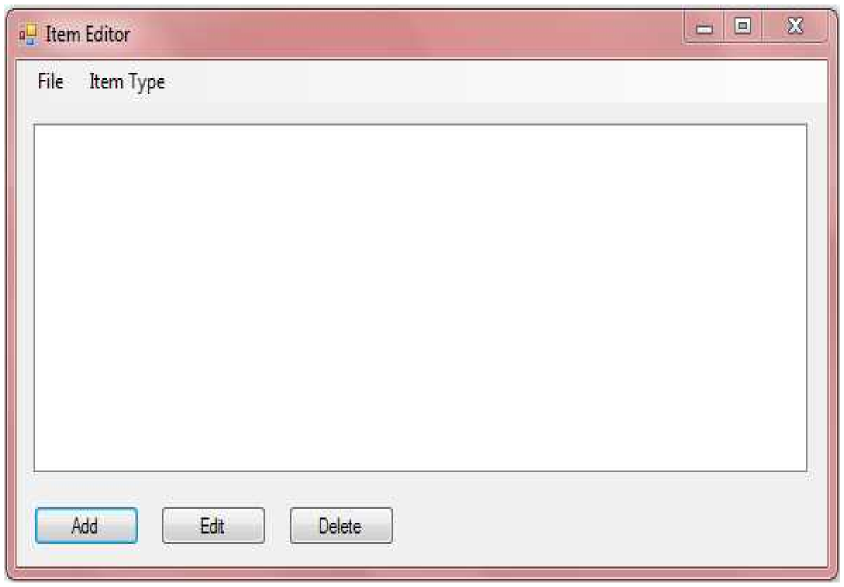
Creating a Role Playing Game with XNA Game Studio
Part 50
Items - Part 4a

To follow along with this tutorial you will have to have read the previous tutorials to understand much of what it going on. You can find a list of tutorials here: XNA Role Playing Game Tutorials You will also find the latest version of the project on the web site on that page. Unlike other tutorials, this tutorial is a stand alone tutorial other than requiring 4 classes whose code I've included in the tutorial.

In this tutorial I will be continuing on with picking up items in the game, this is the **a** part of the tutorial, the tutorial was getting very long so I decided to split it into two pieces. I had a request for being able to read in items at run time from XML files rather than having them hard coded. It was something that I was planning on doing anyway so I thought I would do it in this tutorial rather than the next one. Instead of creating the XML files by hand I decided to make an editor for this. I asked on my forum and two people suggested that they would like a tutorial on this so I will write that in this tutorial as well. In the **b** part of the tutorial I will cover actually writing items from the editor to XML and reading them back in again. I will also add reading them in from the game.

I'm not going to include the item editor as part of the game solution. I'm just going to create it as a project of its own. Load Visual Studio, or Visual C# Express, create a new Windows Form application and call it **ItemEditor**. When the project is complete click **Form1.cs** in the solution explorer. In the properties window below the solution explorer. Click **Form1.cs** and rename it to **FormMain**. Select **Yes** in the dialog that comes up. I added five controls to the form, a **Menu Strip**, **List Box**, and three **Buttons**. This is what the final form looks like.



The way the editor works is you select an **Item Type**. The list box will display all of the items of that type. You can add items, edit items, and delete items. I will start with setting the properties of the main form and then the controls on the form. The easiest way is to just list the properties to be changed and their values. Set the following properties as follows.

- **Text - Item Editor**
- **Size - 590, 325**

Now open the toolbox and pin it for now. Drag a **Menu Strip** onto the form. In the first box that says **Type Here** type **&File**. In the boxes below **&File**, add **&Open**, **&Save**, **-**, and **E&xit**. In the **Type Here** box beside **&File** type **&Item Type**. Below that you will add **&Weapon**, **&Armor**, and **&Shield**. Go back to **&Weapon** and set **CheckState** property to **Checked**.

Now drag a **List Box** control onto the form. Setting the anchor property like I have will have the list box size and the form resizes. Set the following properties:

- **(Name) - lbItems**
- **Anchor - Top, Bottom, Left, Right**
- **Location - 10, 35**
- **Size - 550, 199**

You can now drag three **Buttons** onto the form. Arrange them under the **List Box** in a row. For the first button, the one with the **Add** text. Set the following properties, if you like the location of yours you don't have to set that property. Setting the **Anchor** property of the button has it stay at the bottom of the form as it resizes.

- **(Name) - btnAdd**
- **Anchor - Bottom, Left**
- **Location - 10, 250**
- **Text - Add**

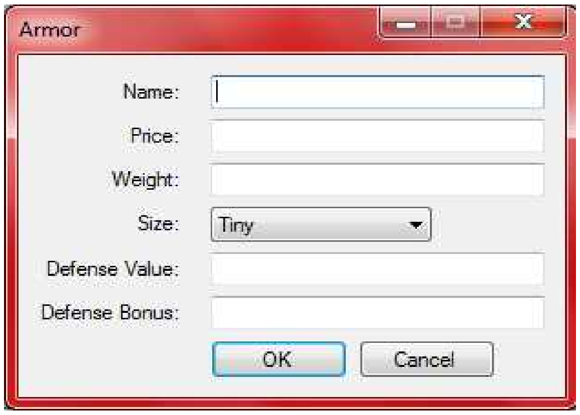
These are the properties for the second button, the one with the **Edit** text.

- **(Name) - btnEdit**
- **Anchor - Bottom, Left**
- **Location - 100, 250**
- **Text - Edit**

These are the properties for the last button.

- **(Name) - btnDelete**
- **Anchor - Bottom, Left**
- **Location - 190, 250**
- **Text - Delete**

I made three other forms for adding and editing items, one for each type of item. They are all very much the same so I will take a short cut when it comes to designing them. I will start with the form for adding or editing armor. On the form are six **Labels**, five **Text Boxes**, one **Combo Box**, and two **Buttons.** Right click the project, select **Add** and the **Windows Form**. Name the form **ArmorForm**. On the next page is what the final form looks like.



Set the following properties for the form.

- **FormBorderStyle - FixedDialog**
- **MaximizeBox - False**
- **Size - 300, 230**
- **StartPosition - CenterParent**
- **Text - Armor**

You can now drag the **Labels** onto the form one at a time. I will give you the properties of all of the **Labels**. Their name properties will be the same as when they were dragged onto the form.

label1
- **Location - 54, 15**
- **Text - Name**

label2
- **Location - 58, 41**
- **Text - Price**

label3
- **Location - 48, 67**
- **Text - Weight**

label4
- **Location - 62, 93**
- **Text - Size**

label5
- **Location - 12, 120**
- **Text - Defense Value**

label6

- **Location - 9, 146**
- **Text - Defense Bonus**

The next thing to do is add the **Text Boxes** to the form. Like I did for the **Labels**, I will give you the properties for the **Text Boxes** as they are dragged onto the form. You can save a little work after you have dragged the first **Text Box** onto the form by selecting it with the mouse, holding down the control key and dragging it to about the right position. This will replicate the **Text Box** with some of the same attributes.

textbox1

- **(Name) - tbName**
- **Location - 105, 12**
- **Size 183, 20**
- **TabIndex - 0**

textbox2

- **(Name) - tbPrice**
- **Location - 105, 38**
- **Size 183, 20**
- **TabIndex - 1**

textbox3

- **(Name) - tbWeight**
- **Location - 105, 64**
- **Size 183, 20**
- **TabIndex - 2**

textbox4

- **(Name) - tbDefenseValue**
- **Location - 105, 117**
- **Size 183, 20**
- **TabIndex - 4**

textbox5

- **(Name) - tbDefenseBonus**
- **Location - 105, 143**
- **Size 183, 20**
- **TabIndex - 5**

Now, drag a **Combo Box** onto the form. At first it won't look like the one on the image, after changing a property the appearance will change. Set the following properties for the **Combo Box**.

comboBox1

- **(Name) - cbSize**
- **DropDownStyle - DropDownList**
- **Location - 105, 90**
- **Size - 121, 21 (**You can make it the same size as the Text Boxes if you like it that way**.)**

- **TabIndex - 3**

That leaves the two **Buttons**, just drag two buttons on to the form at set their properties as follows.

button1
- **(Name) - btnOK**
- **Location - 105, 169**
- **TabIndex - 6**
- **Text - OK**

button2
- **(Name) - btnCancel**
- **Location - 186, 169**
- **TabIndex - 7**
- **Text - Cancel**

There is just one more thing to do in design mode. I want to set it so that when you press the enter key it acts as if you clicked the **OK** button and if you press the escape key it acts like you clicked the **Cancel** button. Set the following properties on **FormArmor.**

- **AcceptButton - btnOK**
- **CancelButton - btnCancel**

Now I'm going to take a short cut. Add a new **Windows Form** to the solution, like you did before, called **FormShield**. Go to **FormArmor** and while holding down the shift key, click all of the controls on the form. Right click a control on the form and select **Copy**. Go back to **FormShield** and make it a little bigger. Right click in the middle of the form and select **Paste**. This will copy all of the controls on **FormArmor** and paste them into **FormShield** with the same names and other properties. Select one of the controls and drag them until the fit nicely on the form. Click the top of **FormShield** and set the following properties.

- **AcceptButton - btnOK**
- **CancelButton - btnCancel**
- **FormBorderStyle - FixedDialog**
- **MaximizeBox - False**
- **Size - 300, 230**
- **StartPosition - CenterParent**
- **Text - Shield**

Add one more **Windows Form** to the solution, again like before, called **FormWeapon**. Make the form bigger again and paste the controls onto the form. Select one of the controls and drag them until they are in the same position. Set the following properties of **FormWeapon**.

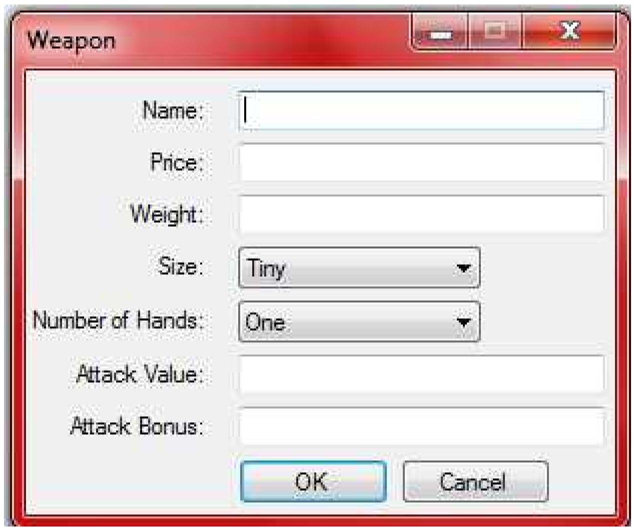
- **AcceptButton - btnOK**
- **CancelButton - btnCancel**

- **FormBorderStyle - FixedDialog**
- **MaximizeBox - False**
- **Size - 300, 250**
- **StartPosition - CenterParent**
- **Text - Weapon**

Change the text of the **Labels** that say **Defense Value** and **Defense Bonus** to say **Attack Value** and **Attack Bonus**. Change the name of **tbDefenseValue** to **tbAttackValue** and **tbDefenseBonus** to **tbAttackBonus**. Now, select those **Labels** and **Text Boxes**, as well as the **Buttons** and drag them down to make room for another **Label** and **Combo Box**. Drag a **Label** and **Combo Box** onto the form so they line up with the other **Label** and **Combo Box** above them. Set the **Text** property of the **Label** to **Number of Hands** and then align it with the other labels. Now set the following properties of the **Combo Box**.

- **(Name) - cbHands**
- **DropDownStyle - DropDownList**

You want the finished form to resemble the following.



There is one more thing to do before coding the forms, you need the classes for weapons, armor, and shields in the project. There is a minor change to each class. I decided to include the code for the entire classes as they are rather short. Alternatively right click your solution, select **Add**, and then **Existing Item**. Then navigate to the **ItemClasses** folder in your game is and add the **BaseItem, Armor**, **Shield**, and **Weapon** classes to the project.

BaseItem.cs

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

namespace ItemEditor
{
    public enum ItemSize { Tiny, Small, Medium, Large };
    public abstract class BaseItem
    {
        string name;
        int price;
        int weight;
        ItemSize itemSize;

        public string Name
        {
            get { return name; }
            protected set { name = value; }
        }

        public int Price
        {
            get { return price; }
            protected set { price = value; }
        }

        public int Weight
        {
            get { return weight; }
            protected set { weight = value; }
        }

        public ItemSize Size
        {
            get { return itemSize; }
            protected set { itemSize = value; }
        }

        public BaseItem(string name, int price, int weight, ItemSize size)
        {
            Name = name;
            Price = price;
            Weight = weight;
            Size = size;
        }
    }
}
```

Armor.cs

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

namespace ItemEditor
{
    public class Armor : BaseItem
    {
        int defenseValue;
        int defenseBonus;

        public int DefenseValue
        {
            get { return defenseValue; }
            protected set { defenseValue = value; }
        }

        public int DefenseBonus
        {
            get { return defenseBonus; }
            protected set { defenseBonus = value; }
        }

        public Armor(
                string armorName,
                int price,
                int weight,
                ItemSize size,
                int defenseValue,
                int defenseBonus)
            : base(armorName, price, weight, size)
        {
            DefenseValue = defenseValue;
            DefenseBonus = defenseBonus;
        }
    }
}
```

Shield.cs

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

namespace ItemEditor
{
    public class Shield : BaseItem
    {
        int defenseValue;
        int defenseBonus;

        public int DefenseValue
        {
            get { return defenseValue; }
            protected set { defenseValue = value; }
        }

        public int DefenseBonus
        {
            get { return defenseBonus; }
            protected set { defenseBonus = value; }
        }

        public Shield(
                string shieldName,
                int price,
                int weight,
                ItemSize size,
                int defenseValue,
                int defenseBonus)
            : base(shieldName, price, weight, size)
        {
            DefenseValue = defenseValue;
            DefenseBonus = defenseBonus;
        }
    }
}
```

Weapon.cs

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

namespace ItemEditor
{
    public enum Hands { One, Two }

    public class Weapon : BaseItem
    {
        Hands hands;
        int attackValue;
        int attackBonus;

        public Hands NumberHands
        {
            get { return hands; }
            protected set { hands = value; }
        }

        public int AttackValue
        {
            get { return attackValue; }
            protected set { attackValue = value; }
        }

        public int AttackBonus
        {
            get { return attackBonus; }
            protected set { attackBonus = value; }
        }

        public Weapon(
                string weaponName,
                int price,
                int weight,
                ItemSize size,
                Hands hands,
                int attackValue,
                int attackBonus)
            : base(weaponName, price, weight, size)
        {
            NumberHands = hands;
            AttackValue = attackValue;
            AttackBonus = attackBonus;
        }
    }
}
```

The change was just the name space. Also, the **Shield** class was made public. Now it is time to start coding the forms. I will start with **FormArmor**. Right click **FormArmor** in the solution explorer and select **View Code**. As I always do with a lot of new code, show you the code and then explain it afterwords. This is the code for **FormArmor**.

```
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 ItemEditor
{
    public partial class FormArmor : Form
    {
        Armor armor;
        bool okPressed = false;

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

        public bool OKPressed
        {
            get { return okPressed; }
        }

        public FormArmor()
        {
            InitializeComponent();

            string[] sizes = Enum.GetNames(typeof(ItemSize));
            foreach (string size in sizes)
                cbSize.Items.Add(size);
            cbSize.SelectedIndex = 0;

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

        void FormArmor_Load(object sender, EventArgs e)
        {
            if (armor != null)
            {
                tbName.Text = armor.Name;

                tbPrice.Text = armor.Price.ToString();
                tbWeight.Text = armor.Weight.ToString();

                cbSize.SelectedIndex = (int)armor.Size;
                cbSize.Text = (string)cbSize.SelectedItem;

                tbDefenseBonus.Text = armor.DefenseBonus.ToString();
                tbDefenseValue.Text = armor.DefenseValue.ToString();
            }
        }


        private void btnOK_Click(object sender, EventArgs e)
        {
            int price = 0;
            int weight = 0;
            int defenseValue = 0;
            int defenseBonus = 0;

            if (string.IsNullOrEmpty(tbName.Text))
            {
                MessageBox.Show("You must enter a weapon name!");
                return;
            }
            if (!Int32.TryParse(tbPrice.Text, out price))
            {
                MessageBox.Show("Price must be an integer value!");
                return;
            }
            if (!Int32.TryParse(tbWeight.Text, out weight))
            {
                MessageBox.Show("Weight must be an integer value!");
                return;
            }
            if (!Int32.TryParse(tbDefenseValue.Text, out defenseValue))
            {
                MessageBox.Show("Defense value must be an integer value!");
                return;
            }
            if (!Int32.TryParse(tbDefenseBonus.Text, out defenseBonus))
            {
                MessageBox.Show("Defense bonus must be an integer value!");
                return;
            }

            armor = new Armor(
                    tbName.Text,
                    price,
                    weight,
                    (ItemSize)cbSize.SelectedIndex,
                    defenseValue,
                    defenseBonus);

            okPressed = true;
            this.Close();
        }

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

There are two fields and properties in this class, and the other two classes, the properties in the other two classes are specific to the type of item. On this form the field **armor** will hold the armor that is being added or edited. The other field **okPressed** will be used to tell if the user has pressed the OK button to close the form or not. The property **Armor** is both get and set so that when a form is created and it is to be edited you can set the previous values of the armor. The other property **OKPressed** will be so that you can tell if the user pressed the OK button on the form to close it.

In the constructor, I wired the events for the form. I thought it would be easier to do it through code that explaining how to do it through the designer. Remember when you type the += that you can press the tab key twice to have the skeleton code generated for you. You should also remember that the controls on the form do not exist until after the call to **InitializeComponent**. Before wiring the events I fill **cbSize** with the values of the enumeration **ItemSize**. I do that using the **GetNames** method of the **Enum** class. In a foreach loop, I loop through of the strings that are returned and add them to the item collection. I then set the **SelectedIndex** property to 0 so that the first item will be selected in **cbSize.** The only events that I was interested in were the **Load** event of the form and the **Click** event of the buttons.

The **Load** event of the form is called when the form is first loaded. What I do is check to see if the **armor** field was set using the **Armor** property. If it was I set the initial values of the text boxes and the combo box.

In the **Click** event for the **OK** button there a few local variables that will hold the price, weight, defense value and defense bonus for the armor. There are then a series of if statements to validate the form. The first if statement checks to see that there is a name for the armor. The next one checks to make sure there is an integer value for the price. Using the **TryParse** method the second parameter will hold the value if the conversion is successful. **TryParse** will not throw an exception if the conversion fails and returns true if the conversion is successful. I also convert and check the weight, defense value and defense bonus the same way. If all of the conversions are successful, I create a new instance, set **okPressed** to true and close the form. To create the instance I used the **Text** property of **tbName**, the variables **price, weight, defenseValue,** and **defenseBonus**. For the size I cast the **SelectedIndex** of the combo box to be **ItemSize**.

The **Click** event for the **Cancel** button is simple. It just sets **okPressed** to false and closes the form. The code for **FormShield** is the same as for **FomrArmor**. The only difference it it works for shields and not armor. So, I will just give you 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;

namespace ItemEditor
{
    public partial class FormShield : Form
    {
        Shield shield;
        bool okPressed = false;

        public Shield Shield
        {
```

```
        get { return shield; }
        set { shield = value; }
    }

    public bool OKPressed
    {
        get { return okPressed; }
    }

    public FormShield()
    {
        InitializeComponent();

        string[] sizes = Enum.GetNames(typeof(ItemSize));
        foreach (string size in sizes)
            cbSize.Items.Add(size);
        cbSize.SelectedIndex = 0;

        this.Load += new EventHandler(FormShield_Load);
        btnOK.Click += new EventHandler(btnOK_Click);
        btnCancel.Click += new EventHandler(bntCancel_Click);
    }

    void FormShield_Load(object sender, EventArgs e)
    {
        if (shield != null)
        {
            tbName.Text = shield.Name;

            tbPrice.Text = shield.Price.ToString();
            tbWeight.Text = shield.Weight.ToString();

            cbSize.SelectedIndex = (int)shield.Size;
            cbSize.Text = (string)cbSize.SelectedItem;

            tbDefenseBonus.Text = shield.DefenseBonus.ToString();
            tbDefenseValue.Text = shield.DefenseValue.ToString();
        }
    }

    void btnOK_Click(object sender, EventArgs e)
    {
        int price = 0;
        int weight = 0;
        int defenseValue = 0;
        int defenseBonus = 0;

        if (string.IsNullOrEmpty(tbName.Text))
        {
            MessageBox.Show("You must enter a weapon name!");
            return;
        }
        if (!Int32.TryParse(tbPrice.Text, out price))
        {
            MessageBox.Show("Price must be an integer value!");
            return;
        }
        if (!Int32.TryParse(tbWeight.Text, out weight))
        {
                MessageBox.Show("Weight must be an integer value!");
                return;
            }
            if (!Int32.TryParse(tbDefenseValue.Text, out defenseValue))
            {
                MessageBox.Show("Defense value must be an integer value!");
                return;
            }
            if (!Int32.TryParse(tbDefenseBonus.Text, out defenseBonus))
            {
                MessageBox.Show("Defense bonus must be an integer value!");
                return;
            }

            shield = new Shield(
                    tbName.Text,
                    price,
                    weight,
                    (ItemSize)cbSize.SelectedIndex,
                    defenseValue,
                    defenseBonus);

            okPressed = true;
            this.Close();
        }

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

The code for **FormWeapon** is basically the same as well. The change is in the constructor and in the code for the **Click** event of **btnOK**. In the constructor I fill **cbHands** in the same fashion I did for the sizes. In the **Click** event of **btnOK** I use variables **attackValue** and **attackBonus** instead of **defenseValue** and **defenseBonus**. For the **Hands** parameter, I cast the **SelectedIndex** of **cbHands** to **Hands**. This is the code for **FormWeapon**.

```
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 ItemEditor
{
    public partial class FormWeapon : Form
    {
        Weapon weapon;
        bool okPressed = false;

        public Weapon Weapon
        {
```

```
        get { return weapon; }
        set { weapon = value; }
    }

    public bool OKPressed
    {
        get { return okPressed; }
    }

    public FormWeapon()
    {
        okPressed = false;

        InitializeComponent();

        string[] sizes = Enum.GetNames(typeof(ItemSize));

        foreach (string size in sizes)
            cbSize.Items.Add(size);
        cbSize.SelectedIndex = 0;

        string[] hands = Enum.GetNames(typeof(Hands));

        foreach (string hand in hands)
            cbHands.Items.Add(hand);
        cbHands.SelectedIndex = 0;

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

    void FormWeapon_Load(object sender, EventArgs e)
    {
        if (weapon != null)
        {
            tbName.Text = weapon.Name;
            tbPrice.Text = weapon.Price.ToString();
            tbWeight.Text = weapon.Weight.ToString();

            cbSize.SelectedIndex = (int)weapon.Size;
            cbSize.Text = (string)cbSize.SelectedItem;

            cbHands.SelectedIndex = (int)weapon.NumberHands;
            cbHands.Text = (string)cbHands.SelectedItem;

            tbAttackBonus.Text = weapon.AttackBonus.ToString();
            tbAttackValue.Text = weapon.AttackValue.ToString();
        }
    }

    private void btnOK_Click(object sender, EventArgs e)
    {
        int price = 0;
        int weight = 0;
        int attackValue = 0;
        int attackBonus = 0;

        if (string.IsNullOrEmpty(tbName.Text))
```

```
            {
                MessageBox.Show("You must enter a weapon name!");
                return;
            }
            if (!Int32.TryParse(tbPrice.Text, out price))
            {
                MessageBox.Show("Price must be an integer value!");
                return;
            }
            if (!Int32.TryParse(tbWeight.Text, out weight))
            {
                MessageBox.Show("Weight must be an integer value!");
                return;
            }
            if (!Int32.TryParse(tbAttackValue.Text, out attackValue))
            {
                MessageBox.Show("Attack value must be an integer value!");
                return;
            }
            if (!Int32.TryParse(tbAttackBonus.Text, out attackBonus))
            {
                MessageBox.Show("Attack bonus must be an integer value!");
                return;
            }

            weapon = new Weapon(
                    tbName.Text,
                    price,
                    weight,
                    (ItemSize)cbSize.SelectedIndex,
                    (Hands)cbHands.SelectedIndex,
                    attackValue,
                    attackBonus);

            okPressed = true;
            this.Close();
        }

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

FormMain is a little more complicated than the other forms. Also, before I get to it, I want to explain a little about how I plan to save items. I decided to have all items in one XML file. The file will have as its root node: **Items**. Inside the root node there will be several other nodes. For now there are three: **Weapons**, **Armors**, and **Shields**. Inside of the children there will be all of the items of that type. Each of the child nodes inside of them will be a specific item. Those nodes will have as children the name of the property for the item and as attributes the values. The exception is the value and bonus fields. They will be attributes of the same node. I also decided to use just one file name: **items.its**. I tried **itms** for the extension but that referenced something in iTunes.

```xml
<Items>
  <Weapons>
    <Weapon>
      <Name Value="dagger" />
      <Price Value="10" />
      <Weight Value="2" />
      <Size Value="Small" />
      <Hands Value="One" />
      <Attack Value="5" Bonus="0" />
    </Weapon>
  </Weapons>
  <Armors>
    <Armor>
      <Name Value="leather armor" />
      <Price Value="20" />
      <Weight Value="10" />
      <Size Value="Medium" />
      <Defense Value="10" Bonus="0" />
    </Armor>
  </Armors>
  <Shields>
    <Shield>
      <Name Value="buckler" />
      <Price Value="10" />
      <Weight Value="5" />
      <Size Value="Medium" />
      <Defense Value="10" Bonus="0" />
    </Shield>
  </Shields>
</Items>
```

It is a rather simple format and easy to remember. Now I will get to the code for the form. I will give you all of the code and explain it. I've tried to organize the code so that it will progress logically. I will get to loading and saving items in the **b** part of this tutorial.

```csharp
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.Xml;

namespace ItemEditor
{
    public partial class FormMain : Form
    {
        Dictionary<string, Weapon> weapons = new Dictionary<string, Weapon>();
        Dictionary<string, Armor> armors = new Dictionary<string, Armor>();
        Dictionary<string, Shield> shields = new Dictionary<string, Shield>();

        public FormMain()
        {
            InitializeComponent();

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

            exitToolStripMenuItem.Click += new EventHandler(exitToolStripMenuItem_Click);

            weaponToolStripMenuItem.Click += new EventHandler(weaponToolStripMenuItem_Click);
            armorToolStripMenuItem.Click += new EventHandler(armorToolStripMenuItem_Click);
            shieldToolStripMenuItem.Click += new EventHandler(shieldToolStripMenuItem_Click);

            lbItems.SelectionMode = SelectionMode.One;
        }

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

        void weaponToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (weaponToolStripMenuItem.CheckState == CheckState.Unchecked)
            {
                weaponToolStripMenuItem.CheckState = CheckState.Checked;
                armorToolStripMenuItem.CheckState = CheckState.Unchecked;
                shieldToolStripMenuItem.CheckState = CheckState.Unchecked;
                FillWeapons();
            }
        }

        void armorToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (armorToolStripMenuItem.CheckState == CheckState.Unchecked)
            {
                weaponToolStripMenuItem.CheckState = CheckState.Unchecked;
                armorToolStripMenuItem.CheckState = CheckState.Checked;
                shieldToolStripMenuItem.CheckState = CheckState.Unchecked;
                FillArmor();
            }
        }

        void shieldToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (shieldToolStripMenuItem.CheckState == CheckState.Unchecked)
            {
                weaponToolStripMenuItem.CheckState = CheckState.Unchecked;
                armorToolStripMenuItem.CheckState = CheckState.Unchecked;
                shieldToolStripMenuItem.CheckState = CheckState.Checked;
                FillShields();
            }
        }

        void btnAdd_Click(object sender, EventArgs e)
        {
            if (weaponToolStripMenuItem.CheckState == CheckState.Checked)
            {
                FormWeapon frmWeapon = new FormWeapon();
                frmWeapon.ShowDialog();

                if (frmWeapon.OKPressed)
                {
                    Weapon weapon = frmWeapon.Weapon;

                    if (weapons.ContainsKey(weapon.Name))
                    {
                        MessageBox.Show("There is already a weapon called " + weapon.Name);
                        return;
                    }

                    weapons.Add(weapon.Name, weapon);
                    string output = weapon.Name + ", " + weapon.Price.ToString() + ", ";
                    output += weapon.Weight.ToString() + ", " + weapon.Size.ToString() + ", ";
                    output += weapon.NumberHands.ToString()+", "+weapon.AttackValue.ToString() + ", ";
                    output += weapon.AttackBonus.ToString();
                    lbItems.Items.Add(output);
                }
                return;
            }
            if (armorToolStripMenuItem.CheckState == CheckState.Checked)
            {
                FormArmor frmArmor = new FormArmor();
                frmArmor.ShowDialog();
```

```
            if (frmArmor.OKPressed)
            {
                Armor newArmor = frmArmor.Armor;

                if (armors.ContainsKey(newArmor.Name))
                {
                    MessageBox.Show("There is already armor called " + newArmor.Name);
                    return;
                }

                armors.Add(newArmor.Name, newArmor);
                string output = newArmor.Name + ", " + newArmor.Price.ToString() + ", ";
                output += newArmor.Weight.ToString() + ", " + newArmor.Size.ToString() + ", ";
                output += newArmor.DefenseValue.ToString() + ", ";
                output += newArmor.DefenseBonus.ToString();
                lbItems.Items.Add(output);
            }
            return;
        }
        if (shieldToolStripMenuItem.CheckState == CheckState.Checked)
        {
            FormShield frmShield = new FormShield();
            frmShield.ShowDialog();

            if (frmShield.OKPressed)
            {
                Shield newShield = frmShield.Shield;

                if (armors.ContainsKey(newShield.Name))
                {
                    MessageBox.Show("There is already armor called " + newShield.Name);
                    return;
                }

                shields.Add(newShield.Name, newShield);
                string output = newShield.Name + ", " + newShield.Price.ToString() + ", ";
                output += newShield.Weight.ToString() + ", " + newShield.Size.ToString() + ", ";
                output += newShield.DefenseValue.ToString() + ", ";
                output += newShield.DefenseBonus.ToString();
                lbItems.Items.Add(output);
            }
            return;
        }
    }

    void btnEdit_Click(object sender, EventArgs e)
    {
        if (weaponToolStripMenuItem.CheckState == CheckState.Checked)
        {
            if (lbItems.SelectedIndex == -1)
                return;

            int itemNumber = lbItems.SelectedIndex;
            string lbItem = (string)lbItems.SelectedItem;
            string[] parts = lbItem.Split(',');
            string wpnName = parts[0].Trim();

            ItemSize wpnSize = (ItemSize)Enum.Parse(typeof(ItemSize), parts[3].Trim());
            Hands wpnHands = (Hands)Enum.Parse(typeof(Hands), parts[4].Trim());

            int wpnPrice;
            int wpnWeight;
            int wpnAttackValue;
            int wpnAttackBonus;

            Int32.TryParse(parts[1].Trim(), out wpnPrice);
            Int32.TryParse(parts[2].Trim(), out wpnWeight);
            Int32.TryParse(parts[5].Trim(), out wpnAttackValue);
            Int32.TryParse(parts[6].Trim(), out wpnAttackBonus);

            Weapon newWeapon = new Weapon(
                    wpnName,
                    wpnPrice,
                    wpnWeight,
                    wpnSize,
                    wpnHands,
                    wpnAttackValue,
                    wpnAttackBonus);

            FormWeapon frmWeapon = new FormWeapon();
            frmWeapon.Weapon = newWeapon;
            frmWeapon.ShowDialog();

            if (frmWeapon.OKPressed)
            {
                newWeapon = frmWeapon.Weapon;
                if (newWeapon.Name != wpnName)
                {
                    DialogResult result = MessageBox.Show(
                        "Proceeding will delete the old entry. Are you sure?",
                        "Warning",
                        MessageBoxButtons.YesNo);
                    if (result == DialogResult.No)
                        return;
                    weapons.Remove(wpnName);
                    weapons.Add(newWeapon.Name, newWeapon);
                }

                string output = newWeapon.Name + ", " + newWeapon.Price.ToString() + ", ";
                output += newWeapon.Weight.ToString() + ", " + newWeapon.Size.ToString() + ", ";
                output += newWeapon.NumberHands.ToString() + ", " +
                     newWeapon.AttackValue.ToString() + ", ";
                output += newWeapon.AttackBonus.ToString();
                weapons[newWeapon.Name] = newWeapon;
                lbItems.Items[itemNumber] = output;
            }
            return;
        }

        if (armorToolStripMenuItem.CheckState == CheckState.Checked)
        {
            if (lbItems.SelectedIndex == -1)
                return;
            int itemNumber = lbItems.SelectedIndex;
            string lbItem = (string)lbItems.SelectedItem;
            string[] parts = lbItem.Split(',');
            string armName = parts[0].Trim();

            ItemSize armSize = (ItemSize)Enum.Parse(typeof(ItemSize), parts[3].Trim());

            int armPrice;
            int armWeight;
            int armDefenseValue;
            int armDefenseBonus;

            Int32.TryParse(parts[1].Trim(), out armPrice);
            Int32.TryParse(parts[2].Trim(), out armWeight);
            Int32.TryParse(parts[4].Trim(), out armDefenseValue);
            Int32.TryParse(parts[5].Trim(), out armDefenseBonus);

            Armor newArmor = new Armor(
                    armName,
                    armPrice,
                    armWeight,
                    armSize,
                    armDefenseValue,
                    armDefenseBonus);

            FormArmor frmArmor = new FormArmor();
            frmArmor.Armor = newArmor;
            frmArmor.ShowDialog();

            if (frmArmor.OKPressed)
            {
                newArmor = frmArmor.Armor;
                if (newArmor.Name != armName)
                {
                    DialogResult result = MessageBox.Show(
                        "Proceeding will delete the old entry. Are you sure?",
                        "Warning",
                        MessageBoxButtons.YesNo);
                    if (result == DialogResult.No)
                        return;
                    armors.Remove(armName);
                    armors.Add(newArmor.Name, newArmor);
                }

                string output = newArmor.Name + ", " + newArmor.Price.ToString() + ", ";
                output += newArmor.Weight.ToString() + ", " + newArmor.Size.ToString() + ", ";
                output += newArmor.DefenseValue.ToString() + ", ";
                output += newArmor.DefenseBonus.ToString();
                armors[newArmor.Name] = newArmor;
                lbItems.Items[itemNumber] = output;
            }
            return;
        }

        if (shieldToolStripMenuItem.CheckState == CheckState.Checked)
        {
            if (lbItems.SelectedIndex == -1)
                return;

            int itemNumber = lbItems.SelectedIndex;
            string lbItem = (string)lbItems.SelectedItem;
            string[] parts = lbItem.Split(',');
            string sldName = parts[0].Trim();

            ItemSize sldSize = (ItemSize)Enum.Parse(typeof(ItemSize), parts[3].Trim());

            int sldPrice;
            int sldWeight;
            int sldDefenseValue;
            int sldDefenseBonus;

            Int32.TryParse(parts[1].Trim(), out sldPrice);
            Int32.TryParse(parts[2].Trim(), out sldWeight);
            Int32.TryParse(parts[4].Trim(), out sldDefenseValue);
            Int32.TryParse(parts[5].Trim(), out sldDefenseBonus);

            Shield newShield = new Shield(
                    sldName,
                    sldPrice,
                    sldWeight,
                    sldSize,
                    sldDefenseValue,
                    sldDefenseBonus);

            FormShield frmShield = new FormShield();
            frmShield.Shield = newShield;
            frmShield.ShowDialog();

            if (frmShield.OKPressed)
            {
                newShield = frmShield.Shield;
                if (newShield.Name != sldName)
                {
                    DialogResult result = MessageBox.Show(
                        "Proceeding will delete the old entry. Are you sure?",
                        "Warning",
                        MessageBoxButtons.YesNo);
                    if (result == DialogResult.No)
                        return;
                    shields.Remove(sldName);
                    shields.Add(newShield.Name, newShield);
                }

                string output = newShield.Name + ", " + newShield.Price.ToString() + ", ";
                output += newShield.Weight.ToString() + ", " + newShield.Size.ToString() + ", ";
                output += newShield.DefenseValue.ToString() + ", ";
                output += newShield.DefenseBonus.ToString();
                shields[newShield.Name] = newShield;
                lbItems.Items[itemNumber] = output;
            }
        }
    }

    void btnDelete_Click(object sender, EventArgs e)
    {
        if (lbItems.SelectedIndex == -1)
            return;

        string line = (string)lbItems.SelectedItem;
        string[] parts = line.Split(',');
        line = parts[0];

        DialogResult result = MessageBox.Show(
            "Are you sure you want to delete " + line + "?");

        if (result == DialogResult.No)
            return;

        if (weaponToolStripMenuItem.CheckState == CheckState.Checked)
        {
            weapons.Remove(line);
            FillWeapons();
            return;
        }
        if (armorToolStripMenuItem.CheckState == CheckState.Checked)
        {
            armors.Remove(line);
            FillArmor();
            return;
        }
        if (shieldToolStripMenuItem.CheckState == CheckState.Checked)
        {
            shields.Remove(line);
            FillShields();
            return;
        }
    }

    private void FillWeapons()
    {
        lbItems.Items.Clear();
        foreach (string name in weapons.Keys)
        {
            Weapon wpn = weapons[name];
            string output = wpn.Name + ", " + wpn.Price.ToString() + ", ";
            output += wpn.Weight.ToString() + ", " + wpn.Size.ToString() + ", ";
            output += wpn.NumberHands.ToString() + ", " + wpn.AttackValue.ToString() + ", ";
            output += wpn.AttackBonus.ToString();
            lbItems.Items.Add(output);
        }
    }

    private void FillArmor()
    {
        lbItems.Items.Clear();
        foreach (string name in armors.Keys)
        {
            Armor newArmor = armors[name];
            string output = newArmor.Name + ", " + newArmor.Price.ToString() + ", ";
            output += newArmor.Weight.ToString() + ", " + newArmor.Size.ToString() + ", ";
            output += newArmor.DefenseValue.ToString() + ", ";
            output += newArmor.DefenseBonus.ToString();
            lbItems.Items.Add(output);
        }
    }

    private void FillShields()
    {
        lbItems.Items.Clear();
        foreach (string name in shields.Keys)
        {
            Shield newShield = shields[name];
```

```
                string output = newShield.Name + ", " + newShield.Price.ToString() + ", ";
                output += newShield.Weight.ToString() + ", " + newShield.Size.ToString() + ", ";
                output += newShield.DefenseValue.ToString() + ", ";
                output += newShield.DefenseBonus.ToString();
                lbItems.Items.Add(output);
            }
        }

    }
}
```

I made the size of the code a little smaller in the tutorial. It allowed more on each line and reduced the size of the tutorial by two pages. A lot of the code is similar as items are quite similar as well as the functions of the events.

Since I'm working with XML there is of course a using statement for the XML name space. I'm not using it in this part of the tutorial but thought it would be best to keep it in. For storing items I thought the best approach would be to use a dictionary for each type of item. The item names are going to be unique for each type of item. To look up an item you can just use its name. So, for the key I used the name of the item and for the value I used the type of item. The field **weapons** is for weapons, **armors** for armor, and **shields** is for shields.

In the constructor for **FormMain** I wire some of the events that I'm interested in. At the moment I'm interested if the buttons are clicked, the items in the Item Type menu are clicked, and the exit item in the file menu. Remember, when you are wiring events like this you can press twice to have visual studio create method stubs for you. The code for the exit menu item is trivial, you just call the **Close** method for the form to close the form.

The code for the Item Type menu items is all similar. It works a little like radio buttons. For the weapon item, I check its **CheckState** to see if it is currently not checked. If it is not checked, it sets **CheckState** to be checked and sets the **CheckState** for the armor and shield items to unchecked. It finally calls a method, **FillWeapon**, that I wrote to fill the list box with the weapons. The other two methods are similar. They check to see if they are not currently unchecked, if they are unchecked the set their state to checked and uncheck the other items. Then call the appropriate method to fill the list box with items.

The **btnAdd** click event is where I handle adding items to the collection of items. Inside there are three if statements that check the **CheckState** of the menu items. The first checked to see if the weapon item is selected. If it is I create and display a **FormWeapon** using the **ShowDialog** method of the **Form** class. This displays the form so that the user has to close the form before continuing.

I then check to see if the OK button was pressed on the form. If it was I set an instance of the **Weapon** class to be the weapon on the form. I then use the **Contains** method of the **Dictionary** class to see if there is already a weapon by that name in the dictionary. This is because the keys in a dictionary have to be unique, you can't have two entries with the same name. That is part of the reason why I used a dictionary. If there is an entry I display a message box saying there is already a weapon with that name and exit the method. I then add the weapon to the dictionary.

The next part I did to list the weapons in the list box. What I did was convert the weapon instance into a string using commas to separate the parts. Doing this I can add that string to the list box, later I will end up having to decode the string and create an instance for editing. I then add the string to the list box. I will mention, this means that you can not use commas in the names of your items. If you do then converting the string back to an object will fail. If you want to use commas in your item names then you can select a character that you won't have in your item names.

The second and third if statements work like the first but are for armor and shields respectively. I create an instance of the form and display. Then check to see if the OK button was pressed. Get the instance of the item from the form. Check to see if an item of that name already exists, if it does display a message an exit. Add the new item to the dictionary, create a string to represent the item, and then add that string to the list box.

The event handler for **btnEdit** has the same structure as **btnAdd**. There are three if statements inside the method that check to see which item is selected in the Item Types menu. Inside those if statements I handle editing the items.

The first if statement checks to see if the weapon item is selected. It then checks to see if the **SelectedIndex** property of the list box is -1. If it is then there is no item selected and nothing to edit so I exit the method. I then save the selected item number. I then get a string that represents the selected item. **SelectedItem** returns **object** so the result has to be cast to **string**. I then split the string into its parts using the **Split** method of the string class using a comma to slit the string. If you used a different character to separate items replace the comma with it. This returns an array of **string** that holds the individual pieces.

In the next section I change the array of strings into variables of the type needed to create an instance of the weapon class. The name is easy as it is a string. I use the **Trim** method to remove any trailing or leading spaces, just to be safe. For the size of the item, I use the **Parse** method of the **Enum** class. This method takes a string and the type of the enumeration that you want to parse and returns its value, you have to cast it. The same is true for the number of hands required to use the item. For the integer values I first have four variable for each of the values. This is because I used the **TryParse** method to parse the integers. You need a variable to hold the parsed value passed as an out parameter to the **TryParse** method.

After all of the conversion, I create a new instance of the weapon class and of **FormWeapon**. Before displaying the form I set the **Weapon** property of the form so the form knows that it will be in edit mode. I then display the form. Like in the add code once the form is closed, I check to see if the OK button was pressed on the form. If it was I get the instance of the weapon using the **Weapon** property. The next part is where I check to see if the old name equals the new name. I decided that if the name was changed the old weapon would be deleted and the new one added. I display a message box asking if they want to remove the old weapon. If they don't I return, if they do I delete the old entry in the dictionary and add the new one. I then create a string to represent the item. I set the entry for that name to be the new weapon and set the list box item at the index to be the entry.

The code for shields and armors follow the same pattern. They are identical except for the fact that they work with armor and shield items instead of weapon items.

The next method I want to talk about is where I handle deleting items, **btnDelete_Click**. The first thing I do is check to see if an item in the list box is selected. If there isn't an item selected then there is nothing to try and delete so I return form the method. I then get the text for the selected item. I split it on the comma again. If you used a different character for separating items, replace the comma with that character. I only need the first part of the array of strings that was created so I set the variable **line** to be that item.

I then display a message box asking if they are sure, always nice to do that. If they aren't I exit the method. There are then three if statements that preform the same actions. They check to see what item type is in the list box. The first one is for weapons, the second for armor, and the third for shields. Inside the if statements I remove the item form the appropriate dictionary. I then call the appropriate fill method, **FillWeapons** for weapons, **FillArmor** for armor, and **FillShield** for shields. I then exit the method.

That just leaves the fill methods for this tutorial. They all have the same structure. I will explain how the **FillWeapons** method works. You should be able to understand the others from that. The first step is to remove all of the items in the list box. You do that by calling the **Clear** method of the **Items** collection. Then in a foreach loop, I loop through all of the keys in the dictionary. Remember, the keys are the names of the items. Inside of the foreach loop, I create a string to add to the **Items** collection of the list box the same as before. I then add the string to the **Items** collection of the list box.

Well, that is it for part **a** of the tutorial. I hope to have part **b** available very soon. So, I encourage you to keep either visiting my site http://xnagpa.net or my blog, XNA Game Programming Adventures Blog for the latest news on my tutorials.