XNA 4.0 RPG Tutorials

Part 14A

Back to Editors

I'm writing these tutorials for the new XNA 4.0 framework. The tutorials will make more sense if they are read in order. You can find the list of tutorials on the XNA 4.0 RPG tutorials page of my web site. I will be making my version of the project available for download at the end of each tutorial. It will be included on the page that links to the tutorials.

Before we go much further you will need data to work with. There is only so much you can do with out data. When we create a character in the game, the player character, non-player character, or monster, you need information about the character. You are going to want to know what weapon a character is holding. What kind of skill do they have with that weapon. So, we need to work on the editors.

Open up the project in Visual C# from last time. Right click on the **RpgEditor** project and select the **Set As StartUp Project** option. To make parsing the data in the list boxes a little easier I want to change the **ToString** method of the **EntityData** class. I want to remove all of the parts that contained an =. This is the new **ToString** method for the **EntityData** class.

```
public override string ToString()
{
    string toString = EntityName + ", ";
    toString += Strength.ToString() + ", ";
    toString += Dexterity.ToString() + ", ";
    toString += Cunning.ToString() + ", ";
    toString += Willpower.ToString() + ", ";
    toString += Magic.ToString() + ", ";
    toString += Constitution.ToString() + ", ";
    toString += HealthFormula + ", ";
    toString += StaminaFormula + ", ";
    toString += MagicFormula;

    return toString;
}
```

You also need to add in overrides of the **ToString** methods to the item data classes: **WeaponData**, **ArmorData** and **ShieldData**. I also included a constructor with no parameters, just to be sure that there will be no problem serializing the data. This is the updated code for those three classes.

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

namespace RpgLibrary.ItemClasses
{
    public class ArmorData
    {
        public string Name;
        public string Type;
        public int Price;
```

```
        public float Weight;
        public bool Equipped;
        public ArmorLocation ArmorLocation;
        public int DefenseValue;
        public int DefenseModifier;
        public string[] AllowableClasses;

        public ArmorData()
        {
        }

        public override string ToString()
        {
            string toString = Name + ", ";
            toString += Type + ", ";
            toString += Price.ToString() + ", ";
            toString += Weight.ToString() + ", ";
            toString += ArmorLocation.ToString() + ", ";
            toString += DefenseValue.ToString() + ", ";
            toString += DefenseModifier.ToString();

            foreach (string s in AllowableClasses)
                toString += ", " + s;

            return toString;
        }
    }
}

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

namespace RpgLibrary.ItemClasses
{
    public class ShieldData
    {
        public string Name;
        public string Type;
        public int Price;
        public float Weight;
        public bool Equipped;
        public int DefenseValue;
        public int DefenseModifier;
        public string[] AllowableClasses;

        public ShieldData()
        {
        }

        public override string ToString()
        {
            string toString = Name + ", ";
            toString += Type + ", ";
            toString += Price.ToString() + ", ";
            toString += Weight.ToString() + ", ";
            toString += DefenseValue.ToString() + ", ";
            toString += DefenseModifier.ToString();

            foreach (string s in AllowableClasses)
                toString += ", " + s;

            return toString;
        }
    }
}

using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;

namespace RpgLibrary.ItemClasses
{
    public class WeaponData
    {
        public string Name;
        public string Type;
        public int Price;
        public float Weight;
        public bool Equipped;
        public Hands NumberHands;
        public int AttackValue;
        public int AttackModifier;
        public int DamageValue;
        public int DamageModifier;
        public string[] AllowableClasses;

        public WeaponData()
        {
        }

        public override string ToString()
        {
            string toString = Name + ", ";
            toString += Type + ", ";
            toString += Price.ToString() + ", ";
            toString += Weight.ToString() + ", ";
            toString += NumberHands.ToString() + ", ";
            toString += AttackValue.ToString() + ", ";
            toString += AttackModifier.ToString() + ", ";
            toString += DamageValue.ToString() + ", ";
            toString += DamageModifier.ToString();

            foreach (string s in AllowableClasses)
                toString += ", " + s;

            return toString;
        }
    }
}
```

You've seen the same code when I added in the overrides to the **ToString** methods of the other item classes. Now everything is in place to start adding in logic to the forms.

I'm planning on making life a little easier when it comes to forms being closed. What I'm going to do is disable the close button on the forms for entering in specific data. Right click **FormEntityData** in the solution explorer and select the **View Designer** option. Click on the title bar of the form and set the **ControlBox** property to **False**. Also, set the **StartPosition** property to **CenterParent**. I'm going to add in a little code to cancel the form being closed if it isn't done by hitting the OK or Cancel buttons. Right click **FormEntityData** and select **View Code**. Change the code for that 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.CharacterClasses;

namespace RpgEditor
{
```

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

        EntityData entityData = null;

        #endregion

        #region Property Region

        public EntityData EntityData
        {
            get { return entityData; }
            set { entityData = value; }
        }

        #endregion

        #region Constructor Region

        public FormEntityData()
        {
            InitializeComponent();

            this.Load += new EventHandler(FormEntityData_Load);
            this.FormClosing += new FormClosingEventHandler(FormEntityData_FormClosing);

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

        #endregion

        #region Event Handler Region

        void FormEntityData_Load(object sender, EventArgs e)
        {
            if (entityData != null)
            {
                tbName.Text = entityData.EntityName;
                mtbStrength.Text = entityData.Strength.ToString();
                mtbDexterity.Text = entityData.Dexterity.ToString();
                mtbCunning.Text = entityData.Cunning.ToString();
                mtbWillpower.Text = entityData.Willpower.ToString();
                mtbConstitution.Text = entityData.Constitution.ToString();
                tbHealth.Text = entityData.HealthFormula;
                tbStamina.Text = entityData.StaminaFormula;
                tbMana.Text = entityData.MagicFormula;
            }
        }

        void FormEntityData_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) || string.IsNullOrEmpty(tbHealth.Text) ||
                string.IsNullOrEmpty(tbStamina.Text) || string.IsNullOrEmpty(tbMana.Text))
            {
                MessageBox.Show("Name, Health Formula, Stamina Formula and Mana Formula must have
values.");
                return;
            }

            int str = 0;
```

```
            int dex = 0;
            int cun = 0;
            int wil = 0;
            int mag = 0;
            int con = 0;

            if (!int.TryParse(mtbStrength.Text, out str))
            {
                MessageBox.Show("Strength must be numeric.");
                return;
            }

            if (!int.TryParse(mtbDexterity.Text, out dex))
            {
                MessageBox.Show("Dexterity must be numeric.");
                return;
            }

            if (!int.TryParse(mtbCunning.Text, out cun))
            {
                MessageBox.Show("Cunning must be numeric.");
                return;
            }

            if (!int.TryParse(mtbWillpower.Text, out wil))
            {
                MessageBox.Show("Willpower must be numeric.");
                return;
            }

            if (!int.TryParse(mtbMagic.Text, out mag))
            {
                MessageBox.Show("Magic must be numeric.");
                return;
            }

            if (!int.TryParse(mtbConstitution.Text, out con))
            {
                MessageBox.Show("Constitution must be numeric.");
                return;
            }

            entityData = new EntityData(
                tbName.Text,
                str,
                dex,
                cun,
                wil,
                mag,
                con,
                tbHealth.Text,
                tbStamina.Text,
                tbMana.Text);

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

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

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

        #endregion
    }
}
```

What the new code is doing is first wiring an event handler for the **FormClosing** event. In that handler I check to see if the reason for the form closing is **UserClosing**. That means the form is being close by the X, hitting ALT+F4, or code calling the **Close** method. If it is, I cancel the event. Then, before calling the **Close** method at the end of the click event handlers for the buttons I remove the subscription to the **FormClosing** event.

Now, I'm going to finish coding **FormClasses**. Right click **FormClasses** in the solution explorer and select **View Code**. I'm going to first add the logic for the **Delete** button. Change the **btnDelete_Click** method of **FormClasses** to the following. Also, add the following using statement with the other using statements at the beginning of the code.

```
using System.IO;

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);
            entityDataManager.EntityData.Remove(entity);

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

You first want to check to see that an item in the list box is selected by check to be sure it is not null. I then get the selected item as a string. Strings are separated by a comma so I call the **Split** method of the string class passing in a comma. The first string in the array is the name so I call the **Trim** method on **parts[0]** to remove any white space. I display a message box asking to make sure that you want to delete the entity and capture the result. If the result was yes I remove the entry form the list box, I then remove it from the entity data manager. I then check to see if a file for the entity exists, if it does I delete it.

To edit an item you follow somewhat the same process. Change the **btnEdit_Click** method to the following.

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

        using (FormEntityData frmEntityData = new FormEntityData())
```

```
        {
            frmEntityData.EntityData = data;
            frmEntityData.ShowDialog();

            if (frmEntityData.EntityData == null)
                return;

            if (frmEntityData.EntityData.EntityName == entity)
            {
                entityDataManager.EntityData[entity] = frmEntityData.EntityData;
                FillListBox();
                return;
            }

            newData = frmEntityData.EntityData;
        }

        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 (entityDataManager.EntityData.ContainsKey(newData.EntityName))
        {
            MessageBox.Show("Entry already exists. Use Edit to modify the entry.");
            return;
        }

        lbDetails.Items.Add(newData);
        entityDataManager.EntityData.Add(newData.EntityName, newData);
    }
}
```

You first want to check to see that an item in the list box is selected by check to be sure it is not null. I then get the selected item as a string. Strings are separated by a comma so I call the **Split** method of the string class passing in a comma. The first string in the array is the name so I call the **Trim** method on **parts[0]** to remove any white space. I then get the **EntityData** for the selected entity and set a local variable to hold the new data of the entity. That is followed by a using statement that I use to create the **FormEntityData** form for editing **EntityData** objects. I set the **EntityData** property of the form to be the currently selected object and then call the **ShowDialog** method. If the **EntityData** property is null after the **ShowDialog** method was called then the Cancel button was clicked and I exit the method. If the **EntityName** proeprty of the **EntityData** object is the same as the **entity** variable then the name of the **EntityData** object didn't change and it is safe to assign it to the entry at **entity** in the entity data manager. I then call the **FillListBox** method to update the list box. I then return out of the method. Before leaving the using statement for the form I set the **newData** variable to be the **EntityData** property of the form. I display a message box asking if the user wants to add a new entry for the **EntityData** object. If the result is No I exit the method. If there exists an entry in the entity data manager already I display a message box and exit the method. I finally add the new object to the list box and add the entry to the entity data manager.

The logic for the other forms that display the list is the same as **FormClasses**. Some of the logic for the forms creating and editing individual objects is similar to **FormEntityData**. The implementation is a little different but that is because the data on the forms is a little different. Before you can work on the forms that hold the list of objects you need to do the logic for the forms for creating new objects and editing existing objects.

Let's get started with **FormArmorDetails**. First right click it in the solution explorer and select **View Designer**. Click on the title bar. Set the **ControlBox** property to **False** and the **StartPosition** property to **CenterParent**. Right click it again and select **View Code**. This is the code for that form.

```
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.CharacterClasses;

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

        ArmorData armor = null;

        #endregion

        #region Property Region

        public ArmorData Armor
        {
            get { return armor; }
            set { armor = value; }
        }

        #endregion

        #region Constructor Region

        public FormArmorDetails()
        {
            InitializeComponent();

            this.Load += new EventHandler(FormArmorDetails_Load);
            this.FormClosing += new FormClosingEventHandler(FormArmorDetails_FormClosing);

            btnMoveAllowed.Click += new EventHandler(btnMoveAllowed_Click);
            btnRemoveAllowed.Click += new EventHandler(btnRemoveAllowed_Click);
            btnOK.Click += new EventHandler(btnOK_Click);
            btnCancel.Click += new EventHandler(btnCancel_Click);
        }

        #endregion

        #region Event Handler Region

        void FormArmorDetails_Load(object sender, EventArgs e)
        {
            foreach (string s in FormDetails.EntityDataManager.EntityData.Keys)
                lbClasses.Items.Add(s);

            foreach (ArmorLocation location in Enum.GetValues(typeof(ArmorLocation)))
                cboArmorLocation.Items.Add(location);

            cboArmorLocation.SelectedIndex = 0;

            if (armor != null)
            {
                tbName.Text = armor.Name;
```

```
            tbType.Text = armor.Type;
            mtbPrice.Text = armor.Price.ToString();
            nudWeight.Value = (decimal)armor.Weight;
            cboArmorLocation.SelectedIndex = (int)armor.ArmorLocation;
            mtbDefenseValue.Text = armor.DefenseValue.ToString();
            mtbDefenseModifier.Text = armor.DefenseModifier.ToString();

            foreach (string s in armor.AllowableClasses)
            {
                if (lbClasses.Items.Contains(s))
                    lbClasses.Items.Remove(s);

                lbAllowedClasses.Items.Add(s);
            }
        }
    }

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

    void btnMoveAllowed_Click(object sender, EventArgs e)
    {
        if (lbClasses.SelectedItem != null)
        {
            lbAllowedClasses.Items.Add(lbClasses.SelectedItem);
            lbClasses.Items.RemoveAt(lbClasses.SelectedIndex);
        }
    }

    void btnRemoveAllowed_Click(object sender, EventArgs e)
    {
        if (lbAllowedClasses.SelectedItem != null)
        {
            lbClasses.Items.Add(lbAllowedClasses.SelectedItem);
            lbAllowedClasses.Items.RemoveAt(lbAllowedClasses.SelectedIndex);
        }
    }

    void btnOK_Click(object sender, EventArgs e)
    {
        int price = 0;
        float weight = 0f;
        int defVal = 0;
        int defMod = 0;

        if (string.IsNullOrEmpty(tbName.Text))
        {
            MessageBox.Show("You must enter a name for the item.");
            return;
        }

        if (!int.TryParse(mtbPrice.Text, out price))
        {
            MessageBox.Show("Price must be an integer value.");
            return;
        }

        weight = (float)nudWeight.Value;

        if (!int.TryParse(mtbDefenseValue.Text, out defVal))
        {
            MessageBox.Show("Defense valule must be an interger value.");
            return;
        }
```

```
            if (!int.TryParse(mtbDefenseModifier.Text, out defMod))
            {
                MessageBox.Show("Defense valule must be an interger value.");
                return;
            }

            List<string> allowedClasses = new List<string>();

            foreach (object o in lbAllowedClasses.Items)
                allowedClasses.Add(o.ToString());

            armor = new ArmorData();
            armor.Name = tbName.Text;
            armor.Type = tbType.Text;
            armor.Price = price;
            armor.Weight = weight;
            armor.ArmorLocation = (ArmorLocation)cboArmorLocation.SelectedIndex;
            armor.DefenseValue = defVal;
            armor.DefenseModifier = defMod;
            armor.AllowableClasses = allowedClasses.ToArray();

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

        void btnCancel_Click(object sender, EventArgs e)
        {
            armor = null;
            this.FormClosing -= FormArmorDetails_FormClosing;
            this.Close();
        }

        #endregion
    }
}
```

This code should look a little familiar from **FormEntityData**. Some of it is new though. There are using statements to bring classes for our **RpgLibrary** into scope. There is a field of type **ArmorData** to hold the armor created or edited. There is a public property to expose the **ArmorData** field as well.

The constructor wires a few event handlers. There are handlers for the **Load** event of the form and the **FormClosing** event, like in **FormEntityData**. There are handlers for the **Click** events of **btnOK** and **btnCancel** as well, again just like **FormEntityData**. There are two new handlers though. They are for **btnMoveAllowed** and **btnRemoveAllowed**. When **btnMoveAllowed** is clicked the currently selected item in **lbClasses** will be moved to **lbAllowedClasses**. The other button, **btnRemoveAllowed**, works in reverse. It will move the currently selected item in **lbAllowedClasses** back to **lbClasses**.

In the **Load** event handler for the form I loop through all if the keys in the **EntityDataManager**. The keys are then added to the items in **lbClasses**. I also fill the combo box with the items from the enum **ArmorLocation**. I use the **GetValues** method to get the values. I also set the **SelectedIndex** of the combo box to be the first item, at index 0. It then checks to see if the **armor** field is not null, meaning the form is being opened to edit an armor. If it is not I set the values of the controls on the form. The text boxes have their **Text** properties set to the appropriate field of the **ArmorData** field. The masked text boxes have their **Text** properties set to the appropriate value using the **ToString** method. For the **nudWeight** I set the **Value** property to the **Weight** field by casting it to a **decimal**. For the combo box I set the **SelectedIndex** property to the **ArmorLocation** field, casting it to an integer. The list boxes work a little differently. I loop through all of the classes in the array **AllowableClasses**. If the **Items** collection of **lbClasses** contains the value it is removed. If you try and remove a value that isn't in the collection you will generate an exception. I then add the value to the **Items** collection of the second list

box, **lbAllowedClasses**. The other two forms, **FormShieldDetails** and **FormWeaponDetails**, have basically the same code for their **Load** events, just appropriate to the item they are for.

The event handler for the **FormClosing** event is a duplicate from **FormEntityData**. It just cancels closing the form if the close reason is **UserClosing**. It was subscribed to in the constructor and will be unsubscribed from if creating the **ArmorData** is successful in the **Click** event of **btnOK** and in the **Click** event of **btnCancel**.

The code for the **Click** event handler of **btnMoveAllowed** handles moving the currently selected item from **lbClasses** to **lbAllowedClasses**. It checks to see if the **SelectedItem** property of **lbClasses** is not null. If it isn't then an item is selected and should be moved. I add the **SelectedItem** from **lbClasses** to **lbAllowedClasses**. I then use the **RemoveAt** method of the **Items** collection of **lbClasses** to remove the **SelectedIndex** of **lbClasses**.

The code for the **Click** event handler of **btnRemoveAllowed** works in reverse. It checks to see if the **SelectedItem** property of **lbAllowedClasses** is not null. If it has a value the **SelectedItem** is added to **lbClasses** and removed from **lbAllowedClasses**.

The event handler for the **Click** event of **btnOK** does a little validation on the form. It checks to make sure that the **Text** property **tbName** has a value. It then uses the **TryParse** method of the integer class to make sure that the masked text boxes' **Text** property have integer values. Creating the array of allowed classes takes a little work. I have a local variable of **List<string>** that will hold all of the items in **lbAllowedClasses**. Items in a list box's **Items** collection are stored as objects so there is a foreach loop looping through all of the objects in **Items**. I add them to the **allowedClasses** using the **ToString** method. I then create a new **ArmorData** object and assign the fields. I unsubscribe the **FormClosing** event and close the form.

The event handler for **btnCancel**'s **Click** event works just like on **FormEntityData**. I set the field to **null**, unsubscribe the event and close the form.

The other two details forms, **FormShieldDetails** and **FormWeaponDetails**, have the same form as **FormArmorDetails**. The difference is they work with shields and weapons respectively. I don't see a reason to go over the code in depth like this form. Right click **FormShieldDetails** and select **View Code**. This is the code for **FormShieldDetails**.

```
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 FormShieldDetails : Form
    {
        #region Field Region

        ShieldData shield;

        #endregion
```

```
        #region Property Region

        public ShieldData Shield
        {
            get { return shield; }
            set { shield = value; }
        }

        #endregion

        #region Constructor Region

        public FormShieldDetails()
        {
            InitializeComponent();

            this.Load += new EventHandler(FormShieldDetails_Load);
            this.FormClosing += new FormClosingEventHandler(FormShieldDetails_FormClosing);

            btnMoveAllowed.Click += new EventHandler(btnMoveAllowed_Click);
            btnRemoveAllowed.Click += new EventHandler(btnRemoveAllowed_Click);
            btnOK.Click += new EventHandler(btnOK_Click);
            btnCancel.Click += new EventHandler(btnCancel_Click);
        }

        #endregion

        #region Event Handler Region

        void FormShieldDetails_Load(object sender, EventArgs e)
        {
            foreach (string s in FormDetails.EntityDataManager.EntityData.Keys)
                lbClasses.Items.Add(s);

            if (shield != null)
            {
                tbName.Text = shield.Name;
                tbType.Text = shield.Type;
                mtbPrice.Text = shield.Price.ToString();
                nudWeight.Value = (decimal)shield.Weight;
                mtbDefenseValue.Text = shield.DefenseValue.ToString();
                mtbDefenseModifier.Text = shield.DefenseModifier.ToString();

                foreach (string s in shield.AllowableClasses)
                {
                    if (lbClasses.Items.Contains(s))
                        lbClasses.Items.Remove(s);

                    lbAllowedClasses.Items.Add(s);
                }
            }
        }

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

        void btnMoveAllowed_Click(object sender, EventArgs e)
        {
            if (lbClasses.SelectedItem != null)
            {
                lbAllowedClasses.Items.Add(lbClasses.SelectedItem);
                lbClasses.Items.RemoveAt(lbClasses.SelectedIndex);
            }
        }
```

```
        void btnRemoveAllowed_Click(object sender, EventArgs e)
        {
            if (lbAllowedClasses.SelectedItem != null)
            {
                lbClasses.Items.Add(lbAllowedClasses.SelectedItem);
                lbAllowedClasses.Items.RemoveAt(lbAllowedClasses.SelectedIndex);
            }
        }

        void btnOK_Click(object sender, EventArgs e)
        {
            int price = 0;
            float weight = 0f;
            int defVal = 0;
            int defMod = 0;

            if (string.IsNullOrEmpty(tbName.Text))
            {
                MessageBox.Show("You must enter a name for the item.");
                return;
            }

            if (!int.TryParse(mtbPrice.Text, out price))
            {
                MessageBox.Show("Price must be an integer value.");
                return;
            }

            weight = (float)nudWeight.Value;

            if (!int.TryParse(mtbDefenseValue.Text, out defVal))
            {
                MessageBox.Show("Defense valule must be an interger value.");
                return;
            }

            if (!int.TryParse(mtbDefenseModifier.Text, out defMod))
            {
                MessageBox.Show("Defense valule must be an interger value.");
                return;
            }

            List<string> allowedClasses = new List<string>();

            foreach (object o in lbAllowedClasses.Items)
                allowedClasses.Add(o.ToString());

            shield = new ShieldData();
            shield.Name = tbName.Text;
            shield.Type = tbType.Text;
            shield.Price = price;
            shield.Weight = weight;
            shield.DefenseValue = defVal;
            shield.DefenseModifier = defMod;
            shield.AllowableClasses = allowedClasses.ToArray();

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

        void btnCancel_Click(object sender, EventArgs e)
        {
            shield = null;
            this.FormClosing -= FormShieldDetails_FormClosing;
            this.Close();
        }

        #endregion
    }
```

```
}
```

Nothing really new there, if anything it is a little simpler than **FormArmorDetails** as you don't have to worry about the location of a shield. **FormWeaponDetails** is basically the same as well. It just works with a weapon rather than armor. Right click **FormWeaponDetails** and select **View Code**. This is the code.

```
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 FormWeaponDetails : Form
    {
        #region Field Region

        WeaponData weapon = null;

        #endregion

        #region Property Region

        public WeaponData Weapon
        {
            get { return weapon; }
            set { weapon = value; }
        }

        #endregion

        #region Constructor Region

        public FormWeaponDetails()
        {
            InitializeComponent();

            this.Load += new EventHandler(FormWeaponDetails_Load);
            this.FormClosing += new FormClosingEventHandler(FormWeaponDetails_FormClosing);

            btnMoveAllowed.Click += new EventHandler(btnMoveAllowed_Click);
            btnRemoveAllowed.Click += new EventHandler(btnRemoveAllowed_Click);
            btnOK.Click += new EventHandler(btnOK_Click);
            btnCancel.Click += new EventHandler(btnCancel_Click);
        }

        #endregion

        #region Event Handler Region

        void FormWeaponDetails_Load(object sender, EventArgs e)
        {
            foreach (string s in FormDetails.EntityDataManager.EntityData.Keys)
                lbClasses.Items.Add(s);

            foreach (Hands location in Enum.GetValues(typeof(Hands)))
                cboHands.Items.Add(location);

            cboHands.SelectedIndex = 0;
```

```
            if (weapon != null)
            {
                tbName.Text = weapon.Name;
                tbType.Text = weapon.Type;
                mtbPrice.Text = weapon.Price.ToString();
                nudWeight.Value = (decimal)weapon.Weight;
                cboHands.SelectedIndex = (int)weapon.NumberHands;
                mtbAttackValue.Text = weapon.AttackValue.ToString();
                mtbAttackModifier.Text = weapon.AttackModifier.ToString();
                mtbDamageValue.Text = weapon.DamageValue.ToString();
                mtbDamageModifier.Text = weapon.DamageModifier.ToString();

                foreach (string s in weapon.AllowableClasses)
                {
                    if (lbClasses.Items.Contains(s))
                        lbClasses.Items.Remove(s);

                    lbAllowedClasses.Items.Add(s);
                }
            }
        }

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

        void btnMoveAllowed_Click(object sender, EventArgs e)
        {
            if (lbClasses.SelectedItem != null)
            {
                lbAllowedClasses.Items.Add(lbClasses.SelectedItem);
                lbClasses.Items.RemoveAt(lbClasses.SelectedIndex);
            }
        }

        void btnRemoveAllowed_Click(object sender, EventArgs e)
        {
            if (lbAllowedClasses.SelectedItem != null)
            {
                lbClasses.Items.Add(lbAllowedClasses.SelectedItem);
                lbAllowedClasses.Items.RemoveAt(lbAllowedClasses.SelectedIndex);
            }
        }

        void btnOK_Click(object sender, EventArgs e)
        {
            int price = 0;
            float weight = 0f;
            int attVal = 0;
            int attMod = 0;
            int damVal = 0;
            int damMod = 0;

            if (string.IsNullOrEmpty(tbName.Text))
            {
                MessageBox.Show("You must enter a name for the item.");
                return;
            }

            if (!int.TryParse(mtbPrice.Text, out price))
            {
                MessageBox.Show("Price must be an integer value.");
                return;
            }

            weight = (float)nudWeight.Value;
```

```
            if (!int.TryParse(mtbAttackValue.Text, out attVal))
            {
                MessageBox.Show("Attack value must be an interger value.");
                return;
            }

            if (!int.TryParse(mtbAttackModifier.Text, out attMod))
            {
                MessageBox.Show("Attack value must be an interger value.");
                return;
            }

            if (!int.TryParse(mtbDamageValue.Text, out damVal))
            {
                MessageBox.Show("Damage value must be an interger value.");
                return;
            }

            if (!int.TryParse(mtbDamageModifier.Text, out damMod))
            {
                MessageBox.Show("Damage value must be an interger value.");
                return;
            }

            List<string> allowedClasses = new List<string>();

            foreach (object o in lbAllowedClasses.Items)
                allowedClasses.Add(o.ToString());

            weapon = new WeaponData();
            weapon.Name = tbName.Text;
            weapon.Type = tbType.Text;
            weapon.Price = price;
            weapon.Weight = weight;
            weapon.AttackValue = attVal;
            weapon.AttackModifier = attMod;
            weapon.DamageValue = damVal;
            weapon.DamageModifier = damMod;
            weapon.AllowableClasses = allowedClasses.ToArray();

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

        void btnCancel_Click(object sender, EventArgs e)
        {
            weapon = null;
            this.FormClosing -= FormWeaponDetails_FormClosing;
            this.Close();
        }

        #endregion
    }
}
```

I don't think there is anything that needs explaining. The only difference is a few variable names and instead of using **ArmorLocation** to fill the combo box use **Hands**.

I'm going to end this tutorial here and add a B part to it. Instead of posting the A part before finishing the B part, I'm going to finish the B part and post them both. I encourage you to visit the news page of my site, XNA Game Programming Adventures, for the latest news on my tutorials.

Good luck in your game programming adventures!

Jamie McMahon