

VB.Net: XML-Dokumente verarbeiten mit XDocument

0. Einleitung

Da ich öfters schon von Problemen mit der XML-Datenverarbeitung gehört habe, entschloss ich mich, dieses Tutorial zu schreiben. Seit *.net Framework 3.5* gibt es den Namensraum `System.Xml.Linq`. Dieser stellt unter anderem die neuen Typen `XDocument`, `XElement` usw. zur Verfügung.

Dieses Tutorial bezieht sich auf VB2008 und VB2010.

1. XML Allgemein

XML steht für *Extensible Markup Language* und hat sich zum wichtigsten Standard zum Austausch von Daten über das Internet entwickelt. Sowohl *RSS-Feeds* als auch *Webservices* basieren auf XML. XML wird zunehmend auch als Datenspeicher für Programme benutzt, in welchem zum Beispiel Konfigurationseinstellungen gespeichert werden können.

Im Folgenden lernen Sie, wie man XML-Dokumente ausliest, sie verarbeitet und sie speichert. Der dafür vorgesehene Namensraum `System.Xml.Linq` ist bereits referenziert und importiert.

2. XML-Dateien einlesen

Mit `XDocument.Load` wird ein XML-Dokument aus der im Parameter angegebenen Datei gelesen:

```
Dim XDoc1 As XDocument = XDocument.Load("Datei.xml")
```

Durch so genannte *XML literals* kann eine Zuweisung allerdings auch so aussehen:

```
Dim XDoc2 As XDocument = <?xml version="1.0" encoding="utf-8"?>
    <config>
        <MainWindow IsMaximized="True">
            <Width>430</Width>
            <Height>300</Height>
        </MainWindow>
    </config>
```

Wie man hier deutlich sieht, ist Visual Basic flexibel und erlaubt eine direkte Zuweisung der XML-Codes ohne Anführungszeichen oder das VB-typische „_“-Zeichen. Der Code wird automatisch korrekt eingerückt und erhält auch *Syntax-Highlighting*.

Lädt man ein XML-Dokument von z.B. einem Webservice in einen *String*, funktioniert auch dieser Befehl:

```
Dim XDoc3 As XDocument = XDocument.Parse(StrWebService)
```

3. XML-Dateien verarbeiten

Lesezugriff

Als Beispiel nehme ich das folgende XML-Dokument, das sich hier im XDocument-Objekt namens XMLConfig befinden soll:

```
<?xml version="1.0" encoding="utf-8"?>
  <config>
    <MainWindow IsMaximized="True">
      <Width>430</Width>
      <Height>300</Height>
    </MainWindow>
  </config>
```

Zunächst wollen wir die Größe und den Zustand des Fensters auslesen und wieder in die entsprechenden Eigenschaften des Fensters schreiben:

```
If CBool(XDoc2.<config>.<MainWindow>.@IsMaximized) Then
  Me.WindowState = FormWindowState.Maximized
End If

Me.Width = CInt(XDoc2.<config>.<MainWindow>.<Width>.Value)
Me.Height = CInt(XDoc2.<config>.<MainWindow>.<Height>.Value)
```

Nun zur Erklärung:

In der ersten Zeile wird das Attribut `IsMaximized` des Elements `MainWindow` ausgelesen und in einen *booleschen Wert* konvertiert. Dies ist notwendig, da XML alle Daten nur als *String* speichern kann. Falls also dieser Wert wahr ist, soll das Fenster maximiert werden.

Danach bestimmen die XML-Tags `Width` und `Height` die gleichnamigen Eigenschaften für die Höhe und Breite des Fensters.

Schreibzugriff

Nehmen wir an, Sie wollen die aktuelle Größe und den Zustand des Fensters speichern. Das geht praktisch umgekehrt zum Lesen, wie es ja auch sein sollte: ☺

```
If Me.WindowState = FormWindowState.Maximized Then
  XDoc2.<config>.<MainWindow>.@IsMaximized = Boolean.TrueString
Else
  XDoc2.<config>.<MainWindow>.@IsMaximized = Boolean.FalseString
End If

XDoc2.<config>.<MainWindow>.<Width>.Value = CStr(Me.Width)
XDoc2.<config>.<MainWindow>.<Height>.Value = CStr(Me.Height)
```

`Boolean.TrueString` und `Boolean.FalseString` sind Konstanten für die Zeichenfolgen „True“ und „False“. Alternativ können auch `"True"` und `"False"` beziehungsweise `CStr(True)` und `CStr(False)` verwendet werden.

Dann speichert man das Ganze mit dem Befehl `Save`:

```
XDoc2.Save("Datei.xml")
```

Zur weiteren Bearbeitung des XDocuments als String benutzt man das altbekannte `ToString()`.