

3 Serialisierung und XML

Einführung

Serialisierung bedeutet, vereinfacht ausgedrückt, das »Überleben« der Objekte nach einem Programmende oder die Verpackung der Objekte für den Transport über Prozess- und Maschinengrenzen bei einem Remote-Aufruf. Für OOP-Systeme, die Mechanismen für solche Aktionen boten, wurde dies dadurch gewährleistet, dass aktuelle Daten und Variableninhalte in eine Datei geschrieben wurden. Der Code lag in der ausführbaren Datei vor und nach dem erneuten Start wurde die Datendatei geöffnet und der Entwickler hatte die Aufgabe, alle Daten wieder in die richtigen Strukturen zu laden. Innerhalb des .NET-Frameworks ist XML das Rückgrat für die Serialisierung von Klasseninstanzen. Die für die Nutzung dieser Funktionalität benötigten Namensräume des .NET-Frameworks sind `System.Xml` und `System.Xml.Serialization`.

Terminologie

Serialisierung mit XmlSerializer

Die Klasse `XmlSerializer` ist für die Serialisierung einer Instanz in eine XML-Datei bzw. den umgekehrten Weg zurück zu einer Instanz zuständig. In der einfachsten Form erhält der Konstruktor einen Parameter mit dem Typ des Objekts, für das er zuständig sein soll.

Hier eine sehr einfache Demoklasse, die in eine XML-Datei serialisiert werden soll:

```
// eine einfache demo-klasse
public class SimpleDemo
{
    public string Kennzeichen;
    public int KmStand;
    public double Verbrauch;
}
```

Listing 3.1: Die erste Demoklasse

Als Ziel für die Serialisierung können Sie bei der Methode `Serialize` des `XmlSerializer` drei mögliche Ausgabetypen spezifizieren:

- Einen `XmlWriter` oder einen Nachfahren
- Einen `Stream` oder einen Nachfahren
- Einen `TextWriter` oder einen Nachfahren

Nachdem es hier nur um das Herausschreiben der Instanzdaten geht, verwenden wir im Listing einen `XmlWriter`, da dieser als reine vorwärtsgerichtete Nur-Schreiben-Klasse den geringsten Overhead besitzt.

Hier der Code, mit dem die Instanz serialisiert wird:

```
class TestClass
{
    static void Main(string[] args)
    {
        XmlSerializer ser =
            new XmlSerializer(typeof(SimpleDemo));
        XmlTextWriter tw =
            new XmlTextWriter("ausgabe.xml", null);

        SimpleDemo d = new SimpleDemo();

        d.Kennzeichen = "LA-DY 2002";
        d.KmStand = 45800;
        d.Verbrauch = 7.85;

        ser.Serialize(tw, d);
        tw.Close();

        Console.WriteLine("Objekt serialisiert.");

        Console.Write("\nPress a key ....");
        Console.ReadLine();
    }
}
```

Listing 3.2: Serialisierung in eine XML-Datei

Nach dem erfolgreichen Durchlaufen des Codes liegt die Datei auf der Platte und Sie können sich deren Inhalt ansehen:

Objekte als XML

```
<?xml version="1.0"?>
<SimpleDemo
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Kennzeichen>LA-DY 2002</Kennzeichen>
  <KmStand>45800</KmStand>
  <Verbrauch>7.85</Verbrauch>
</SimpleDemo>
```

Listing 3.3: Der Inhalt der XML-Ausgabedatei

Sie erkennen, dass das Wurzelement den Namen der Klasse trägt und die einzelnen Elemente mit ihrer Bezeichnung den Namen der Felder entsprechen. Zusätzlich werden die beiden Deklarationen der Namens-

räume für XML-Schemas und XML-Schemainstanzen mit in das Wurzelement aufgenommen (mehr zu Namensräumen erfahren Sie im Kapitel »XML-Namensräume«).

Deserialisierung mit XmlSerializer

Um aus einer XML-Datei eine Instanz einer Klasse zu laden, gehen sie einfach den umgekehrten Weg.

```
class TestClass
{
    static void Main(string[] args)
    {
        XmlSerializer ser =
            new XmlSerializer(typeof(SimpleDemo));

        // diesmal einen reader einsetzen
        XmlTextReader rdr =
            new XmlTextReader("ausgabe.xml");

        SimpleDemo d;

        // klasseninstanz aus dem serializer
        d = ser.Deserialize(rdr) as SimpleDemo;
        rdr.Close();

        // instanzdaten ausgeben
        Console.WriteLine("Objekt deserialisiert:");
        Console.WriteLine(d.Kennzeichen);
        Console.WriteLine(d.KmStand);
        Console.WriteLine(d.Verbrauch);

        Console.Write("\nPress a key ....");
        Console.ReadLine();
    }
}
```

Listing 3.4: Deserialisierung aus einer XML-Datei

Dieser Code produziert die in Abbildung 3.1 zu sehende Ausgabe und zeigt damit, dass so sehr leicht aus einer XML-Datei eine Objektinstanz erzeugt werden kann.

Ungleiche Dokumente und Klassen

Falls ein Schema vorliegt, können Sie mit dem folgenden Aufruf aus dem Schema eine passende Klasse erstellen lassen, um dieses Problem zu vermeiden.

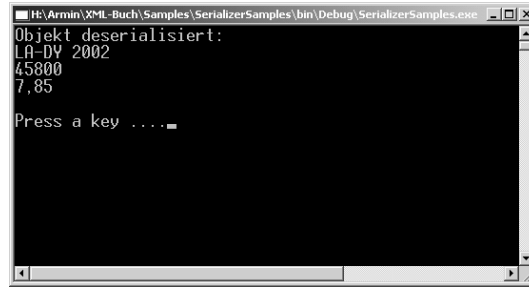


Abbildung 3.1: Die Ausgabe des Deserialisierungsbeispiels

Das Tool `xsd.exe` `xsd schema0.xsd /classes`

Anschließend wird eine Datei `schema0.cs` erzeugt, die eine Klasse mit den aus dem Schema gewonnen Feldern enthält.

```
//-----
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.0.3705.0
//
//     Changes to this file may cause
//     incorrect behavior and will be lost if
//     the code is regenerated.
// </autogenerated>
//-----

//
// This source code was auto-generated by xsd, Version=1.0.3705.0.
//
using System.XML.Serialization;

/// <remarks/>
[System.XML.Serialization.XmlRootAttribute(Namespace="",
IsNullable=true)]
public class SimpleDemo {

    /// <remarks/>
    public string Kennzeichen;

    /// <remarks/>
    public int KmStand;

    /// <remarks/>
    public System.Double Verbrauch;
}
```

Listing 3.5: Automatisch aus einem Schema erzeugte Klasse

Was aber passiert nun, wenn die Definition der Klasse nicht mit dem Aufbau der Datei übereinstimmt? Eine Möglichkeit wäre es, nur Dateien anzunehmen, deren Aufbau genau der Klasse entspricht, indem für die Klasse ein Schema erstellt und das Dokument damit validiert wird. Ein Beispiel dafür finden Sie im Kapitel über die Validierung von XML-Dokumenten.

Für den anderen Fall wird die XML-Datei um ein zusätzliches Element ergänzt, das in der Klasse nicht vorhanden ist. Ebenso wird die Klasse um ein weiteres Feld ergänzt, das in der XML-Datei nicht vorhanden ist.

Zusätzlicher Inhalt in der XML-Datei

```
<?xml version="1.0"?>
<SimpleDemo
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Kennzeichen>LA-DY 2002</Kennzeichen>
  <KmStand>45800</KmStand>
  <Leistung>108</Leistung>
  <Verbrauch>7.85</Verbrauch>
</SimpleDemo>
```

Listing 3.6: Die geänderte XML-Datei

```
// eine einfache demo-klasse
public class SimpleDemo
{
    public string Kennzeichen;
    public int KmStand;
    public double Verbrauch;
    public string Hersteller;
}
```

Listing 3.7: Die geänderte Klasse mit dem neuen Feld

Bei der Ausführung des Programms erfolgt keine Fehlermeldung und die Ausgabe zeigt das Standardverhalten bei der Behandlung unbekannter Elemente in der XML-Datei:



Abbildung 3.2: Deserialisierung mit den geänderten Daten

Elemente ignorieren Zusätzlich vorhandene Elemente werden also bei der Deserialisierung ignoriert, Felder in der Klasse ohne einen Inhalt in der XML-Datei werden von der Laufzeitumgebung bei der Erzeugung der Instanz initialisiert, bekommen dann aber keinen Wert aus der XML-Datei zugewiesen. Dieses Verhalten lässt sich, wie in den späteren Abschnitten gezeigt, auch steuern und beeinflussen.

3.1 XML-Serialisierung im Detail

Steuerung der XML-Struktur

Den Serialisierungsprozess, genauer die Erstellung der XML-Datei, können Sie über den Einsatz verschiedener Attribute steuern. So möchten Sie vielleicht einige Felder der Klasse nicht als eigene Unterelemente, sondern als Attribute des Klassenelementes rendern lassen oder für einige Elemente den Datentyp ändern.

Serialisierungs-Attribute Um das Wurzelement der XML-Datei zu ändern, benutzen Sie vor der Klasse das Attribut `[XmlRoot()]`. Das Attribut `[XmlElement]` dient der Definition eines anderen Elementnamens, der vom Namen des Feldes abweicht. Mit Hilfe des Attributes `[XmlAttribute]` können Sie angeben, dass ein Feld nicht als eigenes Element, sondern als Attribut des umgebenden Elternelements gerendert wird.

```
// eine einfache demo-klasse
[XmlRoot(ElementName = "root")]
public class SimpleDemo
{
    public string Kennzeichen;
    public int KmStand;
    [XmlIgnore]
    public double Verbrauch;
    [XmlAttribute(AttributeName = "marke")]
    public string Hersteller;
}
```

Listing 3.8: Die Klasse mit XML-Steuerungsattributen

Wird die geänderte Klasse jetzt serialisiert, enthält die XML-Datei die neuen Daten, wobei das mit dem Attribut `[XmlIgnore]` deklarierte Feld aus der Datei ausgelassen wird.

```
<?xml version="1.0"?>
<root
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  marke="Renault">
```

```
<Kennzeichen>LA-DY 2002</Kennzeichen>
<KmStand>45800</KmStand>
</root>
```

Listing 3.9: Der neue Inhalt der XML-Datei

Serialisierung komplexerer Datentypen

Für die Serialisierung von Datentypen wie Eigenschaften, Arrays, geschachtelten Objekten usw. gilt im Prinzip das bereits weiter oben gesagte. Für die folgenden Beispiele wird eine etwas erweiterte Klassendefinition benutzt, die über eine öffentliche Eigenschaft, ein Array und Konstruktoren verfügt. Hier der Teil der Klassendefinition mit den entsprechenden Feldern:

```
// demo-klasse für die serialisierung
public class DemoClass
{
    // private felder der klasse
    private int fID;
    private int fIndentLevel = 0;

    // öffentliche felder
    public string StrField = "Nur ein String";
    public System.UInt32[] liste =
        {1,3,5,7,11,13,17,19};

    // konstruktoren
    public DemoClass()
    {
    }

    public DemoClass(int theID)
    {
        fID = theID;
    }

    // eine öffentliche eigenschaft
    public int IndentLevel
    {
        get { return fIndentLevel; }
        set { fIndentLevel = value; }
    }
}
```

Listing 3.10: Der erste Teil der erweiterten Klassendefinition

Diese Klasse soll in der Lage sein, sich selbst zu serialisieren und über eine statische Methode aus einer anzugebenden XML-Datei die Deserialisierung durchzuführen. Dies wird in die beiden Methoden `SaveToXml` und `LoadFromXml` eingepackt.

```
// serialisieren in eine xml-datei
public void SaveToXml(string filename)
{
    XmlSerializer ser =
        new XmlSerializer(typeof(DemoClass));

    // einen xmlwriter einsetzen
    XmlTextWriter xml =
        new XmlTextWriter(filename, null);
    xml.Formatting = Formatting.Indented;
    xml.Indentation = 4;
    xml.IndentChar = (char)32;

    // klasseninsanz serialisieren
    ser.Serialize(xml, this);
    xml.Close();
}

// deserialisieren aus einer xml-datei
public static object LoadFromXml(string filename)
{
    XmlSerializer ser =
        new XmlSerializer(typeof(DemoClass));

    XmlTextReader rdr = new XmlTextReader(filename);

    if(ser.CanDeserialize(rdr))
        return ser.Deserialize(rdr);
    else
        return null;
}

} // ende der klassendefinition
```

Listing 3.11: Die beiden Wrapper für die Serialisierung

Arrays serialisieren

Wird eine Instanz dieser Klasse über den Aufruf von `SaveToXml` in eine XML-Datei geschrieben, wird eine öffentliche Eigenschaft als XML-Element mit dem Namen dieser Eigenschaft geschrieben (also analog zu einem normalen öffentlichen Feld). Das Array wird als XML-Element mit dem Namen des Arrays erstellt und die einzelnen Elemente dieses Arrays werden als Unterelemente, jeweils mit dem Namen des Datentyps, in die Datei geschrieben. Hier die erstellte XML-Datei:

```
<?xml version="1.0"?>
<DemoClass
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <StrField>Hallo!!</StrField>
```



```

<liste>
  <unsignedInt>1</unsignedInt>
  <unsignedInt>3</unsignedInt>
  <unsignedInt>5</unsignedInt>
  <unsignedInt>7</unsignedInt>
  <unsignedInt>11</unsignedInt>
  <unsignedInt>13</unsignedInt>
  <unsignedInt>17</unsignedInt>
  <unsignedInt>19</unsignedInt>
</liste>
<IndentLevel>4</IndentLevel>
</DemoClass>

```

Listing 3.12: Die serialisierte Instanz als XML-Datei

Für die Deserialisierung muss ein parameterloser Konstruktor in der Klasse vorhanden sein (Sie haben sich vielleicht schon darüber gewundert). Da der `XmlSerializer` aus den Daten der XML-Datei nicht erschließen kann, welcher Konstruktor mit welchen Parametern aufzurufen ist, muss ein Konstruktor ohne Parameter vorhanden sein, der dann aufgerufen wird, um die Instanz zu erstellen. Über das Einlesen der restlichen XML-Elemente werden anschließend die gleichlautenden öffentlichen Felder und Eigenschaften gesetzt.

Unbekannten XML-Inhalt behandeln

Sollte in der XML-Datei zusätzlicher Inhalt vorhanden sein (in Form von Elementen, Attributen oder z.B. CDATA-Knoten), dann können Sie über Ereignisbehandlungsroutinen das Standardverhalten der Deserialisierung ändern. Sind keine solchen Eventhandler definiert, wird zusätzlicher Inhalt in der XML-Datei einfach ignoriert und für Felder der Klasse, für die kein Element in der Datei vorhanden ist, wird keine Zuweisung durchgeführt.

Für die Steuerung dieses Verhaltens definiert die Klasse `XmlSerializer` drei Ereignisse: `UnknownElement`, `UnknownAttribute` und `UnknownNode`. Erstellen Sie eine Ereignisbehandlungsroutine für das gewünschte Ereignis, wird diese Methode dann aufgerufen. Für eine Demonstration erweitern wir die Datei aus Listing 3.3 um zusätzliche Elemente.

Unknown-Ereignisse

```

<?xml version="1.0"?>
<SimpleDemo
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Kennzeichen>LA-DY 2002</Kennzeichen>
  <Hersteller>Renault</Hersteller>
  <KmStand>45800</KmStand>
  <Verbrauch>7.85</Verbrauch>

```

```
<Klimaanlage />
</SimpleDemo>
```

Listing 3.13: Der neue Inhalt der XML-Datei

Jetzt wird dafür eine Ereignisbehandlungsroutine definiert:

```
public void XmlElementEventHandler(object sender, XmlElementEventArgs
args)
{
    Console.WriteLine("Bei der Deserialisierung von {0} ",
        args.ObjectBeingDeserialized.ToString());
    Console.WriteLine("wurde ein Element <{0}> gefunden.",
        args.Element.LocalName);
    Console.WriteLine();
}
```

Listing 3.14: Der EventHandler für unbekannte Elemente

Diese Methode wird dem `XmlSerializer` nach der Instantiierung eingetragen.

```
static void Main(string[] args)
{
    XmlSerializer ser =
        new XmlSerializer(typeof(SimpleDemo));
    XmlTextReader rdr =
        new XmlTextReader(@"c:\tmp\ser.xml");

    ser.UnknownElement += new
        XmlElementEventHandler(DoUnknownElement);

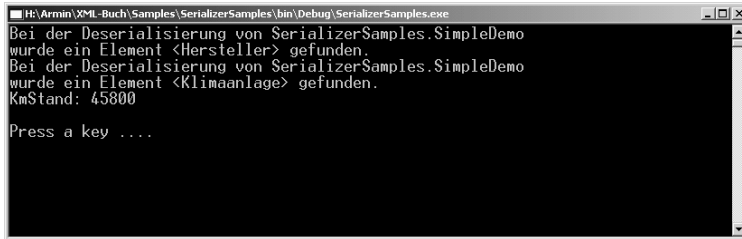
    SimpleDemo sd = (SimpleDemo)ser.Deserialize(rdr);
    rdr.Close();

    Console.WriteLine("KmStand: {0}", sd.KmStand);

    Console.WriteLine("\nPress a key ....");
    Console.ReadLine();
}
```

Listing 3.15: Eintragen der Ereignisbehandlung

Hier das Ergebnis des Programmablaufs:

The screenshot shows a Windows command prompt window titled "H:\Armin\XML-Buch\Samples\SerializerSamples\bin\Debug\SerializerSamples.exe". The text inside the window reads: "Bei der Deserialisierung von SerializerSamples.SimpleDemo wurde ein Element <Hersteller> gefunden.", "Bei der Deserialisierung von SerializerSamples.SimpleDemo wurde ein Element <Klimaanlage> gefunden.", "KmStand: 45800", and "Press a key".

```
H:\Armin\XML-Buch\Samples\SerializerSamples\bin\Debug\SerializerSamples.exe
Bei der Deserialisierung von SerializerSamples.SimpleDemo
wurde ein Element <Hersteller> gefunden.
Bei der Deserialisierung von SerializerSamples.SimpleDemo
wurde ein Element <Klimaanlage> gefunden.
KmStand: 45800
Press a key ....
```

Abbildung 3.3: Die unbekannten XML-Elemente werden gemeldet.

3.2 Zusammenfassung

Serialisierung nach XML ist mit dem .NET-Framework auf einfache Art und Weise möglich, wobei Sie jederzeit in diesen Prozess eingreifen und die Serialisierung der Objektinstanzen genauer steuern können.

