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

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. For this tutorial, you will need the last version of the [Item Editor]. You will also need the last version of the game, [New2DRPG49.zip].

In this tutorial I will cover saving items from the editor, reading them back into the editor, and reading them into the game. In a future tutorial I will work on making a custom content pipeline project for the items.

To get started, open the last version of the Item Editor project. What I will do first is wire the event handler code to the constructor. Don't forget when you type the += that you can press tab twice to generate the method stubs. This is the code for the new constructor. Also, make sure that you have a using statement for the System.Xml namespace.

```
public FormMain()
{
    InitializeComponent();

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

    openToolStripMenuItem.Click += new EventHandler(openToolStripMenuItem_Click);
    saveToolStripMenuItem.Click += new EventHandler(saveToolStripMenuItem_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;
}
```

I will start with the code for saving items as you can't read items until they are being saved. I had originally planned on having a single file for items call **items.its**. It was easy enough to create a **SaveFileDialog** to all you to choose the file name and folder that you wanted to save the files to so I added that in. I will first give you the code for the event handler.

```
void saveToolStripMenuItem_Click(object sender, EventArgs e)
{
    SaveFileDialog saveDialog = new SaveFileDialog();
    saveDialog.AddExtension = true;
    saveDialog.CheckPathExists = true;
    saveDialog.DefaultExt = "its";
    saveDialog.Filter = "(Items *.its)|*.its";
    saveDialog.ValidateNames = true;

    DialogResult result = saveDialog.ShowDialog();
```

```
    if (result == DialogResult.Cancel)
        return;

    XmlDocument xmlDoc = new XmlDocument();

    XmlElement root = xmlDoc.CreateElement("Items");
    xmlDoc.AppendChild(root);

    WriteWeapons(xmlDoc, root);
    WriteArmor(xmlDoc, root);
    WriteShields(xmlDoc, root);

    xmlDoc.Save(saveDialog.FileName);
}
```

I separated much of the saving of items into different methods. What I first do is create a **SaveFileDialog** instance so that I can display it called **saveDialog**. I then set a few of the properties for **saveDialog**. The first one will append the default extension to the file name if one is omitted. The next one checks to make sure that the path to the file is valid. As I mentioned, I decided to use **its** for the extension of item files, so that is the default extension. The next part is the filter for the file names in the save dialog. The filter has two parts, separated by the | character. The first part is the display name and the second is the extension filter. The last property validates file names so that they are valid windows file names. I display the **saveFile** dialog and capture the result. If the result was cancel I exit the method.

The next few steps are creating an **XmlDocument** object and a root **XmlElement**. I append the **root** node to the XML document. I then call methods that will write the various items. **WriteWeapons** will write the weapons, **WriteArmor** will write the armor, and **WriteShields** will write the shields. I then save the XML document.

Since the items are all basically the same, the code for saving them is basically the same as well. The main differences between weapons and armor and shields is that weapons require a certain number of hands to use and weapons have attack values and bonuses rather than defense values and bonuses. This is the code for those three methods. I will explain it after you have read it.

```
private void WriteWeapons(XmlDocument xmlDoc, XmlElement root)
{
    XmlElement weaponsNode = xmlDoc.CreateElement("Weapons");
    root.AppendChild(weaponsNode);

    foreach (string wpn in weapons.Keys)
    {
        XmlElement wpnNode = xmlDoc.CreateElement("Weapon");

        XmlElement element = xmlDoc.CreateElement("Name");
        XmlAttribute attrib = xmlDoc.CreateAttribute("Value");
        attrib.Value = weapons[wpn].Name;
        element.Attributes.Append(attrib);
        wpnNode.AppendChild(element);

        element = xmlDoc.CreateElement("Price");
        attrib = xmlDoc.CreateAttribute("Value");
        attrib.Value = weapons[wpn].Price.ToString();
        element.Attributes.Append(attrib);
        wpnNode.AppendChild(element);
```

```
            element = xmlDoc.CreateElement("Weight");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = weapons[wpn].Weight.ToString();
            element.Attributes.Append(attrib);
            wpnNode.AppendChild(element);

            element = xmlDoc.CreateElement("Size");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = weapons[wpn].Size.ToString();
            element.Attributes.Append(attrib);
            wpnNode.AppendChild(element);

            element = xmlDoc.CreateElement("Hands");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = weapons[wpn].NumberHands.ToString();
            element.Attributes.Append(attrib);
            wpnNode.AppendChild(element);

            element = xmlDoc.CreateElement("Attack");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = weapons[wpn].AttackValue.ToString();
            element.Attributes.Append(attrib);
            attrib = xmlDoc.CreateAttribute("Bonus");
            attrib.Value = weapons[wpn].AttackBonus.ToString();
            element.Attributes.Append(attrib);
            wpnNode.AppendChild(element);

            weaponsNode.AppendChild(wpnNode);
        }
    }

    private void WriteArmor(XmlDocument xmlDoc, XmlElement root)
    {
        XmlElement armorNode = xmlDoc.CreateElement("Armors");
        root.AppendChild(armorNode);

        foreach (string arm in armors.Keys)
        {
            XmlElement armNode = xmlDoc.CreateElement("Armor");

            XmlElement element = xmlDoc.CreateElement("Name");
            XmlAttribute attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = armors[arm].Name;
            element.Attributes.Append(attrib);
            armNode.AppendChild(element);

            element = xmlDoc.CreateElement("Price");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = armors[arm].Price.ToString();
            element.Attributes.Append(attrib);
            armNode.AppendChild(element);

            element = xmlDoc.CreateElement("Weight");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = armors[arm].Weight.ToString();
            element.Attributes.Append(attrib);
            armNode.AppendChild(element);
```

```
            element = xmlDoc.CreateElement("Size");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = armors[arm].Size.ToString();
            element.Attributes.Append(attrib);
            armNode.AppendChild(element);

            element = xmlDoc.CreateElement("Defense");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = armors[arm].DefenseValue.ToString();
            element.Attributes.Append(attrib);
            attrib = xmlDoc.CreateAttribute("Bonus");
            attrib.Value = armors[arm].DefenseBonus.ToString();
            element.Attributes.Append(attrib);
            armNode.AppendChild(element);

            armorNode.AppendChild(armNode);
        }
    }

    private void WriteShields(XmlDocument xmlDoc, XmlElement root)
    {
        XmlElement shieldNode = xmlDoc.CreateElement("Shields");
        root.AppendChild(shieldNode);

        foreach (string sld in shields.Keys)
        {
            XmlElement sldNode = xmlDoc.CreateElement("Shield");

            XmlElement element = xmlDoc.CreateElement("Name");
            XmlAttribute attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = shields[sld].Name;
            element.Attributes.Append(attrib);
            sldNode.AppendChild(element);

            element = xmlDoc.CreateElement("Price");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = shields[sld].Price.ToString();
            element.Attributes.Append(attrib);
            sldNode.AppendChild(element);

            element = xmlDoc.CreateElement("Weight");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = shields[sld].Weight.ToString();
            element.Attributes.Append(attrib);
            sldNode.AppendChild(element);

            element = xmlDoc.CreateElement("Size");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = shields[sld].Size.ToString();
            element.Attributes.Append(attrib);
            sldNode.AppendChild(element);

            element = xmlDoc.CreateElement("Defense");
            attrib = xmlDoc.CreateAttribute("Value");
            attrib.Value = shields[sld].DefenseValue.ToString();
            element.Attributes.Append(attrib);
            attrib = xmlDoc.CreateAttribute("Bonus");
            attrib.Value = shields[sld].DefenseBonus.ToString();
            element.Attributes.Append(attrib);
```

```
            sldNode.AppendChild(element);

            shieldNode.AppendChild(sldNode);
        }
    }
```

As you can see, the methods look nearly identical, with shields and armor being the closest. The code follows the same basic pattern, create a node for all of the items of one type. Append that node to the root node. In a foreach loop go through all of the items of that type. In the foreach loop, create an element for the type of item. For that element create the individual elements and append them to the element for that type. I will explain the **WriteWeapons** method thoroughly and the rest I'm sure you can figure out from that method.

The first thing to do in **WriteWeapons** is to create an **XmlElement** called **Weapons** and append it to **root**. Next is the foreach loop that loops through all of the keys in the dictionary. Inside the foreach loop I create an element called **Weapon** that will hold the individual weapons.

I then create the elements for that weapon. I went with having an elements having the name of the field I'm storing with its value in a Value attribute. The one exception is the attack value and attack bonus fields. I combined them into a single element. After creating an element individual element, it is appended to the **Weapon** element.

The order in which the elements are written is important. They will be read in in the same order as they were written out. The Name element is an easy element. I just create an **XmlElement** with the name **Name**. I then create an **XmlAttribute** named **Value** and set it to the **Name** property of the current weapon, which is found using the **weapons** dictionary using the loop string, **wpn**.

The next element is the **Price** element. Like for **Name**, I create an **XmlElement** named **Price**. I then create an **XmlAttribute** named **Value** and set is value to the price of the weapon as a string. You can only write strings into an XML document. The **Weight** element is done the same way.

The **Size** and **Hands** elements are done the same way. Except, they are written using the string that represents the value of the enum. **Size** can be **Tiny, Small, Medium,** or **Large** and **Hands** can be **One** or **Two**. I could have converted them to integers and then cast them back when they are read in but seemed a little over kill here.

The **Attack** element is a little different from the others. The reason why is I have two attributes here, **Value** and **Bonus**. They represent the **attackValue** and **attackBonus** properties for the weapon. After creating all of the elements I append the **Weapon** element to the **Weapons** element.

The **WriteArmor** and **WriteShield** methods work the same way. The only difference is that instead of having the **Attack** element, they have a **Defense** element. Armors are written into the **Armors** element with each individual armor being written into an **Armor** element. Shields are written into the **Shields** element, with each individual shield being written into a **Shield** element.

Now I will work on reading items back in, which is the last procedure in reverse. The way I implemented it was a little different though. I will start with the event handler and then move to the individual methods that read the items. This is the code for the **openToolStripMenuItem_Click** method.

```
void openToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog openDialog = new OpenFileDialog();
    openDialog.AddExtension = true;
    openDialog.CheckFileExists = true;
    openDialog.CheckPathExists = true;
    openDialog.Filter = "(Items *.its)|*.its";
    openDialog.Multiselect = false;
    openDialog.ValidateNames = true;

    DialogResult result = openDialog.ShowDialog();
    if (result == DialogResult.Cancel)
        return;

    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.Load(openDialog.FileName);

    XmlNode root = xmlDoc.FirstChild;

    weapons.Clear();
    armors.Clear();
    shields.Clear();
    lbItems.Items.Clear();

    foreach (XmlNode node in root.ChildNodes)
    {
        if (node.Name == "Weapons")
        {
            foreach (XmlNode wpn in node.ChildNodes)
                ReadWeapon(wpn);
        }
        if (node.Name == "Armors")
        {
            foreach (XmlNode arm in node.ChildNodes)
                ReadArmor(arm);
        }
        if (node.Name == "Shields")
        {
            foreach (XmlNode sld in node.ChildNodes)
                ReadShield(sld);
        }
    }
    if (weaponToolStripMenuItem.CheckState == CheckState.Checked)
        FillWeapons();
    if (armorToolStripMenuItem.CheckState == CheckState.Checked)
        FillArmor();
    if (shieldToolStripMenuItem.CheckState == CheckState.Checked)
        FillShields();
}
```

The first thing to do is to create an **OpenFileDialog** object to select the file to open. I then set a few properties for **openDialog**. They are the same except for **CheckFileExists** and **MultiSelect**. **CheckFileExists** checks to make sure there is a file to open. Setting **MultiSelect** to false means that only one file can be selected. I capture the result after **openDialog** is displayed. If the Cancel button was selected I exit the method. I then open the **XmlDocument**.

I then get the root node. I probably should have checked to make sure it was the **Items** node to make sure I was reading the right file. I then remove all of the items from the dictionairies, just to be sure that there will be no duplicate keys being added. I also remove the entries form the list box's **Items** collection.

There is then a foreach loop that loops through all of the children of **root**. Inside that foreach loop are three if statements. They check to see what the name of the current node is. If it is **Weapons**, there is a foreach loop that will loop through all of the weapons in that node reading them in using the **ReadWeapon** method. If it is **Armors**, there is a foreach loop that will loop through all of the armors in that node reading them in using the **ReadArmor** method. Finally, there is an if statement that checks to see if the current node is **Shields**. Inside of the if there is another foreach loop that will loop through all of the shields and reads them in using the **ReadShield** method. After reading the items in there are three if statements that check to see which type of item is being worked on. If it is weapons, I call the **FillWeapons** method, for armor, the **FillArmor** method and for shields, **FillShields**.

That just leaves the three reading methods, **ReadWeapon**, **ReadArmor**, and **ReadShield**. All three methods are similar in structure, just like writing them. They try and parse each type of item, throwing exceptions if they aren't in the proper order. I will give you the code and give an in depth explanation of **ReadWeapon**. I believe the other two are similar enough that you should be able to figure them out.

```
private void ReadWeapon(XmlNode wpn)
{
    string name;
    int price;
    int weight;
    ItemSize size;
    Hands hands;
    int attackValue;
    int attackBonus;

    XmlNode node = wpn.FirstChild;
    if (node.Name != "Name")
        throw new Exception("Illegal file format!");
    name = node.Attributes[0].Value;

    node = node.NextSibling;
    if (node.Name != "Price")
        throw new Exception("Illegal file format!");
    price = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Weight")
        throw new Exception("Illegal file format!");
    weight = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Size")
        throw new Exception("Illegal file format!");
    size = (ItemSize)Enum.Parse(typeof(ItemSize), node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Hands")
        throw new Exception("Illegal file format!");
```

```
    hands = (Hands)Enum.Parse(typeof(Hands), node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Attack")
        throw new Exception("Illegal file format!");
    attackValue = int.Parse(node.Attributes[0].Value);
    attackBonus = int.Parse(node.Attributes[1].Value);

    Weapon weapon = new Weapon(
            name,
            price,
            weight,
            size,
            hands,
            attackValue,
            attackBonus);
    weapons.Add(name, weapon);
}

private void ReadArmor(XmlNode arm)
{
    string name;
    int price;
    int weight;
    ItemSize size;
    int defenseValue;
    int defenseBonus;

    XmlNode node = arm.FirstChild;
    if (node.Name != "Name")
        throw new Exception("Illegal file format!");
    name = node.Attributes[0].Value;

    node = node.NextSibling;
    if (node.Name != "Price")
        throw new Exception("Illegal file format!");
    price = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Weight")
        throw new Exception("Illegal file format!");
    weight = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Size")
        throw new Exception("Illegal file format!");
    size = (ItemSize)Enum.Parse(typeof(ItemSize), node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Defense")
        throw new Exception("Illegal file format!");
    defenseValue = int.Parse(node.Attributes[0].Value);
    defenseBonus = int.Parse(node.Attributes[1].Value);

    Armor armr = new Armor(
            name,
            price,
            weight,
            size,
```

```
            defenseValue,
            defenseBonus);
    armors.Add(name, armr);
}

private void ReadShield(XmlNode sld)
{
    string name;
    int price;
    int weight;
    ItemSize size;
    int defenseValue;
    int defenseBonus;

    XmlNode node = sld.FirstChild;
    if (node.Name != "Name")
        throw new Exception("Illegal file format!");
    name = node.Attributes[0].Value;

    node = node.NextSibling;
    if (node.Name != "Price")
        throw new Exception("Illegal file format!");
    price = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Weight")
        throw new Exception("Illegal file format!");
    weight = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Size")
        throw new Exception("Illegal file format!");
    size = (ItemSize)Enum.Parse(typeof(ItemSize), node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Defense")
        throw new Exception("Illegal file format!");
    defenseValue = int.Parse(node.Attributes[0].Value);
    defenseBonus = int.Parse(node.Attributes[1].Value);

    Shield shld = new Shield(
            name,
            price,
            weight,
            size,
            defenseValue,
            defenseBonus);
    shields.Add(name, shld);
}
```

There are local variables for each of the parameters required to create a weapon object: **name**, **price**, **weight**, **size**, **hands**, **attackValue**, and **attackBonus**. To get the first node inside the element, I use the **FirstChild** property. If that node's name is not **Name**, I throw an exception. If it is, I set the **name** variable to value of **Attributes[0]**, the first attribute. I could have also said, **Attributes["Value"]** to get the attribute. Since **name** is a string, I didn't have to convert it to the proper type. I then set **node** to be the **NextSibling** of itself, moving on to the next node.

I then check to see if the current node is the **Price** node. If it is not, I throw and exception. If it is I parse the value of **Attributes[0]** to an integer and move to the next node using the **NextSibling** property.

The next node should be the **Weight** node. If it is not the **Weight** node, I throw an exception. If it is the **Weight** node, I parse the value of **Attributes[0]** to an integer. The move on to the next next node using the **NextSibling** property.

The next node would be the **Size** node. Again, if it isn't I throw an exception. If it is I use the **Parse** method of the **Enum** class to parse it into the proper format for the **size** variable and move to the next node. The same is true for the **Hands** node. If the node is not the **Hands** node, throw and exception. Otherwise convert the node and move onto the next one.

The last node is different than the others but only because it has two attributes, not one. I check to make sure it is the **Attack** node. If it isn't throw an exception. If it is, set **attackValue** to be value of the first attribute and **attackBonus** to be the value of the second attribute.

After parsing all of the elements, I create an instance of the **Weapon** class. I add that instance to the dictionary of weapons using the **name** variable for the key and the instance of the **Weapon** class as the value for the entry. The other read methods are the same. The parse the individual items and add them to the appropriate dictionary. You can now close that project. You can get the entire project from the Item Editor solution.

With that done, it is time to move onto the actual game. Go a head and load the last version of your game. The first thing to do is to add an item file to the game that can be read in. You can download the one I used from Items.zip. If you don't use that one, you will have to change the names of the items that go into the chests.

Once you have the file, you need to add it to the **Content** folder of the game. Right click the **Items** folder inside the **Content** folder, select **Add** and the **Existing Item**. Navigate to your items file, mine was called **items.its**, and add it to the **Items** folder. You now need to make sure that you change the **Build Action** property for **items.its** to **None** and the **Copy To Output Directory** to **Copy Always**.

There is one thing that you should do before going much further. That is go into the **Shield.cs** file in the **ItemsClasses** folder. You want to change the class definition to the following, this will prevent conflicts later on down the road for the project.

```
public class Shield : BaseItem
```

Now the items need to be read into the game. To do that you will need three new fields in the **Game1** class. They are a dictionary for each type of item. Near the **chests** field, add the following three fields.

```
Dictionary<string, Weapon> weapons = new Dictionary<string, Weapon>();
Dictionary<string, Armor> armors = new Dictionary<string, Armor>();
Dictionary<string, Shield> shields = new Dictionary<string, Shield>();
```

The next thing to do is to actually read in the items. I will do that form the **LoadContent** method but calling a new method, **ReadItems**. The **ReadItems** method will be a simpler version of the method from the **Item Editor**. It will call the methods like in the **Item Editor** project as well. You will also need a copy of the **ReadArmor**, **ReadShield**, and **ReadWeapon** methods. The easiest way to do this would be to open the **ItemEditor** project in another window. You can then select those methods and copy and paste them below the **ReadItems** method.Change the **LoadContent** method to the following and add the **ReadItems** method. I've also included the code for the other methods to read in armor, shields, and weapons. If you used a different file name that I did for the items file, change it in the call to **ReadItems**.

```
protected override void LoadContent()
{
    spriteBatch = new SpriteBatch(GraphicsDevice);
    tileSpriteBatch = new SpriteBatch(GraphicsDevice);

    Services.AddService(typeof(SpriteBatch), spriteBatch);
    Services.AddService(typeof(ContentManager), Content);

    dialog = new DialogComponent(this);
    Components.Add(dialog);

    normalFont = Content.Load<SpriteFont>("normal");

    LoadGameScreens();

    CreateAnimations();

    spriteTextures = new Texture2D[assetNames.Length];
    for (int i = 0; i < assetNames.Length; i++)
        spriteTextures[i] = Content.Load<Texture2D>(assetNames[i]);

    script = ReadScript(@"Content\script1.script");

    LoadPlayerSprites();
    CreatePlayerAnimations();

    ReadItems(@"Content\Items\items.its");

    CreateNPCS();
    CreateMonsters();
    CreateChests();
}

private void ReadItems(string filename)
{
    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.Load(filename);

    XmlNode root = xmlDoc.FirstChild;

    weapons.Clear();
    armors.Clear();
    shields.Clear();

    foreach (XmlNode node in root.ChildNodes)
    {
        if (node.Name == "Weapons")
            foreach (XmlNode wpn in node.ChildNodes)
                ReadWeapon(wpn);
        if (node.Name == "Armors")
```

```
            foreach (XmlNode arm in node.ChildNodes)
                ReadArmor(arm);
        if (node.Name == "Shields")
            foreach (XmlNode sld in node.ChildNodes)
                ReadShield(sld);
    }
}

private void ReadWeapon(XmlNode wpn)
{
    string name;
    int price;
    int weight;
    ItemSize size;
    Hands hands;
    int attackValue;
    int attackBonus;

    XmlNode node = wpn.FirstChild;
    if (node.Name != "Name")
        throw new Exception("Illegal file format!");
    name = node.Attributes[0].Value;

    node = node.NextSibling;
    if (node.Name != "Price")
        throw new Exception("Illegal file format!");
    price = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Weight")
        throw new Exception("Illegal file format!");
    weight = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Size")
        throw new Exception("Illegal file format!");
    size = (ItemSize)Enum.Parse(typeof(ItemSize), node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Hands")
        throw new Exception("Illegal file format!");
    hands = (Hands)Enum.Parse(typeof(Hands), node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Attack")
        throw new Exception("Illegal file format!");
    attackValue = int.Parse(node.Attributes[0].Value);
    attackBonus = int.Parse(node.Attributes[1].Value);

    Weapon weapon = new Weapon(
            name,
            price,
            weight,
            size,
            hands,
            attackValue,
            attackBonus);
    weapons.Add(name, weapon);
}
```

```csharp
private void ReadArmor(XmlNode arm)
{
    string name;
    int price;
    int weight;
    ItemSize size;
    int defenseValue;
    int defenseBonus;

    XmlNode node = arm.FirstChild;
    if (node.Name != "Name")
        throw new Exception("Illegal file format!");
    name = node.Attributes[0].Value;

    node = node.NextSibling;
    if (node.Name != "Price")
        throw new Exception("Illegal file format!");
    price = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Weight")
        throw new Exception("Illegal file format!");
    weight = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Size")
        throw new Exception("Illegal file format!");
    size = (ItemSize)Enum.Parse(typeof(ItemSize), node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Defense")
        throw new Exception("Illegal file format!");
    defenseValue = int.Parse(node.Attributes[0].Value);
    defenseBonus = int.Parse(node.Attributes[1].Value);

    Armor armr = new Armor(
            name,
            price,
            weight,
            size,
            defenseValue,
            defenseBonus);
    armors.Add(name, armr);
}

private void ReadShield(XmlNode sld)
{
    string name;
    int price;
    int weight;
    ItemSize size;
    int defenseValue;
    int defenseBonus;

    XmlNode node = sld.FirstChild;
    if (node.Name != "Name")
        throw new Exception("Illegal file format!");
    name = node.Attributes[0].Value;
```

```
    node = node.NextSibling;
    if (node.Name != "Price")
        throw new Exception("Illegal file format!");
    price = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Weight")
        throw new Exception("Illegal file format!");
    weight = int.Parse(node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Size")
        throw new Exception("Illegal file format!");
    size = (ItemSize)Enum.Parse(typeof(ItemSize), node.Attributes[0].Value);

    node = node.NextSibling;
    if (node.Name != "Defense")
        throw new Exception("Illegal file format!");
    defenseValue = int.Parse(node.Attributes[0].Value);
    defenseBonus = int.Parse(node.Attributes[1].Value);

    Shield shld = new Shield(
            name,
            price,
            weight,
            size,
            defenseValue,
            defenseBonus);
    shields.Add(name, shld);
}
```

The last thing to do is change the **CreateChests** method to use the items that were read in. I will give you the code for that method and then explain it. This is the code for the updated **CreateChests** method.

```
private void CreateChests()
{
    Chest tempChest;

    for (int i = 0; i < 2; i++)
    {
        tempChest = new Chest(
            this,
            Content.Load<Texture2D>(@"Items\chest"),
            new Vector2(random.Next(3, 3 + 5), random.Next(3, 3 + 5)),
            random.Next(100),
            random.Next(100, 200),
            null);
        chests.Add(tempChest);
    }

    List<BaseItem> items = new List<BaseItem>();

    items.Add(weapons["dagger"]);
    items.Add(armors["leather armor"]);

    tempChest = new Chest(
```

```
        this,
        Content.Load<Texture2D>(@"Items\chest"),
        new Vector2(random.Next(5, 11), random.Next(5, 11)),
        random.Next(100),
        random.Next(100, 200),
        items);

    chests.Add(tempChest);

    List<BaseItem> items2 = new List<BaseItem>();
    items2.Add(shields["buckler"]);

    tempChest = new Chest(
        this,
        Content.Load<Texture2D>(@"Items\chest"),
        new Vector2(random.Next(5, 11), random.Next(5, 11)),
        items2);

    chests.Add(tempChest);
}
```

What I did is instead of creating items from code, is use items from the dictionary I was interested in. In the first chest I put in a dagger and leather armor. For the dagger I used the **weapons** dictionary using the key, **dagger**. For the leather armor, I used the **armors** dictionary and the key, **leather armor**. In the second chest, I added in a shield called **buckler**.

Well, that is it for part **b** of the tutorial. I am already working on the next tutorial in there series. In this tutorial I will cover adding the items into the player's inventory. 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.