Auf die Reihe gebracht: XML, SOAP und binäre Serialisierung

Das Übertragen von Objekten über Prozessgrenzen hinweg und das persistente Speichern von Objekten ist eine häufige Anforderung in der Software-Entwicklung. Dieser Beitrag stellt die in der .NET-Klassenbibliothek eingebauten Möglichkeiten der Serialisierung und Deserialisierung von Objekten vor.

___ Serialisierung bedeutet die Umwandlung des Zustands eines Objekts in eine Folge von Bytes. Deserialisierung ist der umgekehrte Vorgang, bei dem aus einer Byte-Folge wieder ein programmierbares Objekt erzeugt wird. Dabei wird der ursprüngliche Zustand des Objekts wieder hergestellt. Serialisierung und Deserialisierung sind als Synonyme für die beim Remote Procedure Call (RPC) verwendeten Begriffe *Marshalling* und *Unmarshalling* zu sehen. Abbildung 1 illustriert das Prinzip.

Wann immer Objekte zwischen zwei Umgebungen ausgetauscht werden müssen, die keinen gemeinsamen Speicher besitzen, ist die Serialisierung und spätere Deserialisierung des Objekts notwendig. Im .NET Framework besteht eine solche Barriere nicht nur zwischen Rechnern und Prozessen, sondern auch zwischen Anwendungsdomänen. Daneben wird die Serialisierung/Deserialisierung auch benötigt, wenn Objektpersistenz gewünscht ist, also Objekte dauerhaft auf einem Medium gespeichert werden sollen.

Die .NET-Framework-Klassenbibliothek bietet drei verschiedene Serialisierungsformate: SOAP, XML und ein von Microsoft nicht spezifiziertes und daher proprietäres binäres Format (siehe auch Tabelle 1). Wie die folgenden Ausführungen zeigen werden , hebt sich der XML-Serialisierer nicht nur durch seinen Namen (*Serializer* statt *Formatter*), sondern auch in seiner Funktionsweise und der Bedienung von den anderen beiden Serialisierern erheblich ab.

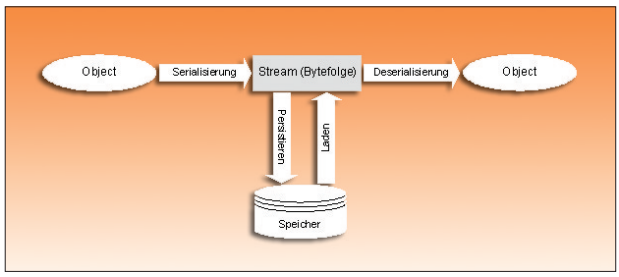


Abbildung 1 | Serialisierung und Persistenz.

Auf den ersten Blick fällt auf: Die Klasse *XmlSerializer* implementiert im Gegensatz zu ihren Verwandten nicht die Schnittstellen *IRemotingFormatter* und *IFormatter*. Auch ist diese Klasse im Namespace *System.Xml* verborgen und nicht in *System.Runtime.Serialization.Formatters* abgelegt.

Serialize und Deserialize

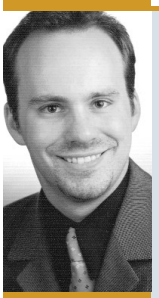
Alle Serialisierer haben gemeinsam, dass sie zwei Methoden *Serialize* und *Deserialize* anbieten. *Serialize* erwartet im ersten Parameter ein *Stream*-Objekt, also ein Objekt einer Klasse, die von *System.IO.Stream* abgeleitet ist, beispielsweise *FileStream*, *MemoryStream*, *NetworkStream*, *CryptoStream*, *BufferedStream,* und im zweiten Parameter das zu serialisierende Objekt.

```
Serializer.Serialize(stream, objekt)
```

Tabelle 1 Serialisierer in der .NET Klassenbibliothek.

Serialisierer	Klasse	Assembly
Binärer Serialisierer	System.Runtime.Serialization.Formatters.Binary.BinaryFormatter	mscorlib.dll
SOAP- Serialisierer	System.Runtime.Serialization.Formatters.Soap.SoapFormatter	System.Runtime.Serialization.Formatters.Soap.dll
XML- Serialisierer	System.Xml.Serialization.XmlSerialization	System.Xml.dll

SUMMARY



Auf einen Blick
Die Klassenbibliothek des .NET Frameworks enthält drei verschiedene Möglichkeiten, Objekte für die Übertragung zwischen Prozess- oder Rechnergrenzen beziehungsweise für das persistente Speichern zu serialisieren. Der Beitrag vergleicht die Serialisierung und Deserialisierung mit dem BinaryFormatter, dem SOAPFormatter und dem XML-Serializer.

Eingesetzte Anwendungen
.NET Framework 1.0 / 1.1, Visual Basic .NET

CD-Code
DrillDown03

Autor
Holger Schwichtenberg ist selbstständiger Programmierer, Consultant, Dozent und Autor. Er unterrichtet an Hochschulen und spricht regelmäßig auf Konferenzen zu Programmierthemen im Microsoft-Umfeld. Bei Addison-Wesley und Microsoft Press hat er mehrere Programmierbücher veröffentlicht. Sie erreichen ihn über seine Webseite www.DotNetFramework.de.

Listing 1 Hilfsroutine zur Serialisierung in eine Datei.

```
Public Enum SerialTyp
 SOAP
 XML
 BINARY
End Enum

' ### Serialisieren in Datei
Sub save(ByVal Typ As SerialTyp, ByVal obj As Object, ByVal datei As _
String)
 ' Datei öffnen
 Dim stream As FileStream
 stream = New FileStream(datei, FileMode.Create, FileAccess.Write, _
 FileShare.None)
 ' Fallunterscheidung
 Dim Serializer As Object
 Select Case Typ
  Case SerialTyp.BINARY
   Serializer = New BinaryFormatter()
  Case SerialTyp.SOAP
   Serializer = New SoapFormatter()
  Case SerialTyp.XML
   Serializer = New XmlSerializer(obj.GetType)
  Case Else
   MsgBox("Nicht unterstütztes Serialisierungsformat!")
   End
 End Select
 ' Serialisierung starten
 Serializer.Serialize(stream, obj)
 ' Datei schließen
 stream.Close()
 ' Ausgabe
 out("Objekt wurde gespeichert in " & datei)
End Sub
```

Listing 2 Hilfsroutine zur Deserialisierung aus einer Datei.

```
' ### Deserialisieren aus Datei
Function load(ByVal typ As SerialTyp, ByVal Datei As String, Optional _
ByVal Objekttyp As Type = Nothing) As Object
 ' Datei öffnen
 Dim stream As FileStream
 stream = New FileStream(Datei, FileMode.Open)
 ' Fallunterscheidung
 Dim Serializer As Object
 Select Case typ
  Case SerialTyp.BINARY
   Serializer = New BinaryFormatter()
  Case SerialTyp.SOAP
   Serializer = New SoapFormatter()
  Case SerialTyp.XML
   If Objekttyp Is Nothing Then
    MsgBox("Fehler: Für den XML-Deserialisierer muss der zu _
    deserialisierende Objekttyp bekannt sein!")
    End
   End If
   Serializer = New XmlSerializer(Objekttyp)
  Case Else
   MsgBox("Nicht unterstütztes Serialisierungsformat!")
   End
 End Select
 ' Deserialisierung
 Dim o As Object
 o = Serializer.Deserialize(stream)
 ' Datei schließen
 stream.Close()
 ' Ausgabe
 out("Objekt wurde geladen aus " & Datei)
 ' Objekt zurückliefern
 Return o
End Function
```

Deserialize erwartet nur ein *Stream*-Objekt und liefert das deserialisierte Objekt als Rückgabewert.

```
objekt = Serializer.Deserialize(stream)
```

Das Speichern eines Objekts in einer Datei kapselt die Routine *Save* in Listing 1. Bei der Deserialisierung aus einer Datei, gekapselt in der Routine *Load* in Listing 2, wird ein erster Unterschied zwischen dem *BinaryFormatter* und dem *SoapFormatter* einerseits und dem *XmlSerializer* andererseits deutlich: Der *XmlSerializer* will vorher wissen, welchen Objekttyp er deserialisieren wird und erwartet daher bei seiner Instanzierung ein Objekt vom Typ *System.Type*. Die in beiden Hilfsroutinen verwendete Enumeration *SerialTyp* stammt nicht aus dem .NET Framework, sondern ist selbst definiert (siehe Listing 1).

Beim Deserialisieren mit dem Binär- und SOAP-Serialisierer wird kein Konstruktor aufgerufen, während der XML-Serialisierer den parameterlosen Konstruktor immer aufruft und daher erwartet, dass ein solcher in der entsprechenden Klasse existiert.

Noch ein Unterschied des XML-Serialisierers: Die Methode *Serialize* in der Klasse *XmlSerializer* kann nicht nur in Streams, sondern auch in von *System.IO.TextWriter* und *System.Xml.XmlWriter* abgeleitete Klassen serialisieren. Umgekehrt arbeitet *Deserialize* auch mit von *System.IO.TextReader* und *System.Xml.XmlReader* abgeleiteten Klassen.

Beispiel

Die Objektserialisierung soll nun an einem Beispiel veranschaulicht werden. Das im Folgenden verwendete Szenario zeigt die Abbildung 2 in Form eines Objektdiagramms mit den Klassen *Buchautor*, *Buch*, *Auflage* und *Verlag*. Zur Klasse *Buchautor* existiert eine Basisklasse *Autor*, die im Gegensatz zur Unterklasse *Buchautor* kein Attribut *Buecher* besitzt (Listing 3).

Ein Objekt kann nur serialisiert werden, wenn die Klasse mit dem Meta-Attribut *<Serializable()>* versehen wurde. Ohne dieses Attribut kommt es zum Fehler *The type is not marked as serializable.* Außerdem muss die Klasse als *Public* deklariert werden.

Die Anwendung der Routinen *save* und *load* zeigen die Listings 4 und 5 anhand der Klasse *Autor*. Das Ergebnis dieser Beispiele sehen Sie in den Abbildungen 3 bis 5. Im direkten Vergleich der Ergebnisse des SOAP-Serialisierers und des XML-Serialisierers fallen zwei Unterschiede sofort auf:

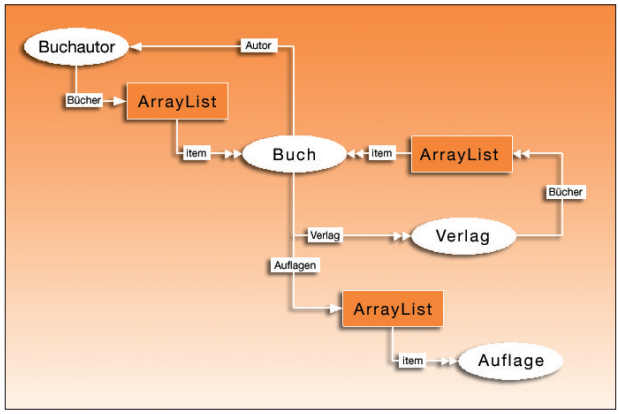


Abbildung 2 | Die Objekthierarchie der Klassen aus Listing 3.

Listing 3 Klassendefinition für die Serialisierungsbeispiele in diesem Beitrag.

```
<Serializable()> Public Class Autor
 Public Name As String
 Public Geschlecht As Char
 Private _Geb As Date
 Public Schwerpunkte As String()

 Public Property Geb()
…
 End Property

 Sub New()
  out("Parameterloser Konstruktor der Klasse 'Autor' aufgerufen...")
 End Sub

 Sub New(ByVal Name As String)
  Name = Name
  out("Konstruktor der Klasse 'Autor' aufgerufen...")
 End Sub
End Class

' ======================================

<Serializable(), XmlInclude(GetType(Auflage))> Public Class Buch
 Public Titel As String
 Public Autor As Autor
 Public Auflagen As New ArrayList()
 <NonSerialized()> Public Deserialisiert As Date
 Public Verlag As Verlag

 Sub New() :  End Sub

 Sub New(ByVal Titel As String)
  Me.Titel = Titel
 End Sub

End Class

' ======================================

<Serializable(), XmlInclude(GetType(Buch))> Public Class Buchautor
 Inherits Autor
 Public buecher As New ArrayList()

 Sub New(ByVal Name As String)
  Name = Name
  out("Konstruktor der Klasse 'Buchautor' aufgerufen...")
 End Sub

 Sub New()
  out("Parameterloser Konstruktor der Klasse 'Buchautor' aufgerufen...")
 End Sub

End Class

' ======================================

<Serializable(), XmlInclude(GetType(Buch))> Public Class Verlag
 Public Name As String
 Public Ort As String
 Public Buecher As New ArrayList()

 Sub New() :  End Sub

 Sub New(ByVal Verlagsname As String)
  Name = Verlagsname
 End Sub
End Class

' ======================================

<Serializable()> Public Class Auflage
 Public Nr As Byte
 Public Jahr As Integer
 Public ISBN As String

 Sub New() :  End Sub

 Sub New(ByVal Nr As Byte, ByVal Jahr As Integer, ByVal ISBN As String)
…
 End Sub
End Class
```

- Während der XML-Serialisierer das Objekt in einer Baumstruktur wiedergibt, arbeitet der SOAP-Serialisierer mit eindeutigen Objektnummern (*id="ref-x"*) und Verweisen anhand dieser Objektnummern (*href="#ref-x"*).
- Der SOAP-Serialisierer serialisiert auch private Klassenmitglieder (wie *_Geb*), während der XML-Serialisierer diese nicht beachtet, allerdings dafür im Gegensatz zu seinem SOAP-Kollegen Aufrufe der vorhandenen Property-Routinen durchführt.

```
00000000  00 01 00 00 00 FF FF FF  FF 01 00 00 00 00 00 00  ................
00000010  00 0C 02 00 00 00 4C 64  6F 74 6E 65 74 70 72 6F  ......Ldotnetpro
00000020  5F 30 33 30 33 2C 20 56  65 72 73 69 6F 6E 3D 31  _0303, Version=1
00000030  2E 31 2E 31 30 38 34 2E  33 34 30 34 34 2C 20 43  .1.1084.34044, C
00000040  75 6C 74 75 72 65 3D 6E  65 75 74 72 61 6C 2C 20  ulture=neutral,
00000050  50 75 62 6C 69 63 4B 65  79 54 6F 6B 65 6E 3D 6E  PublicKeyToken=n
00000060  75 6C 6C 05 01 00 00 00  14 64 6F 74 6E 65 74 70  ull.....dotnetp
00000070  72 6F 5F 30 33 30 33 2E  41 75 74 6F 72 04 00 00  ro_0303.Autor...
00000080  00 04 4E 61 6D 65 0A 47  65 73 63 68 6C 65 63 68  ..Name.Geschlech
00000090  74 04 5F 47 65 62 0C 53  63 68 77 65 72 70 75 6E  t._Geb.Schwerpun
000000a0  6B 74 65 01 00 00 06 03  0D 02 00 00 00 06 03 00  kte.............
000000b0  00 00 15 48 6F 6C 67 65  72 20 53 63 68 77 69 63  ...Holger Schwic
000000c0  68 74 65 6E 62 65 72 67  6D 00 C0 3F 33 F9 64 A2  htenbergm..?3.d.
000000d0  08 09 04 00 00 00 11 04  00 00 00 03 00 00 00 06  ................
000000e0  05 00 00 00 09 53 63 72  69 70 74 69 6E 67 06 06  .....Scripting..
000000f0  00 00 00 03 43 4F 4D 06  07 00 00 00 04 2E 4E 45  ....COM.......NE
00000100  54 0B                                            T.
```

Abbildung 3 | Ergebnis einer binären Serialisierung.

Tabelle 2 Verhalten der Serialisierer beim Deserialisieren eines Objekts, dessen Klasse sich geändert hat.

Änderung	Binär-Serialisierer	SOAP-Serialisierer	XML-Serialisierer
Umbenennen eines Attributs	Fehler	Fehler	Kein Fehler, aber Attribut wird ignoriert
Hinzufügen eines Attributs	Fehler	Fehler	OK
Entfernen eines Attributs	Fehler	Fehler	OK
Ändern des Datentyps von Byte in Long	OK	OK	OK
Ändern des Datentyps von Byte in String	Fehler	OK	OK
Ändern des Datentyps von Long in Byte	Fehler	Nur OK, wenn neuer Typ den serialisierten Wert aufnehmen kann	

Änderungen an serialisierten Objekten

Eine interessante Frage ist, was passiert, wenn sich nach dem Serialisieren die Klasse ändert, zu der das Objekt gehört. Grundsätzlich ist der XML-Serialisierer am unempfindlichsten. Die Tabelle 2 zeigt dazu Detailinformationen anhand typischer Fälle. Eine Lö-

Abbildung 4 | Ergebnis einer SOAP-Serialisierung.

Listing 4 Beispiel zur Serialisierung in eine Datei.

```
' === Beispiel für Serialisierung
Sub Serialisieren_Test()
 ' -- Objekt erzeugen und füllen

 Dim a As New Autor()
 a.Vorname = "Holger" :   a.Name = "Schwichtenberg"
 a.Geb = #8/1/1972# :   a.Geschlecht = "m"
 a.Schwerpunkte = New String() {"Scripting", "COM", ".NET"}

 ' -- Serialisierungsziele
 Const DATEI = "..\_daten\Serial\Autor.bin"
 Const DATEISOAP = "..\_daten\Serial\Autor.soap.xml"
 Const DATEIXML = "..\_daten\Serial\Autor.xml"

 ' -- binäres Serialisieren
 save(SerialTyp.BINARY, a, DATEI)
 a = load(SerialTyp.BINARY, DATEI)
 out(a.Vorname & " " & a.Name & " wurde am " & a.Geb & " geboren!")

 ' -- SOAP Serialisieren
 save(SerialTyp.SOAP, a, DATEISOAP)
 a = load(SerialTyp.SOAP, DATEISOAP)
 out(a.Vorname & " " & a.Name & " wurde am " & a.Geb & " geboren!")

 ' -- XML Serialisieren
 save(SerialTyp.XML, a, DATEIXML)
 a = load(SerialTyp.XML, DATEIXML, a.GetType)
 out(a.Vorname & " " & a.Name & " wurde am " & a.Geb & " geboren!")

End Sub
```

Listing 5 Beispiel zur Deserialisierung aus einer Datei.

```
' === Beispiel für Deserialisierung
Sub Deserialisieren_Test()
 Dim a As Autor
 ' -- KONSTANTEN SIEHE LISTING 4 !!!
...
 ' -- binäres Deserialisieren
 a = load(SerialTyp.BINARY, DATEI)
 out(a.Vorname & " " & a.Name & " wurde am " & a.Geb & " geboren!")
 ' -- SOAP Deserialisieren
 a = load(SerialTyp.SOAP, DATEISOAP)
 out(a.Vorname & " " & a.Name & " wurde am " & a.Geb & " geboren!")
 ' -- XML-Deserialisieren
 a = load(SerialTyp.XML, DATEIXML, a.GetType)
 out(a.Vorname & " " & a.Name & " wurde am " & a.Geb & " geboren!")
End Sub
```

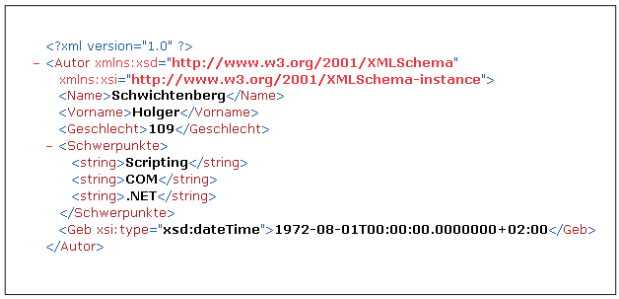


Abbildung 5 | Ergebnis einer XML-Serialisierung.

sung dieser Änderungskonflikte liegt in der Verwendung der benutzerdefinierten Deserialisierung über die Schnittstelle *ISerializable*, auf die später noch eingegangen wird.

Objektmodell serialisieren

Die .NET-Serialisierer unterstützen nicht nur die Serialisierung einzelner Objekte, sondern auch die kompletter Objektbäume. Der Code in Listing 6 erzeugt einzelne Objekte sowie einen kompletten Objektbaum aus den in Listing 3 definierten Klassen. Die Routinen *load* und *save* können genauso wie im Fall des Einzelobjekts angewendet werden. Während der SOAP- und der Binär-Serialisierer einen beliebigen Objektbaum komplett zur Laufzeit analysieren können und mit einem *System.Object* kein Problem haben, möchte der XML-Serialisierer vor Arbeitsbeginn wissen, was ihn in der ArrayList erwartet, mit der in dem vorliegenden Beispiel die Objektmengen realisiert werden.

Aus diesem Grund sind in Listing 3 vor jeder Klasse, die eine ArrayList verwendet, Meta-Attribute des Typs *System. Xml.Serialization.XmlInclude* integriert. In *XmlInclude* muss der Entwickler alle Typen angeben, die in der *ArrayList* vorkommen könnten. Ohne die Verwendung von *XmlInclude* beschwert sich der XML-Serialisierer mit *The type was not expected.* Der Grund dafür ist,

Listing 6 Aufbau eines Objektbaums aus den Klassen Buch, Buchautor, Verlag und Auflage.

```
' -- Objektmodell aufbauen
 Dim a As New Buchautor("Holger Schwichtenberg")
 a.Geb = #8/1/1972#
 Dim v As New Verlag()
 v.Name = "Addison Wesley"

 Dim b1 As New Buch("Windows Scripting")
 b1.Auflagen.Add(New Auflage(1, 2000, "3-8273-1637-5"))
 b1.Auflagen.Add(New Auflage(2, 2001, "3-8273-1843-2"))
 b1.Auflagen.Add(New Auflage(3, 2002, "3-8273-2061-5"))
 b1.Verlag = v

 Dim b2 = New Buch("ASP.NET - Das Entwicklerbuch")
 b2.Auflagen.add(New Auflage(1, 2002, "3-86063-667-7"))
 b2.verlag = New Verlag("MSPress")

 Dim b3 = New Buch("Programmieren mit der .NET-Klassenbibliothek")
 b3.Auflagen.Add(New Auflage(1, 2002, "3-8273-1905-6"))
 b3.verlag = v

 a.buecher.Add(b2)
 a.buecher.Add(b1)
 a.buecher.Add(b3)
```

dass der *XmlSerializer* mithilfe des Code-DOM eine eigene Serialisierungsroutine für jede Klasse generiert und übersetzt.

Abbildung 6 | Der vom Buchautor ausgehende Objektbaum als XML-Dokument (Ausschnitt).

Abbildung 7 | Der vom Buchautor ausgehende Objektbaum als SOAP-Nachricht (Ausschnitt).

Zirkuläre Referenzen

Das Beispiel in Listing 6 erzeugt bewusst einen echten Objektbaum ohne zirkuläre Referenzen. Ergänzt man die folgenden drei Zeilen, die einen Rückverweis von den einzelnen Büchern auf deren Autor herstellen, so verweigert der XML-Serialisierer sofort seinen Dienst:

```
b1.Autor = a
b2.Autor = a
b3.Autor = a
```

Der XML-Serialisierer beantwortet jeden Versuch, eine Objekthierarchie mit zirkulären Referenzen zu serialisieren, mit der Fehlermeldung *There was an error generating the XML document. —-> System.InvalidOperationException: A circular reference was detected while serializing an object.* Der Grund dafür ist, dass der XML-Serialisierer einen einzelnen Baum von XML-Elementen erzeugt, in dem jedes Objekt an die Stelle einsortiert wird, an der es referenziert wird (Abbildung 6). Wenn ein einzelnes Objekt in einem Objektbaum öfter vorkommt, wird es durch den XML-Serialisierer mehrfach serialisiert. Bei der Deserialisierung entstehen dann zwei voneinander unabhängige Objekte mit gleichem Inhalt. Der XML-Serialisierer dagegen baut das Objekt bei jeder Referenzierung erneut in den XML-Dokumentenbaum ein, und eine zirkuläre Referenz würde zu einem Endlosdokument führen.

Mit Objektnummern (*id="ref-x"*) und Verweisen anhand dieser Objektnummern (*href="#ref-x"*) erreicht der SOAP-Serialisierer, dass jedes Objekt nur einmal gespeichert werden muss, auch wenn es mehrfach referenziert wird (Abbildung 7). Somit erklärt sich auch, warum der SOAP-Serialisierer zirkuläre Referenzen umwandeln kann, der XML-Serialisierer aber nicht. Auch der binäre Serialisierer hat keine Probleme mit zirkulären Referenzen.

Serialisieren in Byte-Folge oder Datenbank

Flexiblere Möglichkeiten als die direkte Serialisierung in eine Datei eröffnet die Serialisierung innerhalb des Speichers in eine Byte-Folge (Listing 7). Sie kann dann beliebig weiterverwendet werden, also zum Beispiel auch in einem BLOB-Feld einer Datenbank geschrieben werden.

Die Möglichkeit, eine Byte-Folge zu gewinnen, eröffnet die Klasse *System.IO.MemoryStream*. Das *MemoryStream*-Objekt benötigt keinen Parameter, die eigentliche Serialisierung mit *Serialize* ist gleich. Um die Byte-Folge in Form eines Array of Byte zu erhalten, muss die Methode *ToArray* des *MemoryStream*-Objekts aufgerufen werden.

Beim Deserialisierungsvorgang aus einer Byte-Folge kann ausgenutzt werden, dass sich ein *MemoryStream*-Objekt aus einer Byte-Folge erzeugen lässt. Die Hilfsroutinen in Listing 7 und 8 sind Varianten der bereits vorgestellten Serialisierung in eine Datei.

Allerdings kann auch die *Serialize*-Methode des *XML*-Serialisierers nur mit Stream- und Writer-Objekten, nicht aber mit String-Objekten arbeiten. Zur In-Memory-Serialisierung in einen String muss auch hier die *MemoryStream*-Klasse verwendet werden. Zwar gibt es keine direkte Methode, um den Stream-Inhalt als String zu erhalten, aber zwei Wege führen zum Ziel:

- Lesen des *MemoryStream*-Objekts über ein *StreamReader*-Objekt oder
- Umwandeln des *MemoryStream*-Objekts in eine Byte-Folge, zeichenweise Umwandlung in ein *Char*-Objekt und Zusammensetzen zu einem String.

Bei der Deserialisierung aus einer Zeichenkette, die ein XML-Dokument enthält, ist ein kleiner unschöner Winkelzug notwendig. Grundsätzlich kann die Methode *Deserialize* genauso wenig wie *Serialize* direkt einen String verarbeiten. Daher ist ein *MemoryStream* notwendig. Um diesen zu beschreiben, verwendet man ein *StreamWriter*-Objekt. Dieses hat die Eigenschaft, dass es sein Werk nur dann zu Ende bringt, wenn es ordnungsgemäß geschlossen wird. Mit dem Schließen des *StreamWriter*-Objekts mit *Close* wird aber gleichzeitig auch der Stream geschlossen, weshalb der Deserialisierer nicht mehr darauf zugreifen kann. Der Inhalt des *MemoryStream*-Objekts steht aber noch via *ToArray* zur

Listing 7 Serialisieren in eine Byte-Folge.

```
' ### Serialisieren in Byte-Folge
Function bin_serialize(ByVal o) As Byte()
  Dim form As New BinaryFormatter()
  Dim stream As New MemoryStream()
  form.Serialize(stream, o)
  stream.Close()
  out("Objekt serialisiert!")
  Return (stream.ToArray)
End Function

' ### Deserialisieren aus einer Byte-Folge
Function bin_deserialize(ByVal ba As Byte()) _
  As Object
  Dim o As Object
  Dim form As New BinaryFormatter()
  Dim stream As New MemoryStream(ba)
  o = form.Deserialize(stream)
  stream.Close()
  out("Objekt wurde deserialisiert")
  Return o
End Function
```

Verfügung. Daher muss der Inhalt in ein anderes *MemoryStream*-Objekt umkopiert werden (Listing 8).

Auf der Heft-CD finden Sie ein komplettes Beispiel, das ein Objekt in einem BLOB- beziehungsweise String-Feld in einer Access-Datenbank ablegt.

ObjectSpaces

Im .NET Framework 1.0 und 1.1 gibt es keine spezielle Unterstützung für objektrelationales Mapping, also die Möglichkeit, Objektattribute gezielt auf einzelne Tabellenspalten in einer Relation abzubilden. Microsoft bietet dazu schon seit Ende 2001 einen Technology Preview für eine Klassenbibliothek mit Namen *ObjectSpaces* (Namespace *Microsoft.ObjectSpaces*).

Mehr dazu erfahren Sie unter *http://groups.msn.com/ObjectSpaces* und in der Newsgroup *microsoft.public.objectspaces*.

Benutzerdefinierte (De-)Serialisierung

Der Entwickler hat die Möglichkeit, selbst zu steuern, welche Attribute wie serialisiert werden. Auch hier gibt es die Fraktionstrennung: Die Steuerung ist unterschiedlich in den Serialisierern in *System.Runtime.Serialization.Formatters* und dem XML-Serialisierer in *System.Xml.Serialization*.

Der binäre Serialisierer und der SOAP-Serialisierer bieten drei unterschiedliche Konzepte zur Steuerung der Serialisierung:

- Mit dem Meta-Attribut *<System.NonSerialized()>* kann ein Attribut von der Serialisierung ausgenommen werden.
- Über die Schnittstelle *ISerializable* kann die Form der Serialisierung einzelner Attribute gesteuert werden.
- Über die Schnittstelle *IDeserializationCallback* kann die Klasse nach der Deserialisierung aufgerufen werden.

ISerializable und IDeserializationCallback

Zur Steuerung der Serialisierung beim binären Serialisieren und beim Serialisieren via SOAP muss die zu serialisierende Klasse die Schnittstelle *ISerializable* anbieten. Darin ist Folgendes zu implementieren:

Listing 8 Serialisieren/Deserialisieren mit einem XML-String.

```
'' ### XML-Serialisieren in String
 Function xml_deserialize_string(ByVal obj As Object) As String
  Dim serializer As New XmlSerializer(obj.GetType)
  Dim s As String

  ' -- Serialisieren in MemoryStream
  Dim ms As New MemoryStream()
  serializer.Serialize(ms, obj)

  ' -- Stream in String umwandeln
  Dim r As StreamReader = New StreamReader(ms)
  r.BaseStream.Seek(0, SeekOrigin.Begin)
  s = r.ReadToEnd

  ' Alternative:
  'Dim b As Byte
  'Dim ba As Byte()
  'ba = ms.ToArray
  'For Each b In ba
  '  s = s & Convert.ToChar(b)
  'Next

  Return s
 End Function

' ### Deserialisieren aus XML-String
Function xml_deserialize_string(ByVal t As Type,_
  ByVal s As String) As Object

  Dim obj As Object
  ' -- Objekt in Stream kopieren
  Dim stream As New MemoryStream()
  Dim w As New StreamWriter(stream)
  w.BaseStream.Seek(0, SeekOrigin.End)
  w.WriteLine(s)
  w.Close()
  ' Stream umkopieren, weil jetzt geschlossen
  stream = New MemoryStream(stream.ToArray)
  Dim serializer As New XmlSerializer(t)
  obj = serializer.Deserialize(stream)
  stream.Close()
  Return (obj)
End Function
```

- Die Methode *GetObjectData()* mit nachfolgender Signatur:

```
Public Sub GetObjectData( _
  ByVal info As SerializationInfo, _
  ByVal context As StreamingContext) _
  Implements ISerializable.GetObjectData
```

  Das *SerializationInfo*-Objekt nimmt dabei die zu serialisierenden Daten als Attribut-Wert-Paare auf.
- Ein Konstruktor, der als Parameter ein *SerializationInfo*-Objekt und ein *StreamingContext*-Objekt erwartet. Dieser Konstruktor wird bei der Deserialisierung aufgerufen.

```
Sub New(ByVal info As SerializationInfo, _
  ByVal context As StreamingContext)
```

  Das *SerializationInfo*-Objekt übergibt dabei die serialisierten Daten als Attribut-Wert-Paare.
- Falls nicht schon existent, muss ein parameterloser Konstruktor implementiert werden.

```
Public Sub New()
```

Wenn diese drei Methoden vorhanden sind, ruft der Serialisierer während des Serialisierens automatisch die Methode *GetObjectData*

Listing 9 Benutzerdefinierte binäre Serialisierung.

```
Public Module Serialisieren_Binaer_mit_Kontrolle
  <Serializable()> Public Class Person
    Implements ISerializable, _
               IDeserializationCallback
    Public Name As String
    Public Vorname As String
    …
    Public Taetigkeiten As String()
    Public Deserialisiert As Date

    Public Sub New()  :  End Sub

    ' -- Serialisierung ausgewählter Eigenschaften
    Public Sub GetObjectData( _
      ByVal info As SerializationInfo, _
      ByVal context As StreamingContext) _
      Implements ISerializable.GetObjectData
      out("Serialisierung des Typs: " & _
      info.FullTypeName.ToString)
      info.AddValue("kompletterName", Vorname & " " & Name)
      info.AddValue("Geb", Geb)
    End Sub

    ' Deserialisierung ausgewählter Eigenschaften
    Public Sub New( _
      ByVal info As SerializationInfo, _
      ByVal context As StreamingContext)
      Dim namen As String()
      namen = CType(info.GetValue( _
       "kompletterName", GetType(String)), _
       String).Split
      Vorname = namen(0)
      Name = namen(1)
      Geb = CDate(info.GetValue("Geb", _
                GetType(Date)))
    End Sub

    ' wird am Ende der Deserialisierung aufgerufen
    Public Sub OnDeserialization( _
      ByVal s As Object) Implements _
      DeserializationCallback.OnDeserialization
      Deserialisiert = Now
      out("Deserialisierung komplett!")
    End Sub
  End Class

  Sub serial_kontrolle_1()
  Const DATEI = "h:\Data\person.bin"
    ' -- Objekt füllen
    Dim p As New Person()
    p.Vorname = "Holger" : p.Name = "Schwichtenberg"
    p.Taetigkeiten = …

    Dim form As New BinaryFormatter()
    Dim stream As FileStream

    ' -- Serialisieren
    stream = New FileStream(DATEI, __
      FileMode.Create, FileAccess.Write, __
      FileShare.None)
    form.Serialize(stream, p)
    stream.Close()
    out("Objekt wurde gespeichert!")
    p = Nothing

    ' -- Deserialisieren
    stream = New FileStream(DATEI, _
      FileMode.Open, FileAccess.Read, _
      FileShare.Read)
    p = form.Deserialize(stream)
    stream.Close()
    out("Objekt wurde geladen!")
    out(p.Vorname & " " & p.Name & " wurde am " __
        & p.Geb & " geboren!")
  End Sub
End Module
```

auf und während des Deserialisierens den spezifischen Konstruktor. Eine zu serialisierende Klasse kann auch *IDeserializationCallback* implementieren. Diese Schnittstelle umfasst nur eine Methode, *OnDeserialization(ByVal s As Object)*, die aufgerufen wird, wenn die Deserialisierung beendet ist.

Diese Methode zu füllen ist sinnvoll, um Daten des Objekts zu vervollständigen, die nicht serialisiert wurden oder aus serialisierten Daten errechnet werden.

Das Beispiel in Listing 9 zeigt die Klasse *Person*, die nur die Attribute *Vorname*, *Name* und *Geb* serialisiert. Die beiden Namen werden dabei in einem Feld serialisiert und bei der Deserialisierung wieder getrennt (dieses Vorgehen ist natürlich nicht zwingend notwendig – es dient hier nur zu Demonstrationszwecken). In dem Attribut *Deserialisiert* werden Datum und Uhrzeit am Ende der Deserialisierung erfasst.

Benutzerdefinierte Serialisierung für den XML-Serializer

Die Steuerung der Serialisierung beim XML-Serialisierer erfolgt durch Meta-Attribute. Dafür seien vier Beispiele genannt:

- Mit *<XmlElement("Name")>* erhält ein Klassenattribut im XML-Dokument einen vom Typnamen abweichenden Elementnamen.
- *<XmlAttributeAttribute()>* bewirkt, dass das damit versehene Klassenattribut in ein XML-Attribut und nicht – wie es Standard ist – in ein XML-Element serialisiert wird.
- Das Meta-Attribut *<XmlIgnore()>* bewirkt, dass das damit versehene Klassenattribut gar nicht serialisiert wird.
- Schließlich kann man *XmlRoot("Name")* auf eine Klasse anwenden, um das Wurzelelement des XML-Dokuments anders als mit dem Typnamen zu benennen.

Listing 10 Steuerung der Serialisierung durch XML-Attribute.

```
<Serializable(), XmlInclude(GetType(Buch)), XmlRoot("BAutor")> Public
Class Buchautor
 Inherits Autor
 <XmlElement("GeschriebeneBuecher")> Public buecher As New ArrayList()
 <XmlIgnore()> Public Gehalt As Single
<XmlAttributeAttribute()> Public Aktiv As Boolean = True
 Sub New(ByVal Name As String)
  Name = Name
  out("Konstruktor der Klasse 'Buchautor' aufgerufen...")
 End Sub
 Sub New()
  out("Parameterloser Konstruktor der Klasse 'Buchautor'
aufgerufen...")
 End Sub
End Class
```

Tabelle 3 Die Unterschiede zwischen den drei Serialisierern im Überblick.

	Binär	SOAP	XML
Klasse	BinaryFormatter	SOAPFormatter	XmlSerializer
Namespace	System.Runtime.Serialization.Formatters.Binary	System.Runtime.Serialization.Formatters.Soap	System.Xml.Serialization
Oberklasse	System.Object	System.Object	System.Object
Implementierte Schnittstellen	IRemotingFormatter, IFormatter	IRemotingFormatter, IFormatter	Keine
Ausgabeform	Nicht spezifiziertes binäres Format	SOAP mit eindeutigen Objektreferenzen	Echter Baum von XML-Elementen
Typ der zu serialisierenden Klasse muss explizit angegeben werden	Nein	Nein	Ja
Serialisierte Klasse muss Public sein	Nein	Nein	Ja
Serialisierung privater Mitglieder	Ja	Ja	Nein
Aufruf der Property-Routinen	Nein	Nein	Ja
Objekthierarchien mit zirkulären Referenzen	Ja	Ja	Nein
Steuerung der Serialisierung	Über ISerializable und ein Meta-Attribut	Über ISerializable und ein Meta-Attribut	Über Meta-Attribute
Steuerung der Deserialisierung	Über IDeserialization-Callback	Über IDeserialization-Callback	Nicht möglich
Serialisierung in Streams	Ja	Ja	Ja
Serialisierung in TextWriter und XmlWriter	Nein	Nein	Ja
Aufruf des parameterlosen Konstruktors beim Deserialisieren	Nein	Nein	Ja

```
<BAutor Aktiv="true">
  <Geschlecht>0</Geschlecht>
  <Geb xsi:type="xsd:dateTime">1972-08-01T00:00:00.0000000+02:00</Geb>
  <GeschriebeneBuecher xsi:type="Buch">
    <Titel>ASP.NET - Das Entwicklerbuch</Titel>
    <Auflagen>
      <anyType xsi:type="Auflage">
        <Nr>1</Nr>
        <Jahr>2002</Jahr>
        <ISBn>3-86063-667-7</ISBn>
      </anyType>
    </Auflagen>
  </GeschriebeneBuecher>
</BAutor>
```

Abbildung 8 | Ergebnis der durch Meta-Attribute gesteuerten XML-Serialisierung (Ausschnitt).

Eine Klassendefinition wie in Listing 3 führt zu dem in Abbildung 8 ausschnitthaft dargestellten Ergebnis mit: *BAutor* als Wurzelelement, *Aktiv* als Attribut zu *BAutor,* Auslassen der Speicherung des Gehalts und dem Elementnamen *GeschriebeneBuecher* für die Objektmenge, die in der Klasse nur *Buecher* heißt.

Fazit

Warum es zwei so unterschiedliche Verfahrensweisen bei Binär- und SOAP-Serialisierer einerseits und dem XML-Serialisierer andererseits gibt, lässt sich nicht beantworten. Ein Microsoft-Mitarbeiter vermutete, dass wohl wieder einmal zwei miteinander nicht kommunizierende Entwicklungsteams am Werk waren. Tabelle 3 stellt die Unterschiede zusammenfassend gegenüber. |||||||