

# .Net Expansions

---

## XML IO Guide

Copyright © 2009 by Rainer Hilmer

**Microsoft**  
**CERTIFIED**  
*Technology  
Specialist*

**.Net Framework 2.0**  
*Web Applications*

# XML IO

## Guide

---

# XML IO

## Guide

---

### Table of contents

- What XML IO is and why
- Quick Reference
- Business Objects
- Under the hood of XmlIo
- Other Samples

# XML IO

## Guide

---

### What and why

There are multiple ways to persist configuration data. Ini files have become pretty outdated. So this is the only time I mention them.

Another way to persist data are SQL databases and XML files. The latter can be a pain in the neck when you don't have much of a routine in handling XML tasks. Even the ConfigurationManager of the .net Framework isn't very comfortable to handle.

XML IO comes to the rescue. It makes persisting configuration data a piece of cake! Here is how it goes. Oh and by the way. XmlIo is not just for handling configurations but for any data that should be persisted in XML files. One of the demos shows how to handle a whole database in XML format!

# XML IO

## Guide

---

### Quick Reference

No background information, just a fast track sample that shows how to use this. The first thing you see here is a configuration Business Object, which can hold various values needed in a typical GUI configuration scenario.

```
// This type of Configuration Business Object is for use without XSD-validation only!
```

```
using System.Drawing;
using System.Runtime.Serialization;

namespace QuickReferenceSample
{
    [KnownType(typeof (Color))]
    [KnownType(typeof (Point))]
    [KnownType(typeof (Size))]
    [KnownType(typeof (Font))]
    [KnownType(typeof (FontStyle))]
    [KnownType(typeof (GraphicsUnit))]
    public class Configuration
    {
        public Size WindowSize { get; set; }
        public Point WindowLocation { get; set; }
        public Color BackgroundColor { get; set; }
        public Font UserFont { get; set; }
        public Color FontColor { get; set; }
    }
}
```

The points of interest here are the reference to System.Runtime.Serialization and the KnownType-Attribute (which the reference is needed for). XmlIo makes use of DataContracts (see the chapter „Under the hood of XmlIo“). In order to make XmlIo handle complex types as Color, Font and so on, they must be declared by this Attribute.

**Important note:** The sample shown here and on the next page is strictly for use without XSD validation only. A sample with validation follows after that.

# XML IO

## Guide

---

### Quick Reference (XML handling without validation)

This sample shows XML handling without validation, using the Business Object from the page before.

```
using System;
using System.Drawing;
using System.Windows.Forms;
using DotNetExpansions.IO;

namespace WithoutValidation
{
    internal class Program
    {
        private static void Main()
        {
            // Variation for use without validation.
            var config = new Configuration
            {
                BackgroundColor = Color.FromArgb(255, 255, 255),
                FontColor = Color.FromArgb(0, 0, 0),
                UserFont = new Font("Calibri", 12),
                WindowLocation = new Point(300, 200),
                WindowSize = new Size(800, 600)
            };

            var configManager = new XmlIo(Application.StartupPath, "config.xml");
            // Saving your configuration to the XML file.
            configManager.Save(config);

            // Loading your configuration from the xml file.
            Configuration newConfig = configManager.Load<Configuration>();

            // Verhindert das selbsttätige Schließen des Konsolenfensters.
            Console.WriteLine("\nPress any key to terminate the program.");
            Console.ReadKey();
        }
    }
}
```

# XML IO

## Guide

---

### Quick Reference (XML handling without validation)

This is the output, produced by this sample.

```
<Configuration xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/WithoutValidation">
  <BackgroundColor xmlns:d2p1="http://schemas.datacontract.org/2004/07/
System.Drawing">
    <d2p1:knownColor>0</d2p1:knownColor>
    <d2p1:name i:nil="true" />
    <d2p1:state>2</d2p1:state>
    <d2p1:value>4294967295</d2p1:value>
  </BackgroundColor>
  <FontColor xmlns:d2p1="http://schemas.datacontract.org/2004/07/
System.Drawing">
    <d2p1:knownColor>0</d2p1:knownColor>
    <d2p1:name i:nil="true" />
    <d2p1:state>2</d2p1:state>
    <d2p1:value>4278190080</d2p1:value>
  </FontColor>
  <UserFont xmlns:d2p1="http://schemas.datacontract.org/2004/07/
System.Drawing">
    <Name xmlns:d3p1="http://www.w3.org/2001/XMLSchema" i:type="d3p1:string"
xmlns="">Calibri</Name>
    <Size xmlns:d3p1="http://www.w3.org/2001/XMLSchema" i:type="d3p1:float"
xmlns="">12</Size>
    <Style i:type="d2p1:FontStyle" xmlns="">Regular</Style>
    <Unit i:type="d2p1:GraphicsUnit" xmlns="">Point</Unit>
  </UserFont>
  <WindowLocation xmlns:d2p1="http://schemas.datacontract.org/2004/07/
System.Drawing">
    <d2p1:x>300</d2p1:x>
    <d2p1:y>200</d2p1:y>
  </WindowLocation>
  <WindowSize xmlns:d2p1="http://schemas.datacontract.org/2004/07/
System.Drawing">
    <d2p1:height>600</d2p1:height>
    <d2p1:width>800</d2p1:width>
  </WindowSize>
</Configuration>
```

# XML IO

## Guide

---

### Quick Reference (XML handling with validation)

Here is a sample which uses XSD validation.

The first important thing you notice is the different structure of the Business Object. This is because it makes creating the Schema file way more simple!

```
// This type of Configuration Business Object can be used with XSD-validation.
```

```
namespace WithValidation
{
    public class Configuration4Validation
    {
        public WindowParams WindowSettings { get; set; }
        public BackGroundColor BackGroundColorSettings { get; set; }
        public FontParams UserFont { get; set; }
        public FontColor FontColorSettings { get; set; }

        #region Nested type: BackGroundColor

        public class BackGroundColor
        {
            public byte R { get; set; }
            public byte G { get; set; }
            public byte B { get; set; }
        }

        #endregion

        #region Nested type: FontColor

        public class FontColor
        {
            public byte R { get; set; }
            public byte G { get; set; }
            public byte B { get; set; }
        }

        #endregion

        #region Nested type: FontParams

        public class FontParams
        {
            public string Name { get; set; }
            public byte Size { get; set; }
        }
    }
}
```



# XML IO

## Guide

---

### Quick Reference (XML handling with validation)

```
        public string Style { get; set; }
        public string Unit { get; set; }
    }

    #endregion
    #region Nested type: WindowParams

    public class WindowParams
    {
        public uint Height { get; set; }
        public uint Width { get; set; }
        public uint X { get; set; }
        public uint Y { get; set; }
    }

    #endregion
}
}
```

The following pages show the sample code for using this kind of Business Object with validation.

# XML IO

## Guide

---

### Quick Reference (XML handling with validation)

```
using System;
using System.Windows.Forms;
using System.Xml.Schema;
using DotNetExpansions.IO;

namespace WithValidation
{
    internal class Program
    {
        private static void Main()
        {
            // Variation for use with Validation.
            var config = new Configuration4Validation
            {
                BackGroundColorSettings =
                    new Configuration4Validation.BackgroundColor
                    {
                        R = 255,
                        G = 255,
                        B = 255
                    },
                FontColorSettings =
                    new Configuration4Validation.FontColor
                    {
                        R = 0,
                        G = 0,
                        B = 0
                    },
                UserFont =
                    new Configuration4Validation.FontParams
                    {
                        Name = "Calibri",
                        Size = 12
                    },
                WindowSettings =
                    new Configuration4Validation.WindowParams
                    {
                        Height = 600,
                        Width = 800,
                        X = 300,
                        Y = 200
                    }
            };
        }
    }
}
```

# XML IO

## Guide

---

### Quick Reference (XML handling with validation)

```
var configManager = new XmlIo(Application.StartupPath, "config.xml");
// Saving your configuration to the XML file.
configManager.Save(config);

// Loading a configuration with validation
configManager.ValidationEvent += OnValidationEvent;
Configuration4Validation newConfig;
try
{
    if(configManager.FileIsValidWith("configSchema.xsd"))
        newConfig = configManager.Load<Configuration4Validation>();
    else
        DoSomethingAppropriate();
}
catch(XmlSchemaValidationException problem)
{
    Console.WriteLine(problem.ToString());
}

// Verhindert das selbsttätige Schließen des Konsolenfensters.
Console.WriteLine("\nPress any key to terminate the program.");
Console.ReadKey();
}

// Only needed when validation is involved.
private static void OnValidationEvent(
    object sender, System.Xml.Schema.ValidationEventArgs e)
{
    // Just a demo. Do something appropriate with the event.
    Console.WriteLine(e.Exception);
    Console.WriteLine(e.Message);
    Console.WriteLine(e.Severity);
}

// Only needed when validation is involved.
private static void DoSomethingAppropriate()
{
    // Do something appropriate.
}
}
```

# XML IO

## Guide

---

### Quick Reference (XML handling with validation)

This is the resulting XML which the sample generates.

```
<Configuration4Validation xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/WithValidation">
  <BackgroundColorSettings>
    <B>255</B>
    <G>255</G>
    <R>255</R>
  </BackgroundColorSettings>
  <FontColorSettings>
    <B>0</B>
    <G>0</G>
    <R>0</R>
  </FontColorSettings>
  <UserFont>
    <Name>Calibri</Name>
    <Size>12</Size>
    <Style i:nil="true" />
    <Unit i:nil="true" />
  </UserFont>
  <WindowSettings>
    <Height>600</Height>
    <Width>800</Width>
    <X>300</X>
    <Y>200</Y>
  </WindowSettings>
</Configuration4Validation>
```

# XML IO

## Guide

---

### Quick Reference (XML handling with validation)

This is the XSD file which was created using the Menu in Visual Studio 2008 (XML -> Create Schema).

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:i="http://www.w3.org/2001/XMLSchema-instance" attributeFormDe-
fault="unqualified" elementFormDefault="qualified" targetNamespace="http://
schemas.datacontract.org/2004/07/WithValidation" xmlns:xs="http://
www.w3.org/2001/XMLSchema">
  <xs:element name="Configuration4Validation">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="BackColorSettings">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="B" type="xs:unsignedByte" />
              <xs:element name="G" type="xs:unsignedByte" />
              <xs:element name="R" type="xs:unsignedByte" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="FontColorSettings">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="B" type="xs:unsignedByte" />
              <xs:element name="G" type="xs:unsignedByte" />
              <xs:element name="R" type="xs:unsignedByte" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="UserFont">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Name" type="xs:string" />
              <xs:element name="Size" type="xs:unsignedByte" />
              <xs:element name="Style" nillable="true" />
              <xs:element name="Unit" nillable="true" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="WindowSettings">
          <xs:complexType>
            <xs:sequence>
```

# XML IO

## Guide

---

### Quick Reference (XML handling with validation)

```
<xs:element name="Height" type="xs:unsignedShort" />
<xs:element name="Width" type="xs:unsignedShort" />
<xs:element name="X" type="xs:unsignedShort" />
<xs:element name="Y" type="xs:unsignedByte" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

# XML IO

## Guide

---

### Business Objects (BO)

First of it's best practice to use Business objects as transportation medium. These are pretty simple classes with properties which hold the various configuration values.

#### The purpose of Business Objects

BOs represent entities in an application domain They are modeled in a way so they can hold all information about a certain domain object, like for instance Customer or Product or an application configuration. The latter is the main focus of this guide.

#### The architecture of BOs

The structure of BOs should always reflect the domain they are used for. If you are interested in more detailed information about domain driven design, I recommend you get the e-book version of [Eric Evan's „Domain Driven Design Quickly“ over at InfoQ](#), or you just google for even more information.

The following code blocks show some Business Objects.

```
public class SimpleConfig
{
    public byte Dimensions { get; set; }
    public int HyperspaceEntrySection { get; set; }
    public string NanobotsEmitterName { get; set; }
    public string PicobotControllerIP { get; set; }
    public float WarpFactor { get; set; }
}
```

This one is a most simple form of BO. It has Properties for different Types. The names you see in this sample are absolutely useless in business applications, of course but that's just a demo anyway. Use whatever names are appropriate for your application.

# XML IO

## Guide

### Business Objects (BO)

Here's another one which is more realistic.

```
public class NestedConfig
{
    public string MachineName { get; set; }
    public UiConfig UiSettings { get; set; }
    public UserConfig UserSettings
    { get; set; }
    public DalConfig DalSettings { get; set; }

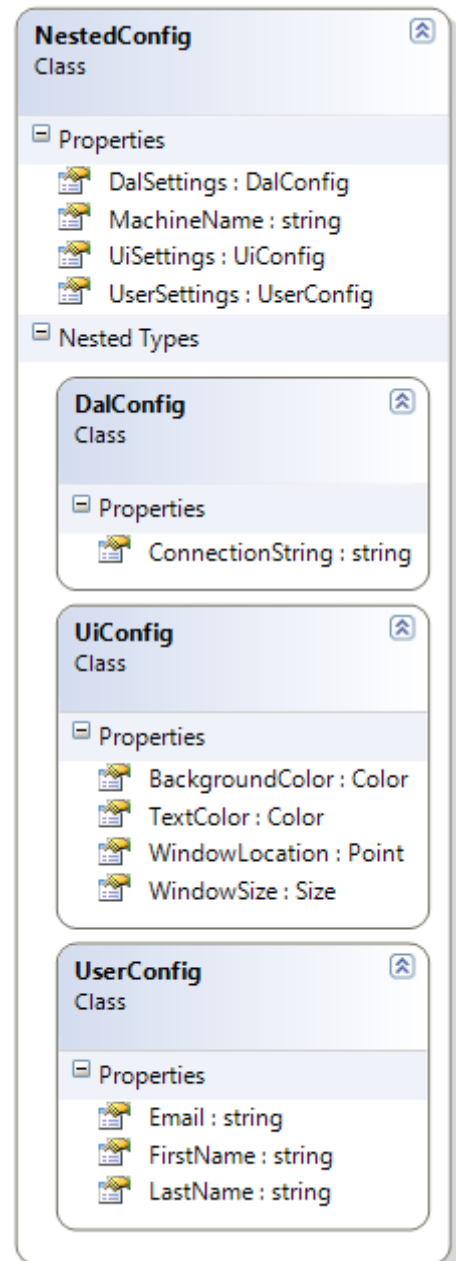
    public class UiConfig
    {
        public Size WindowSize
        { get; set; }
        public Point WindowLocation
        { get; set; }
        public Color BackgroundColor
        { get; set; }
        public Color TextColor
        { get; set; }
    }

    public class UserConfig
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
    }

    public class DalConfig
    {
        public string ConnectionString
        { get; set; }
    }
}
```

This is more complex because it contains nested classes. You see a class diagram of this BO on the right. The design reflects the architecture of the application. There is one Data Access Layer (DAL) which handles all database requests at the lowermost level. Another component is the presentation layer which cares for the user interface (UI).

The last one in this sample is the user component which handles typical user data. We would see more components in a real world application but again, this is just a demo with the purpose to give you an idea of what it's all about. However, all these classes which reflects the different components with their values to be persisted, are contained in a configuration Business object class. The container itself holds a general purpose property (MachineName). This architecture makes it very convenient to use because the developer does not have to deal with several class instances for the data transfer but just a single one!





# XML IO

## Guide

---

### Business Objects (BO)

Now I will show you one last kind of object that some developers might use (could also be an array or something similar to a dictionary):

```
public Dictionary<string, object> Configuration { get; set; }
```

This is pretty awkward because it's not even a Business object! XML IO needs BOs to make it a useful addition to your development work. To make this dictionary a BO we have to modify the code as follows.

```
public class ConfigDictionary
{
    public ConfigDictionary()
    {
        if (Configuration == null)
            Configuration = new Dictionary<string, object>();
    }

    public Dictionary<string, object> Configuration { get; set; }
}
```

Now we have a container class which can be tossed across application components. You can simplify it even more if you make use of `DotNetExpansions.CollectionContainer<T>` or `DotNetExpansions.SharedCollectionContainer<T>`.

Why did I show you these three kinds of BOs anyway? Well simply because to show you that XmlIo is capable to deal with all of them. Yes right, you can use whatever Business Object you like!

# XML IO

## Guide

---

### Under the hood

This API makes use of [DataContract](#)s which is part of WCF (Windows Communication Foundation). The reason for this is to make the user (you) stick to certain conventions. What conventions, you might ask. Well, you can use all of the BOs shown above exactly as they are printed there except for the last one (the DictionaryBO). To make this one work you must tell XmlIo which types you use inside the dictionary. Since the value type is object, it could be anything and that's the problem. „Anything“ cannot be saved in an XML file!

Here is the complete code for the Dictionary-BO.

```
using System.Collections.Generic;
using System.Drawing;
using System.Runtime.Serialization;

namespace ConfigListDemo
{
    [KnownType(typeof(Point))]
    [KnownType(typeof(Size))]
    [KnownType(typeof(Color))]
    public class ConfigDictionaryBO
    {
        public ConfigDictionaryBO()
        {
            if(ConfigDictionary == null)
                ConfigDictionary = new Dictionary<string, object>();
        }

        public Dictionary<string, object> ConfigDictionary { get; set; }
    }
}
```

1. You have to reference System.Runtime.Serialization because we have to use a certain Attribute class.
2. Tell XmlIo which types you will use in the dictionary by setting the [KnownTypeAttribute](#) for each and every type as shown in the code example above. In this sample the dictionary can be persisted by XmlIo as long as you write values of the type of Point, Size, Color, or any primitive type like int, double, string and so on into the dictionary.

# XML IO

## Guide

---

### Samples

Please refer to the DotNetExpansionsHelp.chm file for several detailed samples.  
You can examine even more samples in the Solution, subfolder „Demos\XmlIo Demos“.