cbTrap.Location = new System.Drawing.Point(9, 20);
            this.cbTrap.Name = "cbTrap";
            this.cbTrap.Size = new System.Drawing.Size(84, 21);
            this.cbTrap.TabIndex = 0;
            this.cbTrap.Text = "Trapped";
            this.cbTrap.UseVisualStyleBackColor = true;
            //
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.Location = new System.Drawing.Point(10, 44);
            this.label3.Margin = new System.Windows.Forms.Padding(4, 0, 4, 0);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(83, 17);
            this.label3.TabIndex = 1;
            this.label3.Text = "Trap Name:";
            //
            // tbTrap
            //
```

```
this.tbTrap.Enabled = false;
this.tbTrap.Location = new System.Drawing.Point(100, 41);
this.tbTrap.Name = "tbTrap";
this.tbTrap.Size = new System.Drawing.Size(132, 22);
this.tbTrap.TabIndex = 1;
//
// label4
//
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(33, 76);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(77, 17);
this.label4.TabIndex = 3;
this.label4.Text = "Key Name:";
//
// label5
//
this.label5.AutoSize = true;
this.label5.Location = new System.Drawing.Point(38, 106);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(72, 17);
this.label5.TabIndex = 5;
this.label5.Text = "Key Type:";
//
// tbKeyName
//
this.tbKeyName.Enabled = false;
this.tbKeyName.Location = new System.Drawing.Point(117, 73);
this.tbKeyName.Margin = new System.Windows.Forms.Padding(4);
this.tbKeyName.Name = "tbKeyName";
this.tbKeyName.Size = new System.Drawing.Size(132, 22);
this.tbKeyName.TabIndex = 4;
//
// tbKeyType
//
this.tbKeyType.Enabled = false;
this.tbKeyType.Location = new System.Drawing.Point(117, 103);
this.tbKeyType.Margin = new System.Windows.Forms.Padding(4);
this.tbKeyType.Name = "tbKeyType";
this.tbKeyType.Size = new System.Drawing.Size(132, 22);
this.tbKeyType.TabIndex = 6;
//
// label6
//
this.label6.AutoSize = true;
this.label6.Location = new System.Drawing.Point(13, 134);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(97, 17);
this.label6.TabIndex = 7;
this.label6.Text = "Keys Needed:";
//
// nudKeys
//
this.nudKeys.Enabled = false;
this.nudKeys.Location = new System.Drawing.Point(117, 132);
this.nudKeys.Name = "nudKeys";
this.nudKeys.Size = new System.Drawing.Size(132, 22);
this.nudKeys.TabIndex = 8;
//
// lbItems
```

```
//
this.lbItems.FormattingEnabled = true;
this.lbItems.ItemHeight = 16;
this.lbItems.Location = new System.Drawing.Point(6, 21);
this.lbItems.Name = "lbItems";
this.lbItems.Size = new System.Drawing.Size(359, 324);
this.lbItems.TabIndex = 0;
//
// btnAdd
//
this.btnAdd.Name = "btnAdd";
this.btnAdd.Size = new System.Drawing.Size(75, 23);
this.btnAdd.TabIndex = 1;
this.btnAdd.Text = "Add";
this.btnAdd.UseVisualStyleBackColor = true;
//
// btnRemove
//
this.btnRemove.Location = new System.Drawing.Point(201, 351);
this.btnRemove.Name = "btnRemove";
this.btnRemove.Size = new System.Drawing.Size(75, 23);
this.btnRemove.TabIndex = 2;
this.btnRemove.Text = "Remove";
this.btnRemove.UseVisualStyleBackColor = true;
//
// label8
//
this.label8.AutoSize = true;
this.label8.Location = new System.Drawing.Point(9, 23);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(101, 17);
this.label8.TabIndex = 0;
this.label8.Text = "Minimum Gold:";
//
// nudMinGold
//
this.nudMinGold.Location = new System.Drawing.Point(117, 21);
this.nudMinGold.Maximum = new decimal(new int[] {
10000,
0,
0,
0});
this.nudMinGold.Name = "nudMinGold";
this.nudMinGold.Size = new System.Drawing.Size(132, 22);
this.nudMinGold.TabIndex = 0;
//
// nudMaxGold
//
this.nudMaxGold.Location = new System.Drawing.Point(117, 49);
this.nudMaxGold.Maximum = new decimal(new int[] {
10000,
0,
0,
0});
this.nudMaxGold.Name = "nudMaxGold";
this.nudMaxGold.Size = new System.Drawing.Size(132, 22);
this.nudMaxGold.TabIndex = 1;
//
// label9
//
```

```
this.label9.AutoSize = true;
this.label9.Location = new System.Drawing.Point(6, 51);
this.label9.Name = "label9";
this.label9.Size = new System.Drawing.Size(104, 17);
this.label9.TabIndex = 2;
this.label9.Text = "Maximum Gold:";
//
// groupBox1
//
this.groupBox1.Controls.Add(this.cboDifficulty);
this.groupBox1.Controls.Add(this.tbKeyName);
this.groupBox1.Controls.Add(this.tbKeyType);
this.groupBox1.Controls.Add(this.label2);
this.groupBox1.Controls.Add(this.label4);
this.groupBox1.Controls.Add(this.label5);
this.groupBox1.Controls.Add(this.label6);
this.groupBox1.Controls.Add(this.cbLock);
this.groupBox1.Controls.Add(this.nudKeys);
this.groupBox1.Location = new System.Drawing.Point(12, 41);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Size = new System.Drawing.Size(270, 163);
this.groupBox1.TabIndex = 2;
this.groupBox1.TabStop = false;
this.groupBox1.Text = "Lock Properties";
//
// groupBox2
//
this.groupBox2.Controls.Add(this.cbTrap);
this.groupBox2.Controls.Add(this.tbTrap);
this.groupBox2.Controls.Add(this.label3);
this.groupBox2.Location = new System.Drawing.Point(12, 220);
this.groupBox2.Name = "groupBox2";
this.groupBox2.Size = new System.Drawing.Size(270, 82);
this.groupBox2.TabIndex = 3;
this.groupBox2.TabStop = false;
this.groupBox2.Text = "Trap Properties";
//
// groupBox3
//
this.groupBox3.Controls.Add(this.nudMinGold);
this.groupBox3.Controls.Add(this.nudMaxGold);
this.groupBox3.Controls.Add(this.label8);
this.groupBox3.Controls.Add(this.label9);
this.groupBox3.Location = new System.Drawing.Point(12, 308);
this.groupBox3.Name = "groupBox3";
this.groupBox3.Size = new System.Drawing.Size(270, 87);
this.groupBox3.TabIndex = 4;
this.groupBox3.TabStop = false;
this.groupBox3.Text = "Gold Properties";
//
// groupBox4
//
this.groupBox4.Controls.Add(this.lbItems);
this.groupBox4.Controls.Add(this.btnAdd);
this.groupBox4.Controls.Add(this.btnRemove);
this.groupBox4.Location = new System.Drawing.Point(302, 16);
this.groupBox4.Name = "groupBox4";
this.groupBox4.Size = new System.Drawing.Size(376, 379);
this.groupBox4.TabIndex = 5;
this.groupBox4.TabStop = false;
```

```
        this.groupBox4.Text = "Item Properties";
        //
        // btnOK
        //
        this.btnOK.Location = new System.Drawing.Point(262, 415);
        this.btnOK.Name = "btnOK";
        this.btnOK.Size = new System.Drawing.Size(75, 23);
        this.btnOK.TabIndex = 6;
        this.btnOK.Text = "OK";
        this.btnOK.UseVisualStyleBackColor = true;
        //
        // btnCancel
        //
        this.btnCancel.Location = new System.Drawing.Point(356, 415);
        this.btnCancel.Name = "btnCancel";
        this.btnCancel.Size = new System.Drawing.Size(75, 23);
        this.btnCancel.TabIndex = 7;
        this.btnCancel.Text = "Cancel";
        this.btnCancel.UseVisualStyleBackColor = true;
        //
        // FormChestDetails
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(692, 450);
        this.ControlBox = false;
        this.Controls.Add(this.btnCancel);
        this.Controls.Add(this.btnOK);
        this.Controls.Add(this.groupBox4);
        this.Controls.Add(this.groupBox3);
        this.Controls.Add(this.groupBox2);
        this.Controls.Add(this.groupBox1);
        this.Controls.Add(this.tbName);
        this.Controls.Add(this.label1);
        this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
        this.Name = "FormChestDetails";
        this.StartPosition = System.Windows.Forms.FormStartPosition.CenterParent;
        this.Text = "Chest";
        ((System.ComponentModel.ISupportInitialize)(this.nudKeys)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.nudMinGold)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.nudMaxGold)).EndInit();
        this.groupBox1.ResumeLayout(false);
        this.groupBox1.PerformLayout();
        this.groupBox2.ResumeLayout(false);
        this.groupBox2.PerformLayout();
        this.groupBox3.ResumeLayout(false);
        this.groupBox3.PerformLayout();
        this.groupBox4.ResumeLayout(false);
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion

    private System.Windows.Forms.TextBox tbName;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.ComboBox cboDifficulty;
    private System.Windows.Forms.CheckBox cbLock;
```

```
        private System.Windows.Forms.CheckBox cbTrap;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.TextBox tbTrap;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.TextBox tbKeyName;
        private System.Windows.Forms.TextBox tbKeyType;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.NumericUpDown nudKeys;
        private System.Windows.Forms.ListBox lbItems;
        private System.Windows.Forms.Button btnAdd;
        private System.Windows.Forms.Button btnRemove;
        private System.Windows.Forms.Label label8;
        private System.Windows.Forms.NumericUpDown nudMinGold;
        private System.Windows.Forms.NumericUpDown nudMaxGold;
        private System.Windows.Forms.Label label9;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.GroupBox groupBox3;
        private System.Windows.Forms.GroupBox groupBox4;
        private System.Windows.Forms.Button btnOK;
        private System.Windows.Forms.Button btnCancel;
    }
}
```

I'm going to add some logic to the form now. I'm going to skip the items for now and add that in a future tutorial. Right click **FormChestDetails** in the solution explorer and select **View Code**. Change the code for **FormChestDetails** 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.ItemClasses;
using RpgLibrary.TrapClasses;
using RpgLibrary.SkillClasses;

namespace RpgEditor
{
    public partial class FormChestDetails : Form
    {
        #region Field Region

        ChestData chest;

        #endregion
        #region Property Region

        public ChestData Chest
        {
            get { return chest; }
            set { chest = value; }
        }

        #endregion
```

```
#region Constructor Region

public FormChestDetails()
{
    InitializeComponent();

    this.Load += new EventHandler(FormChestDetails_Load);
    this.FormClosing += new FormClosingEventHandler(FormChestDetails_FormClosing);

    foreach (string s in Enum.GetNames(typeof(DifficultyLevel)))
    {
        cboDifficulty.Items.Add(s);
    }

    cboDifficulty.SelectedIndex = 0;

    cbLock.CheckedChanged += new EventHandler(cbLock_CheckedChanged);
    cbTrap.CheckedChanged += new EventHandler(cbTrap_CheckedChanged);

    btnAdd.Click += new EventHandler(btnAdd_Click);
    btnRemove.Click += new EventHandler(btnRemove_Click);

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

#endregion

#region Form Event Handler Region

void FormChestDetails_Load(object sender, EventArgs e)
{
    if (chest != null)
    {
        tbName.Text = chest.Name;
        cbLock.Checked = chest.IsLocked;
        tbKeyName.Text = chest.KeyName;
        tbKeyType.Text = chest.KeyType;
        nudKeys.Value = (decimal)chest.KeysRequired;
        tbKeyName.Enabled = chest.IsLocked;
        tbKeyType.Enabled = chest.IsLocked;
        nudKeys.Enabled = chest.IsLocked;
        cbTrap.Checked = chest.IsTrapped;
        tbTrap.Text = chest.TrapName;
        tbTrap.Enabled = chest.IsTrapped;
        nudMinGold.Value = (decimal)chest.MinGold;
        nudMaxGold.Value = (decimal)chest.MaxGold;
    }
}

void FormChestDetails_FormClosing(object sender, FormClosingEventArgs e)
{
    if (e.CloseReason == CloseReason.UserClosing)
    {
        e.Cancel = true;
    }
}

#endregion
```

```
        #region Check Box Event Handler Region

        void cbLock_CheckedChanged(object sender, EventArgs e)
        {
            cboDifficulty.Enabled = cbLock.Checked;

            tbKeyName.Enabled = cbLock.Checked;
            tbKeyType.Enabled = cbLock.Checked;

            nudKeys.Enabled = cbLock.Checked;
        }

        void cbTrap_CheckedChanged(object sender, EventArgs e)
        {
            tbTrap.Enabled = cbTrap.Checked;
        }

        #endregion

        #region Button Event Handler Region

        void btnAdd_Click(object sender, EventArgs e)
        {
        }

        void btnRemove_Click(object sender, EventArgs e)
        {
        }

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

            if (cbTrap.Checked && string.IsNullOrEmpty(tbTrap.Text))
            {
                MessageBox.Show("You must supply a name for the trap on the chest.");
                return;
            }

            if (nudMaxGold.Value < nudMinGold.Value)
            {
                MessageBox.Show("Maximum gold in chest must be greater or equal to minimum
gold.");
                return;
            }

            ChestData data = new ChestData();

            data.Name = tbName.Text;
            data.IsLocked = cbLock.Checked;

            if (cbLock.Checked)
            {
                data.DifficultyLevel = (DifficultyLevel)cboDifficulty.SelectedIndex;
                data.KeyName = tbKeyName.Text;
```

```
                data.KeyType = tbKeyType.Text;
                data.KeysRequired = (int)nudKeys.Value;
            }

            data.IsTrapped = cbTrap.Checked;

            if (cbTrap.Checked)
            {
                data.TrapName = tbTrap.Text;
            }

            data.MinGold = (int)nudMinGold.Value;
            data.MaxGold = (int)nudMaxGold.Value;

            chest = data;

            this.FormClosing -= FormChestDetails_FormClosing;
            this.Close();
        }

        void btnCancel_Click(object sender, EventArgs e)
        {
            chest = null;

            this.FormClosing -= FormChestDetails_FormClosing;
            this.Close();
        }

        #endregion
    }
}
```

The code should look some what familiar. I've used the same basic code in all of the forms for creating a specific item. There is some new stuff though. As usual there is a field and property for the type of item that is being created, **ChestData** in this case.

The constructor wires several event handlers. The **Load** and **FormClosing** events are wired first. I also fill the **Combo Box** with the names from the **DifficultyLevel** enumeration and set the **SelectedIndex** of the **Combo Box** to 0. I then wire the handlers for the **Check** event of the two **Check Boxes**. I also wire the handlers for all four of the buttons.
In the **Load** event handler I check to see if the **chest** field is not null. If it has a value I fill the form with values. I set the **Enabled** properties of controls associated with a chest being locked to be the **IsLocked** field of the **chest** field. So, if the chest is locked the controls will be enabled and disabled if the chest is not locked. I do the same with the **IsTrapped** field and the controls associated with a chest being trapped. I'm not worrying about items at the moment. That will be added in down the road.
There is nothing you haven't seen before in the **FormClosing** event handler. It just cancels the event if the reason for closing the form is **UserClosing**. The event will be unsubscribed from if a chest is successfully created or the user cancels the changes.

In the **CheckChanged** handlers I set the **Enabled** property of the controls associated with the **Check Box** to the **Checked** property of the **Check Box**. If the **Check Box** is checked then the controls will be enabled and disabled if the **Check Box** is not checked.

The **Click** event handler for **btnOK** does a little validation of the form. It checks to see if the chest has a name. If **cbTrap** is checked and the **Text** property of **tbTrap** is null or empty then the user must supply a name for the trap. Ideally you will want to confirm that the trap actually exists or in game ignore the **IsTrapped** property if no trap exists. I also check to make sure that the value of max gold is not less than min gold. I then create a new chest using the values on the form. I set the field to be the new chest, unsubscribe from the **FormClosing** event handler and then close the form.
The event handler for the **Click** event of **btnCancel** holds nothing new or interesting. It just assigns the **chest** field to be null, unsubscribes the **FormClosing** event handler and closes the form.

I'm now going to add the code to **FormChest**. Right click **FormChest** in the solution explorer and select **View Code** to bring up the code. This is the code for **FormChest**.

```
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 System.IO;
using RpgLibrary.ItemClasses;

namespace RpgEditor
{
    public partial class FormChest : FormDetails
    {
        #region Field Region
        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public FormChest()
        {
            InitializeComponent();

            btnAdd.Click += new EventHandler(btnAdd_Click);
            btnEdit.Click += new EventHandler(btnEdit_Click);
            btnDelete.Click += new EventHandler(btnDelete_Click);
        }

        #endregion

        #region Event Handler Region

        void btnAdd_Click(object sender, EventArgs e)
        {
            using (FormChestDetails frmChestDetails = new FormChestDetails())
            {
                frmChestDetails.ShowDialog();

                if (frmChestDetails.Chest != null)
                {
                    AddChest(frmChestDetails.Chest);
```

```
        }
    }
}
void btnEdit_Click(object sender, EventArgs e)
{
    if (lbDetails.SelectedItem != null)
    {
        string detail = lbDetails.SelectedItem.ToString();
        string[] parts = detail.Split(',');
        string entity = parts[0].Trim();

        ChestData data = itemManager.ChestData[entity];
        ChestData newData = null;

        using (FormChestDetails frmChestData = new FormChestDetails())
        {
            frmChestData.Chest = data;
            frmChestData.ShowDialog();

            if (frmChestData.Chest == null)
                return;

            if (frmChestData.Chest.Name == entity)
            {
                itemManager.ChestData[entity] = frmChestData.Chest;
                FillListBox();

                return;
            }

            newData = frmChestData.Chest;
        }

        DialogResult result = MessageBox.Show(
            "Name has changed. Do you want to add a new entry?",
            "New Entry",
            MessageBoxButtons.YesNo);

        if (result == DialogResult.No)
            return;

        if (itemManager.ChestData.ContainsKey(newData.Name))
        {
            MessageBox.Show("Entry already exists. Use Edit to modify the entry.");
            return;
        }

        lbDetails.Items.Add(newData);
        itemManager.ChestData.Add(newData.Name, newData);
    }
}
void btnDelete_Click(object sender, EventArgs e)
{
    if (lbDetails.SelectedItem != null)
    {
        string detail = (string)lbDetails.SelectedItem;
        string[] parts = detail.Split(',');
        string entity = parts[0].Trim();

        DialogResult result = MessageBox.Show(
```

```
                "Are you sure you want to delete " + entity + "?",
                "Delete",
                MessageBoxButtons.YesNo);

            if (result == DialogResult.Yes)
            {
                lbDetails.Items.RemoveAt(lbDetails.SelectedIndex);
                itemManager.ChestData.Remove(entity);

                if (File.Exists(FormMain.ItemPath + @"\Chest\" + entity + ".xml"))
                    File.Delete(FormMain.ItemPath + @"\Chest\" + entity + ".xml");
            }
        }
    }

    #endregion

    #region Method Region

    public void FillListBox()
    {
        lbDetails.Items.Clear();

        foreach (string s in FormDetails.ItemManager.ChestData.Keys)
            lbDetails.Items.Add(FormDetails.ItemManager.ChestData[s]);
    }

    private void AddChest(ChestData ChestData)
    {
        if (FormDetails.ItemManager.ChestData.ContainsKey(ChestData.Name))
        {
            DialogResult result = MessageBox.Show(
                ChestData.Name + " already exists. Overwrite it?",
                "Existing Chest",
                MessageBoxButtons.YesNo);

            if (result == DialogResult.No)
                return;

            itemManager.ChestData[ChestData.Name] = ChestData;
            FillListBox();

            return;
        }

        itemManager.ChestData.Add(ChestData.Name, ChestData);
        lbDetails.Items.Add(ChestData);
    }

    #endregion
  }
}
```

Again, this should look familiar as other forms use the same code. The difference is that instead of working with armor, shields, or weapons, I'm working with chests. There is the using statement for the **System.IO** name space. Event handlers for the click event of the buttons are wired in the constructors.

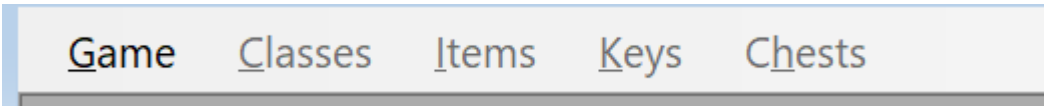
The **Click** event handler of **btnAdd** creates a form in a using block. I show the form. If the **Chest**

property of the form is not null I call the **AddChest** method passing in the **Chest** property. The **Click** event handler of **btnEdit** checks to see if the **SelectedItem** of **lbDetails** is not null. It parses the string to get the name of the chest. It then gets the **ChestData** for the selected item and sets **newData** to null.

In a using statement a form is created. The **Chest** property of the form is set to the **ChestData** of **SelectedItem.** I call the **ShowDialog** method to display the form. If the **Chest** property of form is null I exit the method. If the name is the same as before I assign the entry in the item manager to be the new key, call **FillListBox** to update the key and exit the method. I then set **newData** to be the **Chest** property of the form. The name of the chest changed so I display a message box asking if the new chest should be added. If the result is no I exit the method. If there is a chest with that name already I display a message box and exit. If there wasn't I add the new chest to the list box and the item manager. The **Click** event handler for **btnDelete** checks to make sure that the **SelectedItem** of the list box is not null.

It parses the selected item and displays a message box asking if the chest should be deleted. If the result of the message box is Yes I remove the chest from the list box and I remove if from the item manager as well. I then delete the file, if it exists.

The last thing I'm going to tackle in the editor is adding the functionality for the new forms. That will be done in **FormMain**. I need to update the form a little to handle chests and keys. They will have menu entries of their own. Right click **FormMain** in the solution explorer and select **View Designer**. Click the **Menu Strip** on the form. Beside the **Items** entry add a new entry **&Keys**. Set the **Enabled** property of this item to **False**. Beside the **Keys** entry add a new entry **C&hests** and set its **Enabled** property to **False** as well. Your menu bar should resemble the following.




The original tutorial 18 was split into two parts but I am going to continue on in one part. I was working on adding chests and keys to the editor and being able to read in chests at run time. At the end of the last tutorial I had add in new items to **FormMain** in the designer but I haven't added in the code to handle them. Right click **FormMain** in the editor and select **View Code**. This is the code for **FormMain**.

```
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 System.IO;
using RpgLibrary;
using RpgLibrary.CharacterClasses;
using RpgLibrary.ItemClasses;

namespace RpgEditor
```

```
{
    public partial class FormMain : Form
    {
        #region Field Region

        RolePlayingGame rolePlayingGame;
        FormClasses frmClasses;
        FormArmor frmArmor;
        FormShield frmShield;
        FormWeapon frmWeapon;
        FormKey frmKey;
        FormChest frmChest;

        static string gamePath = "";
        static string classPath = "";
        static string itemPath = "";
        static string chestPath = "";
        static string keyPath = "";

        #endregion

        #region Property Region

        public static string GamePath
        {
            get { return gamePath; }
        }

        public static string ClassPath
        {
            get { return classPath; }
        }

        public static string ItemPath
        {
            get { return itemPath; }
        }

        public static string ChestPath
        {
            get { return chestPath; }
        }

        public static string KeyPath
        {
            get { return keyPath; }
        }

        #endregion

        #region Constructor Region

        public FormMain()
        {
            InitializeComponent();

            this.FormClosing += new FormClosingEventHandler(FormMain_FormClosing);

            newGameToolStripMenuItem.Click += new EventHandler(newGameToolStripMenuItem_Click);
            openGameToolStripMenuItem.Click += new
```

```
EventHandler(openGameToolStripMenuItem_Click);
            saveGameToolStripMenuItem.Click += new
EventHandler(saveGameToolStripMenuItem_Click);
            exitToolStripMenuItem.Click += new EventHandler(exitToolStripMenuItem_Click);
            classesToolStripMenuItem.Click += new EventHandler(classesToolStripMenuItem_Click);
            armorToolStripMenuItem.Click += new EventHandler(armorToolStripMenuItem_Click);
            shieldToolStripMenuItem.Click += new EventHandler(shieldToolStripMenuItem_Click);
            weaponToolStripMenuItem.Click += new EventHandler(weaponToolStripMenuItem_Click);
            keysToolStripMenuItem.Click += new EventHandler(keysToolStripMenuItem_Click);
            chestsToolStripMenuItem.Click += new EventHandler(chestsToolStripMenuItem_Click);
        }

        #endregion

        #region Menu Item Event Handler Region

        void FormMain_FormClosing(object sender, FormClosingEventArgs e)
        {
            DialogResult result = MessageBox.Show(
                "Unsaved changes will be lost. Are you sure you want to exit?",
                "Exit?",
                MessageBoxButtons.YesNo,
                MessageBoxIcon.Warning);

            if (result == DialogResult.No)
                e.Cancel = true;
        }

        void newGameToolStripMenuItem_Click(object sender, EventArgs e)
        {
            using (FormNewGame frmNewGame = new FormNewGame())
            {
                DialogResult result = frmNewGame.ShowDialog();

                if (result == DialogResult.OK && frmNewGame.RolePlayingGame != null)
                {
                    FolderBrowserDialog folderDialog = new FolderBrowserDialog();

                    folderDialog.Description = "Select folder to create game in.";
                    folderDialog.SelectedPath = Application.StartupPath;

                    DialogResult folderResult = folderDialog.ShowDialog();

                    if (folderResult == DialogResult.OK)
                    {
                        try
                        {
                            gamePath = Path.Combine(folderDialog.SelectedPath, "Game");
                            classPath = Path.Combine(gamePath, "Classes");
                            itemPath = Path.Combine(gamePath, "Items");
                            keyPath = Path.Combine(gamePath, "Keys");
                            chestPath = Path.Combine(gamePath, "Chests");

                            if (Directory.Exists(gamePath))
                                throw new Exception("Selected directory already exists.");

                            Directory.CreateDirectory(gamePath);
                            Directory.CreateDirectory(classPath);
                            Directory.CreateDirectory(itemPath + @"\Armor");
                            Directory.CreateDirectory(itemPath + @"\Shield");
```

```
                    Directory.CreateDirectory(itemPath + @"\Weapon");
                    Directory.CreateDirectory(keyPath);
                    Directory.CreateDirectory(chestPath);

                    rolePlayingGame = frmNewGame.RolePlayingGame;

                    XnaSerializer.Serialize<RolePlayingGame>(gamePath + @"\Game.xml",
                        rolePlayingGame);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.ToString());
                    return;
                }

                classesToolStripMenuItem.Enabled = true;
                itemsToolStripMenuItem.Enabled = true;
                keysToolStripMenuItem.Enabled = true;
                chestsToolStripMenuItem.Enabled = true;
            }
        }
    }
}

void openGameToolStripMenuItem_Click(object sender, EventArgs e)
{
    FolderBrowserDialog folderDialog = new FolderBrowserDialog();

    folderDialog.Description = "Select Game folder";
    folderDialog.SelectedPath = Application.StartupPath;

    bool tryAgain = false;

    do
    {
        DialogResult folderResult = folderDialog.ShowDialog();
        DialogResult msgBoxResult;

        if (folderResult == DialogResult.OK)
        {
            if (File.Exists(folderDialog.SelectedPath + @"\Game\Game.xml"))
            {
                try
                {
                    OpenGame(folderDialog.SelectedPath);
                    tryAgain = false;
                }
                catch (Exception ex)
                {
                    msgBoxResult = MessageBox.Show(
                        ex.ToString(),
                        "Error opening game.",
                        MessageBoxButtons.RetryCancel);

                    if (msgBoxResult == DialogResult.Cancel)
                        tryAgain = false;
                    else if (msgBoxResult == DialogResult.Retry)
                        tryAgain = true;
                }
            }
```

```
            else
            {
                msgBoxResult = MessageBox.Show(
                    "Game not found, try again?",
                    "Game does not exist",
                    MessageBoxButtons.RetryCancel);

                if (msgBoxResult == DialogResult.Cancel)
                    tryAgain = false;
                else if (msgBoxResult == DialogResult.Retry)
                    tryAgain = true;
            }
        }
    } while (tryAgain);
}

void saveGameToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (rolePlayingGame != null)
    {
        try
        {
            XnaSerializer.Serialize<RolePlayingGame>(gamePath + @"\Game.xml",
                rolePlayingGame);

            FormDetails.WriteEntityData();
            FormDetails.WriteItemData();
            FormDetails.WriteChestData();
            FormDetails.WriteKeyData();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.ToString(), "Error saving game.");
        }
    }
}

void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}

void classesToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (frmClasses == null)
    {
        frmClasses = new FormClasses();
        frmClasses.MdiParent = this;
    }

    frmClasses.Show();
    frmClasses.BringToFront();
}

void armorToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (frmArmor == null)
    {
        frmArmor = new FormArmor();
        frmArmor.MdiParent = this;
```

```
    }

    frmArmor.Show();
    frmArmor.BringToFront();
}

void shieldToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (frmShield == null)
    {
        frmShield = new FormShield();
        frmShield.MdiParent = this;
    }

    frmShield.Show();
    frmShield.BringToFront();
}
void weaponToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (frmWeapon == null)
    {
        frmWeapon = new FormWeapon();
        frmWeapon.MdiParent = this;
    }

    frmWeapon.Show();
    frmWeapon.BringToFront();
}

void chestsToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (frmChest == null)
    {
        frmChest = new FormChest();
        frmChest.MdiParent = this;
    }

    frmChest.Show();
    frmChest.BringToFront();
}

void keysToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (frmKey == null)
    {
        frmKey = new FormKey();
        frmKey.MdiParent = this;
    }

    frmKey.Show();
    frmKey.BringToFront();
}

#endregion

#region Method Region

private void OpenGame(string path)
{
    gamePath = Path.Combine(path, "Game");
```

```
        classPath = Path.Combine(gamePath, "Classes");
        itemPath = Path.Combine(gamePath, "Items");
        keyPath = Path.Combine(gamePath, "Keys");
        chestPath = Path.Combine(gamePath, "Chests");

        if (!Directory.Exists(keyPath))
        {
            Directory.CreateDirectory(keyPath);
        }

        if (!Directory.Exists(chestPath))
        {
            Directory.CreateDirectory(chestPath);
        }

        rolePlayingGame = XnaSerializer.Deserialize<RolePlayingGame>(
            gamePath + @"\Game.xml");
        FormDetails.ReadEntityData();
        FormDetails.ReadItemData();
        FormDetails.ReadKeyData();
        FormDetails.ReadChestData();

        PrepareForms();
    }

    private void PrepareForms()
    {
        if (frmClasses == null)
        {
            frmClasses = new FormClasses();
            frmClasses.MdiParent = this;
        }

        frmClasses.FillListBox();

        if (frmArmor == null)
        {
            frmArmor = new FormArmor();
            frmArmor.MdiParent = this;
        }

        frmArmor.FillListBox();

        if (frmShield == null)
        {
            frmShield = new FormShield();
            frmShield.MdiParent = this;
        }

        frmShield.FillListBox();

        if (frmWeapon == null)
        {
            frmWeapon = new FormWeapon();
            frmWeapon.MdiParent = this;
        }

        frmWeapon.FillListBox();

        if (frmKey == null)
```

```
            {
                frmKey = new FormKey();
                frmKey.MdiParent = this;
            }

            frmKey.FillListBox();

            if (frmChest == null)
            {
                frmChest = new FormChest();
                frmChest.MdiParent = this;
            }

            frmChest.FillListBox();

            classesToolStripMenuItem.Enabled = true;
            itemsToolStripMenuItem.Enabled = true;
            keysToolStripMenuItem.Enabled = true;
            chestsToolStripMenuItem.Enabled = true;
        }

        #endregion
    }
}
```

Quite a few new additions to the code of **FormMain**. I added in fields for **FormKey** and **FormChest**. I also static fields for the path to keys and chests. The fields are **keyPath** and **chestPath**. The properties are **KeyPath** and **ChestPath**. The properties, like the other static properties, are read only.
What is new in the constructor is that I wire event handlers for **Click** events for the two new menu items, like the other menu items. In the event handler for the new game I create paths for **keyPath** and **chestPath** like the other paths. Keys and chests reside in their own directories. I also create the new directories like the other paths. I also enable the two new menu items. The event handler for the **Click** event for saving a game calls two static methods that I haven't written yet on **FormDetails** called **WriteChestData** and **WriteKeyData**. As you can guess they will write out chest and key data.

In the event handlers for the **Click** event of the new menu items I first check to see if the form for that menu item is null. If it is null then I create a new form and set its **MdiParent** property to be the current form. I then call the **Show** and **BringToFront** methods to display the forms and bring them to the front. Since the directory structure for a game has changed in the **OpenGame** method I check to see if the directories pointed to by **keyPath** and **chestPath** don't exist. If they don't exists then I create them. I then call the **ReadKeyData** and **ReadChestData** methods of **FormDetails** that I also haven't written yet and will get to soon.

The last change is in the **PrepareForms** method. I check to see if the forms are null. If they are null I create new instances and set their **MdiParent** properties to **this**. I also call their **FillListBox** methods to fill them with the appropriate items.

I need to update the code for **FormDetails**. I have to add the write and read methods for writing and reading chests and keys. Right click **FormDetails** and select **View Code** to view the code. Add the following methods.

```
public static void WriteKeyData()
{
    foreach (string s in ItemManager.KeyData.Keys)
    {
        XnaSerializer.Serialize<KeyData>(
            FormMain.KeyPath + @"\" + s + ".xml",
            ItemManager.KeyData[s]);
    }
}

public static void WriteChestData()
{
    foreach (string s in ItemManager.ChestData.Keys)
    {
        XnaSerializer.Serialize<ChestData>(
            FormMain.ChestPath + @"\" + s + ".xml",
            ItemManager.ChestData[s]);
    }
}

public static void ReadKeyData()
{
    string[] fileNames = Directory.GetFiles(FormMain.KeyPath, "*.xml");

    foreach (string s in fileNames)
    {
        KeyData keyData = XnaSerializer.Deserialize<KeyData>(s);
        itemManager.KeyData.Add(keyData.Name, keyData);
    }
}

public static void ReadChestData()
{
    string[] fileNames = Directory.GetFiles(FormMain.ChestPath, "*.xml");

    foreach (string s in fileNames)
    {
        ChestData chestData = XnaSerializer.Deserialize<ChestData>(s);
        itemManager.ChestData.Add(chestData.Name, chestData);
    }
}
```

The code is similar to the other writing and reading code. For writing in a foreach loop I loop through all of the keys in the **ItemManager** for that type of data. I then call the **Serialize** method of the **XnaSerializer** class with the right data type, path to the file, and the actual item to write. To create the path you take the path to that type of object and append a \ the name of the item and .xml for the extension. To read in the items you first need to get all of the items of that type in the directory. You use the **GetFiles** method of the **Directory** class to call all of the files with an xml extension. In a foreach loop I loop through all of the file names. I then call the **Deserialize** method of the **XnaSerilializer** class with the proper data type and the file name storing the object in a variable. I then add the object to the **ItemManager** passing in the name of the object for the key and the actual object for the value.

Right click the **RpgEditor** and select **Set As StartUp Project** if the editor isn't already the start up project. Build and run to launch the editor. Once you've done that select the **Open** menu item and navigate to your game data from tutorial 14. I'm going to add a couple keys and chests to the editor.

Add the following keys and chests then save the game. Optionally, you can download the content for the tutorial from https://cynthiamcmahon.ca/blog/downloads/gamedata18.zip.

**Keys**

| Key Name | Key Type |
|---|---|
| Rusty Key | |
| Golden Key | Golden Key |

**Chests**

| Name | Difficulty Level | Locked | Key Name | Key Type | Keys Required | Trapped | Trap Name | Min Gold | Max Gold | Item Collection |
|---|---|---|---|---|---|---|---|---|---|---|
| Rusty Chest | Normal | Yes | Rusty Key | | 1 | No | | 100 | 150 | |
| Gold Chest | Normal | Yes | Golden Key | Golden Key | 1 | No | | 200 | 250 | |
| Big Gold Chest | Normal | Yes | Golden Key | Golden Key | 2 | No | | 500 | 750 | |
| Plain Chest | Easy | Yes | | | 0 | No | | 10 | 50 | |
| Broken Crate | Normal | No | | | 0 | No | | 5 | 10 | |

The next step will be to get a chest we created with the editor into the game. The first step will be to add the game data to the **EyesOfTheDragon** project and set the **MonoGame Pipeline Tool** to recognize our XML files as content. Open the **MonoGame Pipeline Tool** by double clicking the **Content.mgcb** node in the **Content** folder of the **EyesOfTheDragon** project.

Select the **Content** node in the **MonoGame Pipeline Tool**. Scroll down the left side until you see **References** under **Propertipes**. Click **References** to bring up the references dialog. Click the **Add** button. In that dialog navigate to .\EyesOfTheDragon\MGRpgLibrary\bin\Debug\netstandard2.0 and add the **MGRpgLibrary.dll** file. Repeat the process but this time navigate to the folder .\EyesOfTheDragon\RpgLibrary\bin\Debug\netstandard2.0 and add the **RpgLibrary.dll** file. Close the dialog to add the references to the project.

Still in the **MonoGame Pipeline Tool** right click the **Content** node, select **Add** and then **New Folder**. Name this new folder **Game**. Right click the **Game** folder, select **Add** and then **Existing Folder**. Browse to your **Game** folder and once inside the folder click **Open**. When prompter choose the **Copy** option and to use this action for all items. Save and builder your project.

You should now have a **Game** folder in your **Content** folder of your **EyesOfTheDragon** project. Right click the **EyesOfTheDragon** project and select **Set As StartUp Project** to make your game the start up project. You will need to add a reference to the **EyesOfTheDragonContent** project for your **RpgLibrary** to compile the XML files into XNB files that can be read using the **Content Pipeline**. Right click the **EyesOfTheDragonContent** project and select **Add Reference**. From the **Projects** tab select **RpgLibrary**.

The last thing I want to do is show how easy it is to read in a **ChestData** object using the **Content Pipeline**. Change the **CreateWorld** method of the **CharacterGeneratorScreen** to the following.

```
        private void CreateWorld()
        {
            Texture2D tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset1");
            Tileset tileset1 = new Tileset(tilesetTexture, 8, 8, 32, 32);

            tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset2");

            Tileset tileset2 = new Tileset(tilesetTexture, 8, 8, 32, 32);
            List<Tileset> tilesets = new List<Tileset>();
```

```
tilesets.Add(tileset1);
tilesets.Add(tileset2);

MapLayer layer = new MapLayer(100, 100);

for (int y = 0; y < layer.Height; y++)
{
    for (int x = 0; x < layer.Width; x++)
    {
        Tile tile = new Tile(0, 0);
        layer.SetTile(x, y, tile);
    }
}

MapLayer splatter = new MapLayer(100, 100);
Random random = new Random();

for (int i = 0; i < 100; i++)
{
    int x = random.Next(0, 100);
    int y = random.Next(0, 100);
    int index = random.Next(2, 14);

    Tile tile = new Tile(index, 0);

    splatter.SetTile(x, y, tile);
}

splatter.SetTile(1, 0, new Tile(0, 1));
splatter.SetTile(2, 0, new Tile(2, 1));
splatter.SetTile(3, 0, new Tile(0, 1));

List<MapLayer> mapLayers = new List<MapLayer>();

mapLayers.Add(layer);
mapLayers.Add(splatter);

TileMap map = new TileMap(tilesets, mapLayers);

Level level = new Level(map);

ChestData chestData = Game.Content.Load<ChestData>(@"Game\Chests\Plain Chest");
Chest chest = new Chest(chestData);

BaseSprite chestSprite = new BaseSprite(
    containers,
    new Rectangle(0, 0, 32, 32),
    new Point(10, 10));

ItemSprite itemSprite = new ItemSprite(
    chest,
    chestSprite);

level.Chests.Add(itemSprite);

World world = new World(GameRef, GameRef.ScreenRectangle);

world.Levels.Add(level);
world.CurrentLevel = 0;
```

```
            GamePlayScreen.World = world;
        }
```

As you can see I replaced the code where I created a **ChestData** object by hand with a call to **Load** of the **ContentManager** of our **Game** class. I passed in the **Plain Chest** that I created in the editor. We went through a little work to get there but it is well worth it in the end. Most classes that you create can be serialized using the **IntermediateSerializer** and then read in using the **Content Pipeline** at run time with out having to create a custom **Content Pipeline** extension.

I think that this is enough for this tutorial. It covers the original tutorial with a few modification. 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