

# XML als universeel dataformaat

**D**it hoofdstuk gaat over de omgang met XML in het kader van programmeren met Visual Basic. Met Visual Basic 2008 kunt u nu ook comfortabeler werken met XML, zeker in vergelijking met voorgaande versies van Visual Basic. Na een korte inleiding in XML, dat al sinds het begin van de jaren negentig een officiële standaard is, duikt u de praktijk in. XML-gegevens zijn vandaag de dag alomtegenwoordig, in het bijzonder natuurlijk op internet, bijvoorbeeld met RSS-feeds, zodat er veel interessante toepassingen zijn voor het verwerken van XML-gegevens in Visual Basic. Het draait daarbij natuurlijk niet alleen om de vraag hoe XML-gegevens in het algemeen verwerkt worden, maar ook hoe dat samenspel met de voor de praktijk zo belangrijke technieken als XML Namespaces, XPath en XML Schema eruitziet.

## XML in tien minuten

Een XML-document bezit een uniforme en voor alles eenvoudige structuur. De vishaken doen weliswaar sterk denken aan HTML, maar daarmee heeft XML niets te maken. De naam tussen elk paar vishaken staat voor een XML-element. De naam zelf heeft geen betekenis en kunt u willekeurig declareren, waarbij uiteraard wel enkele algemene naamgevingsregels gelden. De belangrijkste regel is, dat XML *case sensitive* is, dat wil zeggen dat het gebruik van hoofdletters en kleine letters verschil maakt, voor de XML-parser zijn *Boek* en *boek* twee verschillende namen.

Een XML-document begint met een kopregel die altijd op dezelfde wijze is opgebouwd:

```
<?xml version="1.0"?>
```

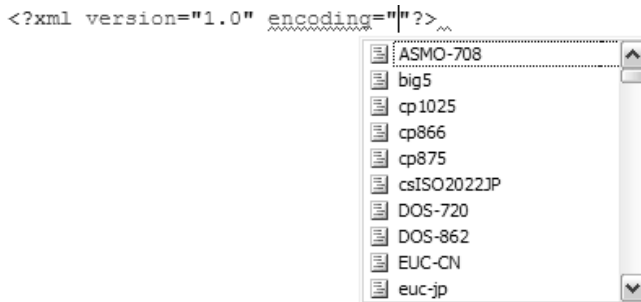
Het is gebruikelijk om met het attribuut `encoding` de tekenset voor de XML-inhoud vast te leggen:

```
<?xml version="1.0" encoding="utf-8"?>
```

of:

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

In het algemeen is `utf-8` een goede keuze, omdat deze tekenset ook diakritische tekens bevat. Maakt u zich geen zorgen als u niet de precieze benaming weet. Visual Basic heeft een goed uitgeruste XML-editor met IntelliSense, zodat u de juiste tekenset gewoon uit een lijst kunt kiezen. Een gewone teksteditor zoals Kladblok (Notepad) biedt u deze mogelijkheid niet.



**Afbeelding 15.1** De Visual Basic XML-editor toont u een lijst met tekensets waaruit u een keuze kunt maken.

## In drie stappen naar een XML-bestand

Visual Basic maakt het makkelijk om een XML-bestand te maken.

- 1 Kies in het menu **Project** voor **Add New Item**.
- 2 Kies de sjabloon **Text File**, geef het bestand een passende naam en wijzig de extensie van *.txt* in *.xml*.
- 3 Voer de XML-tekst in, waarbij de editor erop let of u de algemene regels voor een XML-document in acht neemt en waar nodig waarschuwingen en foutmeldingen geeft, terwijl IntelliSense u hulp biedt.

Na de kop volgt het *root element*, dat de top van de hiërarchie van elementen markeert:

```
<Boeken>
</Boeken>
```

Wat binnen het rootelement staat, maakt XML eigenlijk niets uit. Hoofdzakelijk is dat de basisregels wat betreft de algemene opbouw intact blijven. Een een dergelijke regel is dat elk openend element ook een afsluitend element kent:

```
<Boek> </Boek>
```

Binnen een element kunnen willekeurig veel andere elementen volgen:

```
<Boek>
  <Titel> </Titel>
  <Auteur>
    <Naam> </Naam>
    <Verjaardag> </Verjaardag>
  </Auteur>
</Boek>
```

Welke elementen en hoeveel elementen volgen, speelt geen rol. Ook is het niet verplicht dat bepaalde kindelementen fundamenteel aanwezig moeten zijn. In het volgende voorbeeld heeft het eerste `<Boek>`-element een `<Categorie>`-element, het tweede niet, maar dat heeft een `<Prijs>`-element:

```
<Boeken>
  <Boek>
    <Titel>Handboek Visual Basic 2008</Titel>
    <Categorie>Beginners</Categorie>
  </Boek>
```

```
<Boek>  
  <Titel>Leer jezelf PROFESSIONEEL... Visual Basic for Applications</Titel>  
  <Prijs>24,90</Prijs>  
</Boek>  
</Boeken>
```

Theoretisch kan een XML-document willekeurig diep genest zijn, zodat dit een theoretisch oneindig diepe boomstructuur oplevert. Ook wat betreft de grootte is er geen limiet. XML-bestanden met daarin bijvoorbeeld de gegevens van een productcatalogus, kunnen al snel vele megabytes of zelfs gigabytes groot zijn. Een nadeeltje van de opslag als tekst ten opzichte van binaire gegevens (zoals bitmaps) is, dat tekstbestanden duidelijk omvangrijker zijn dan wanneer dezelfde hoeveelheid gegevens binair wordt opgeslagen.

Het invoegen van regelomhalen en witregels in een XML-document doet u om de leesbaarheid te bevorderen, maar het is voor een XML-bestand niet noodzakelijk. Ze dragen niet bij tot de structuur van het document en ze worden net als spaties en tabs behandeld als zogenaamde *whitespaces*. Afhankelijk of de toepasselijke optie bij het inlezen van een XML-document is aangevinkt, ontstaan hierdoor extra knooppunten die eventueel bij de verwerking in acht genomen moeten worden. Normaal gesproken spelen ze echter geen rol, zodat het geen nadeel is om ze in te voegen. In de voorbeelden staat elk element op een eigen regel om de leesbaarheid te verhogen, maar het is niet dwingend voorgeschreven.

Tussen het openende en het bijbehorende sluitende element staat in het algemeen tekst:

```
<Titel>Handboek Visual Basic 2008</Titel>
```

Dit is de waarde van het XML-element en dat is waarop het bij het verwerken van een XML-document in de regel aankomt. Omdat er in XML zelf geen mogelijkheid bestaat om een datatype aan te geven, komt *XML Schema* in het spel, dat deze mogelijkheid wel biedt. XML Schema komt later uitgebreid aan bod.

Blijft het gebied tussen het openende en afsluitende element leeg, dan kan het afsluitende element ook vervallen, in dat geval sluit u het element af met een `</>`:

```
<Titel/>
```

Teksten kunnen niet alleen tussen twee elementen staan, maar ook binnen het startelement in de vorm van zogenoemde attributen. Een attribuut is een eveneens willekeurige naam, waarop, gescheiden door een isgelijktteken, een waarde volgt die steeds tussen aanhalingstekens moet staan:

```
<Boeken>
  <Boek Nr="1000">Handboek Visual Basic 2008</Boek>
  <Boek Nr="2000">Handboek Visual C# 2008</Boek>
</Boeken>
```

In dit geval bezit het element <Boek> een attribuut met de naam Nr. Als de waarden van Nr uniek zijn, dan kan het attribuut de rol van een key spelen, wat belangrijk is wanneer een XML-document als alternatief voor een database fungeert. Het is echter net zo goed mogelijk om Nr niet als attribuut, maar als een element in te bouwen:

```
<Boeken>
  <Boek>
    <Nr>1000</Nr>
    <Titel>Handboek Visual Basic 2008</Titel>
  </Boek>
  <Boek>
    <Nr>2000</Nr>
    <Titel>Handboek Visual C# 2008</Titel>
  </Boek>
</Boeken>
```

Meer regels voor het bouwen van een XML-document zijn er niet.



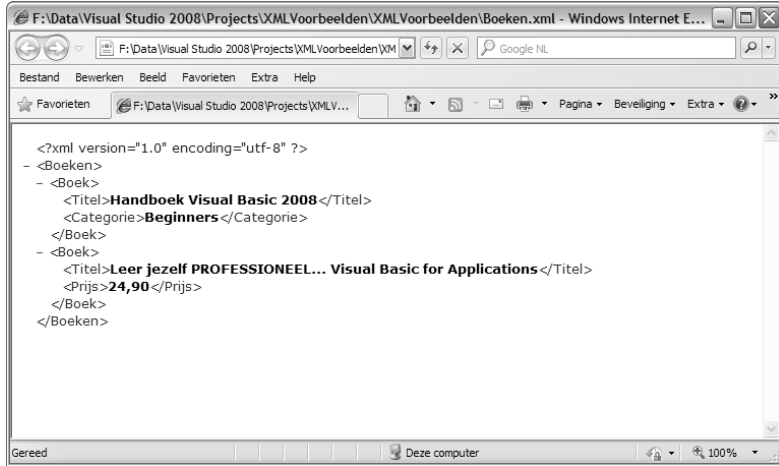
### Volgens de regels

XML-documenten die deze eenvoudige basisregels volgen, worden als *well formed* aangeduid.

Hoewel de bovenstaande oefening niet moeilijk is, komt het zelden voor dat een programmeur XML-documenten op deze wijze maakt. In de regel ontstaan XML-documenten indirect, namelijk als resultaat van een export.

### XML-bestanden in Internet Explorer

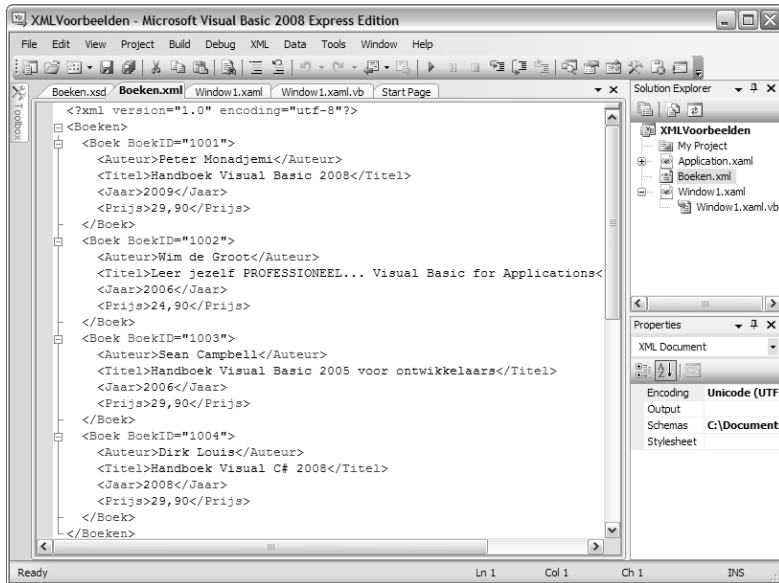
Internet Explorer toont (al sinds versie 5.0) ook XML-bestanden. Dat is een goede gelegenheid om de opbouw van een XML-bestand te testen. Als het XML-document fouten bevat, worden deze op de juiste plaats gemarkeerd. Laad het document in Internet Explorer, dat is de snelste en makkelijkste manier om vast te stellen of een XML-document *well formed* is.



Afbeelding 15.2 Internet Explorer toont een XML-document.

## Visual Basic en XML

Visual Basic heeft een ingebouwde XML-editor die weliswaar niet vergelijkbaar is met een professioneel XML-tool als Oxygen, maar die het bewerken van XML wel een stuk gemakkelijker maakt:



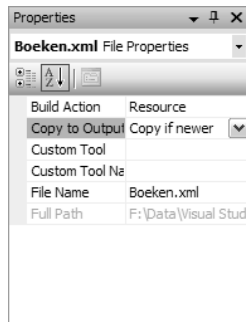
Afbeelding 15.3 Visual Basic heeft een ingebouwde XML-editor.

- Voegt automatisch bij een openend element het sluitende element toe.
- Van een XML-bestand kan een XML Schema worden afgeleid.
- Als voor een XML-bestand een XML Schema bestaat, krijgt u bij de invoer van elementen een keuzelijst te zien.
- Er zijn verschillende invoerhulpen en enkele sneltoetsen beschikbaar die de invoer van XML-gegevens vereenvoudigen, zie hiervoor de hulp-functie.

Alle voorbeelden in dit hoofdstuk hebben betrekking op een XML-bestand met de naam *Boeken.xml*. In dit bestand staat een *rootelement* Boeken en een reeks Boek-elementen. Elk Boek-element heeft de kindelementen Auteur, Titel, Jaar en Prijs en het attribuut BoekID.

## XML-bestanden zonder padopgave aanspreken

U kunt een tot het project behorend XML-bestand zonder padopgave in het programma aanspreken. Alles wat u daarvoor hoeft te doen, is ervoor zorgen dat het tijdens de compilatie in de uitvoermap terechtkomt. Stel hiervoor de eigenschap Copy to Output Directory van het XML-bestand in op Copy if newer. De eigenschap Build Action mag eventueel op None staan.



**Afbeelding 15.4** De eigenschappen van het XML-bestand bepalen of het bestand tijdens de compilatie in de uitvoermap terechtkomt.

## XML-gegevens verwerken

Het .NET Framework – en daarmee ook Visual Basic – kent verschillende mogelijkheden om XML-gegevens in een programma binnen te halen om ze te verwerken:

- Met de klasse `XmlDocument` in de namespace `System.Xml`.
- Met de klasse `XmlTextReader` in de namespace `System.Xml`.

- Met de klasse `XDocument` in de namespace `System.Xml.Linq`.
- Met de algemene klasse `DataSet`, die XML-gegevens direct kan lezen.
- Met de klasse `XmlSerializer` – in dit geval wordt de inhoud van een XML-bestand, voor zover het een passende opbouw heeft, direct naar objecten geconverteerd.

Natuurlijk kunt u in theorie ook een XML-bestand net als ieder ander tekstbestand met klassen als `FileStream` of `StreamReader` inlezen en de inhoud direct verwerken, maar in de praktijk is dat veel te omslachtig. Van de opgesomde varianten is het gebruik van de klasse `XDocument` in het algemeen het eenvoudigst en daarom staat deze werkwijze in dit hoofdstuk op de voorgrond. Er zijn echter twee nadelen die vooral de meer ervaren programmeurs interesseren:

- 1 De klasse `XDocument` is beschikbaar sinds .NET Framework 3.5.
- 2 Het samenspel met de andere XML-namespaces zoals `System.Xml.Schema` en `System.Xml.Xslt` gebeurt via uitbreidingsmethoden, zodat de prestaties in vergelijking met de klasse `XmlDocument` wat slechter zijn. Dit aspect speelt echter zelden een rol.

Als het gaat om de gegevens als geheel en minder om de toegang tot de afzonderlijke elementen, dan is het gebruik van een `DataSet` eenvoudiger, zoals het volgende voorbeeld duidelijk maakt.

```
Dim Ds As New DataSet
Ds.ReadXml("Boeken.xml")
```

Aansluitend staat de XML-inhoud, al naar gelang de structuur, via een of meer `DataTables` ter beschikking en kan bijvoorbeeld aan een `DataGridView` worden gekoppeld:

```
DataGridView1.DataSource = Ds.Tables(0)
```

Deze variant gaat ervan uit dat de XML-inhoud in een tabelstructuur kan worden omgezet. Als een XML Schema bestaat, kan dat vooraf in de `DataSet` worden geladen:

```
Ds.ReadXmlSchema("Boeken.xsd")
Ds.ReadXml("Boeken.xml")
```

Als het XML-bestand niet overeenkomt met het XML Schema, dan heeft het laden een exception tot gevolg.



U hebt overigens geen DataSet nodig om de inhoud van een XML-bestand direct te laden. Met LINQ gaat het net zo compact:

```
DataGridView1.DataSource = (From B in XDoc...<Boek> Select New With
    {.Title = B...<Title>.Value,
     .Auteur = B...<Auteur>.Value,
     .Jaar = B...<Jaar>.Value,
     .Prijs = B...<Prijs>.Value,
     .BoekNr = B.@BoekID}).ToList
```

Voor LINQ maakt het geen verschil welke soort objecten de collection bevat. In dit geval zijn dat objecten van de klasse XElement, waarover later meer. De wat vreemd aandoende syntaxis zoals B...<Title> is gewoon een vereenvoudiging die de Visual Basic-compiler toestaat. De rest is normale LINQ. Aan het eind ontstaat een getypeerde collectie van een anoniem type dat aan de eigenschap DataSource van de DataGridView wordt toegewezen.

Het is fascinerend hoe u dankzij de met Visual Basic 2008 ingevoerde compileruitbreidingen een XML-query kunt vereenvoudigen. Een conventionele query, waarbij elk element van de collection via de eigenschap Element van Boek wordt aangesproken, ziet er zo uit:

```
Dim XDoc As XDocument = XDocument.Load("Boeken.xml")
DataGridView1.DataSource = (From B in XDoc...<Boek> Select New With
    {.Title = B.Element("Title").Value,
     .Auteur = B.Element("Auteur").Value,
     .Jaar = B.Element("Jaar").Value,
     .Prijs = B.Element("Prijs").Value,
     .BoekNr = B.Attribute("BoekID").Value}).ToList
```

Deze variant eist echter dat bij elk element alle kindelementen en het attribuut aanwezig zijn, omdat dit type query geen Null als waarde toestaat.

De volgende variant is een typische hack, die ervoor zorgt dat er geen fout optreedt als het attribuut BoekID ontbreekt. Ontbreekt het attribuut BoekID, dan wordt een defaultwaarde toegekend. Dit is een volkomen legale, doch omvangrijke LINQ-query:

```
Dim AlleBoeken = (From B in XDoc...<Boek>
    BoekID = (From Boe In B.Attributes
        Where Boe.Name = "BoekID" Select Bu.Value).SingleOrDefault
    Select New With
```

## Hoofdstuk 15 – XML als universeel dataformaat

```
{.Titel = B...<Titel>.Value,  
.Auteur = B...<Auteur>.Value,  
.Jaar = B...<Jaar>.Value,  
.Prijs = B...<Prijs>.Value,  
.BoekNr = Iff(BoekID Is Nothing, "Geen ID", BoekID)}.ToList
```

Er wordt voor alle Boek-elementen een nieuw object aangelegd, waarbij een subquery test of het attribuut BoekID een waarde heeft en de defaultwaarde Geen ID toekent als dat niet het geval is.

Maar geen programmeur is verplicht om XML-gegevens met een LINQ-query binnen te halen. Het gaat natuurlijk ook gewoon met een lus:

```
Dim AlleBoekenTemp As New List(Of XElement)  
For Each Boek As XElement In  
  XDoc.Elements("Boeken").Elements("Boek")  
  ' Bestaat het attribute BoekID?  
  If Boek.Attributes("BoekID").SingleOrDefault Is Nothing Then  
    Dim BoekIDAttr As New XAttribute("BoekID", "9999")  
    Boek.Add(BoekIDAttr)  
  End If  
  AlleBoekenTemp.Add(Boek)  
Next
```

De lus For Each doorloopt alle elementen van Boeken als XElement-objecten en test of een element het attribuut BoekID bezit. Is dat niet het geval, dan wordt het als XAttribute-object aan Boek toegevoegd.

Aansluitend kan de collection die in alle XElement-objecten een attribuut BoekID bezit met LINQ worden samengesteld:

```
Dim AlleBoeken = (From B in AlleBoekenTemp Select B.@BoekID,  
  B...<Titel>.Value, B...<Auteur>.Value,  
  B...<Jaar>.Value, B...<Prijs>.Value).ToList
```

De reden waarom u niet de collection AlleBoekenTemp direct aan het stuur-element DataGridView koppelt, is dat een XElement-object talrijke andere eigenschappen bezit die bij een directe koppeling ook in de DataGridView verschijnen.

## De klassen XDocument, XElement en XName

Met XDocument, XElement en XName biedt de namespace System.Xml.Linq drie klassen waarmee u XML-gegevens makkelijk en voor alles programmeur-vriendelijk kunt aanspreken.

### XML-Documenten laden

Met XDocument, dat voor een XML-document als geheel staat, kunt u XML-inhoud zowel lezen als schrijven. Het is interessant dat u daartoe geen instantiatie van de klasse XDocument nodig hebt, omdat de methode Load een Shared member is. De volgende opdracht laadt het XML-bestand *Boeken.xml*:

```
Dim XDoc As XDocument = XDocument.Load("Boeken.xml")
```

Tijdens het inlezen – dat men in dit verband ook wel *parsen* noemt – wordt intern een boomstructuur opgebouwd, die u via de variabele XDoc op verschillende manieren kunt aanspreken.

Wilt u de witregels ook in de boomstructuur opnemen – iets dat normaal niet nodig is – dan dient u dit bij het laden op te geven:

```
Dim XDoc As XDocument = XDocument.Load("Boeken.xml", LoadOptions.PreserveWhitespace)
```

Hierdoor stijgt het aantal knooppunten, maar niet het aantal XElement-objekten. Daarom maakt men onderscheid tussen elementen en knooppunten. Een element staat alleen voor een XML-Element en heeft alleen een eigenschap Value, een knooppunt kan ook voor andere XML-elementen staan, bijvoorbeeld een XML-commentaar.

De klasse XDocument is niet vereist, ook de klasse XElement heeft een methode Load, waarmee u XML-inhoud direct in een XElement-object kunt laden.

### Toegang tot afzonderlijke elementen

Visual Basic 2008 biedt in samenhang met de klasse XElement een verbaazingwekkende verscheidenheid aan manieren om afzonderlijke elementen aan te spreken. Naast de formele schrijfwijze, waarbij u de kindelementen via de eigenschap Element van het XElement-object aanspreekt, is er ook de al eerder geïntroduceerde verkorte schrijfwijze, waarbij de elementrelatie door drie punten wordt gesymboliseerd.

Als derde variant kunt u de elementen over hun relatie met andere knooppunten in de boomstructuur aanspreken. Met de eigenschap `Descendants` krijgt u alle ‘afstammelingen’ van een element (lager in de hiërarchie), met de eigenschap `Ancestors` overeenkomstig de ‘voorvaders’ van een element (hoger in de hiërarchie). Dan is er nog een vierde variant om in plaats van elementen de knooppunten aan te spreken met de eigenschap `Nodes` van een `XElement`-object. Dit biedt een voordeel wanneer u aanwezig commentaar of andere speciale elementen wilt aanspreken, maar dit komt slechts zelden voor en komt daarom in dit hoofdstuk verder niet meer aan bod. Als vijfde alternatief kunt u ook elk element via een index aanspreken.

Uitgangspunt voor alle voorbeelden is het bestand *Boeken.xml*, waarvan de opbouw in afbeelding 15.3 te zien is.

De volgende opdracht retourneert de waarde van het eerste element `Boek` als string die alle kindelementen samenvat:

```
Dim EersteBoek = XDoc.Elements("Boeken").Elements("Boek").Value
```

U krijgt het kindelement `<Title>` met de opdracht:

```
Dim EersteTitel = XDoc.Elements("Boeken").Elements("Boek").Elements("Title").Value
```

De hieraan ten grondslag liggende logica is eenvoudig. Met de eigenschap `Elements` selecteert u een element, waarvan u met de eigenschap `Elements` een kindelement selecteert enzovoort. Dat op `Elements` steeds een naam volgt, is bij nadere beschouwing vreemd, omdat de methode `Elements` geen string, maar een parameter van het type `XName` verwacht. Dit is blijkbaar een verdere verkorting die de compiler toestaat wanneer uit het `String`-element een `XName`-element wordt gemaakt.

Voor het geval dat een XML-document namespaces declareert, gaat het niet meer zo eenvoudig. Aangenomen dat het XML-document als volgt gedeclareerd is:

```
<boe:Boeken xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:boe="urn:XMLVoorbeeld"
  xsi:schemaLocation="urn:XMLVoorbeeld Boeken.xsd">
  <Boek>
    <Title>Het grote VB boek</Title>
```

In dit document is de namespace `boe` voor het element `Boeken` gebruikt (verderop in dit hoofdstuk meer over XML-namespaces).

De opdracht:

```
Dim EersteTitel = XDoc.Elements("Boeken").Elements("Boek").Elements("Titel").Value
```

retourneert nu `Nothing`, omdat voor het element `Boeken` ook de namespace moet worden aangegeven:

```
Dim EersteTitel = XDoc.Elements(XName.Get("Boeken", "urn:XMLVoorbeeld")).Elements("Boek").Elements("Titel").Value
```

Nu is het element `XName` expliciet aangegeven. Maar ook hier is weer een vereenvoudiging in het spel, omdat u voor de namespace een `String`- en geen `XNamespace`-waarde (die de hele namespace representeert) kunt ingeven. Naast deze formele schrijfwijze biedt Visual Basic ook een verkorte schrijfwijze:

```
Dim EersteBoek As XElement = XDoc...<Boek>(0)
```

Of als het om de titel gaat:

```
Dim EersteBoek = XDoc...<Boek>(0)...<Titel>.Value
```

En het kan zelfs nog een beetje korter:

```
Dim EersteBoek = XDoc...<Titel>(0).Value
```



### Index gebruiken

Officieel staat `Elements` het aanspreken van een afzonderlijk element via een index niet toe. Probeert u het toch, dan gebruikt Visual Basic de uitbreidingsmethode `ElementAtOrDefault`, waarmee een element ook via een getalsindex valt aan te spreken. En niet alleen dat, maar behoort er bij die index geen waarde, dan wordt automatisch de defaultwaarde geretourneerd. Zo blijft programmeren leuk.

Het is bovendien niet noodzakelijk steeds de klasse `XDocument` te gebruiken, een `XElement`-knooppunt kunt u ook direct laden. Op deze manier:

```
Dim EersteBoek As XElement = XDocument.Load("Boeken.xml")...<Boek>(0)...<Titel>(0)
```

Of zo:

```
Dim EersteBoek = XElement.Load("Boeken.xml")...<Boek>(0)...<Titel>(0)
```

De klassen in de namespace `System.Xml.Linq` zijn niet alleen in dit opzicht erg flexibel.

Het behoort tot de fascinerende mogelijkheden van de klasse `XElement`, dat u zich niet aan de hiërarchie hoeft te houden. De opdracht

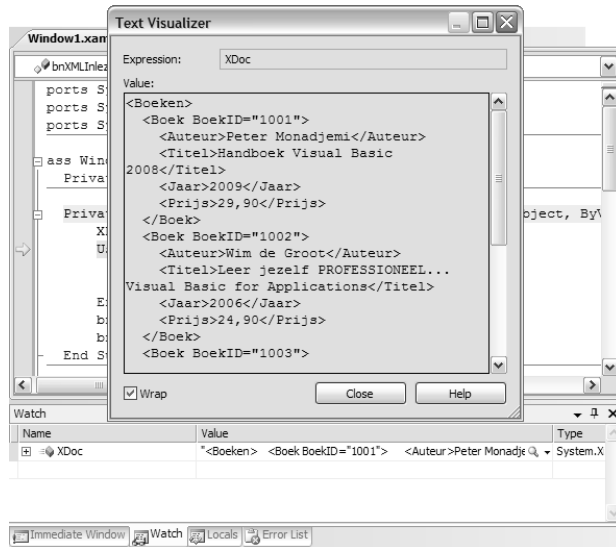
```
Dim AlleTitels = XDoc.Descendants("Titel")
```

retourneert de kindelementen `Titel` als `XElement`-objecten, hoewel dit kind-elementen zijn van de elementen `Boeken` en `Boek` die in de hiërarchie op een hoger niveau staan. Als alternatief werkt ook:

```
Dim AlleTitels = XDoc...<Titel>
```

Attributen spreekt u aan met `.@`. De expressie `XDoc...<Boek>.@<BoekID>` retourneert het attribuut `BoekID`, voor zover het bestaat, van het eerste element `Boek`.

Een andere aangename eigenschap van `XElement` is, dat u geen onderscheid hoeft te maken tussen de XML-tekst van een knooppunt en de waarde die tussen de XML-elementen staat. De eigenschap `Value` van een `XElement`-object retourneert de tekst tussen de XML-elementen. Omvat het bereik andere XML-elementen, dan wordt van deze kindelementen ook alleen de tekstinhoud meegenomen. Wilt u alle XML inlezen, dan gebruikt u hiervoor de methode `ToString`.



Afbeelding 15.5 De Text Visualizer toont de inhoud van de variabele.



### XML-expressies evalueren

Tijdens een programmaonderbreking kunt u XML-expressies, die de korte schrijfwijze gebruiken, schijnbaar niet altijd evalueren. Maar dat is slechts schijn. Markeer de hele expressie en gebruik de sneltoets Shift+F9. U kunt het resultaat van de expressie in het venster Immediate verder onderzoeken. Wanneer u een XML-waarde bekijkt tijdens een programmaonderbreking, dan ziet u de XML-code vaak onvolledig in het vak **Quick Info**. Wilt u dit helemaal zien, klik dan op het vergrootglas in het venster om Text Visualizer op te roepen.

Tabel 15.1 – De belangrijkste members van de klasse XElement.

Member	Betekenis
...	Zorgt voor het aanspreken van het kindelement tussen de vishaken.
.@	Zorgt voor het aanspreken van een attribuut. Als een schema voorhanden is, worden de attributen in een keuzelijst aangeboden.
Add	Methode. Voegt een element of attribuut toe aan het actuele element.
Ancestors	Methode. Retourneert een collection van de ‘voorvaders’ van het element (bovenliggend in de hiërarchie).
Attributes	Methode. Retourneert een collection van attributen van het element.
CreateWriter	Methode. Maakt een <code>XmlWriter</code> voor het toevoegen van knooppunten.
Descendants	Methode. Retourneert een collection van afstammelingen van het element in volgorde van het document.
Element	Methode. Retourneert het eerste kindelement (in volgorde van het document) met de opgegeven naam.
Elements	Methode. Retourneert een collection van kindelementen van het element of document in volgorde van het document.
HasAttributes	Property. True als het element minimaal één attribuut heeft, anders False.
HasElements	Property. True als het element minimaal één kindelement heeft, anders False.
Load	Methode. Laadt een element uit een bestand.
Name	Property. Retourneert de naam van het element.
Nodes	Methode. Retourneert een collection van kindknooppunten van dit element of knooppunt in volgorde van het document.
Parent	Property. Retourneert het ouderelement van het element (direct bovenliggend in de hiërarchie).
Remove	Methode. Verwijdert het knooppunt van het ouderknooppunt.
ReplaceWith	Methode. Vervangt het knooppunt met de opgegeven inhoud.
Save	Methode. Schrijft het element naar een bestand.
Value	Property. Retourneert de volledige tekstinhoud van het element.

## XML-gegevens met LINQ uitlezen

De algemene querytaal LINQ (zie hoofdstuk 7) is ook geschikt voor het aanspreken van afzonderlijke XML-elementen en het filteren van XML-knoop-punten met betrekking tot de gezamenlijke boomstructuur. LINQ is daarmee indirect een alternatief voor de standaard XML-methoden XPath en XQuery (de laatste wordt overigens ook niet in .NET 3.5 ondersteund).

De volgende opdracht spreekt alle <Title>-elementen aan:

```
Dim AlleTitles = From T in XDoc...<Title> Select T
```

De voordelen van LINQ komen pas goed tot hun recht wanneer u filter-criteria gebruikt. De volgende opdracht retourneert slechts die boeken als <Book>-element die uit het jaar 2008 stammen.

```
Dim Boeken2008 = From B In XDoc...<Book> Where B...<Jaar>.Value = 2008 Select B
For Each B in Boeken2008
    Console.WriteLine(B...<Title>.Value)
Next
```

In principe staan natuurlijk ook in een LINQ-query alle members van de klasse XElement ter beschikking.

De volgende opdracht retourneert alleen die boeken als <Book>-element die een attribuut (BoekID) bezitten:

```
Dim BoekenMetID = From B In XDoc...<Book> Where B.Attributes.SingleOrDefault <"> Select B
For Each B in BoekenMetID
    Console.WriteLine(B.@BoekID)
Next
```

### Samenspel van LINQ met XML Schema

Hoe flexibel Visual Basic XML tegenwoordig verwerkt, maakt het volgende voorbeeld duidelijk. In hoofdstuk 13 hebt u het bestand *KoersGegevens.xml* gebruikt als voorbeeld van het aanspreken van valutakoersen via internet. In het volgende voorbeeld gebruikt u dit XML-bestand in een LINQ-query.

Hiervoor hebt u als eerste een schema nodig, zodat u binnen de LINQ-query de afzonderlijke XML-elementen via IntelliSense krijgt aangeboden en daarnaast hebt u de zekerheid dat de query ook kan worden uitgevoerd. Het onderwerp XML Schema komt later nog aan bod.



Voor nu volstaat het dat u weet dat een XML Schema de structuur van een XML-bestand beschrijft en dat IntelliSense het schema gebruikt om passende invoerhulp te bieden bij het bewerken van XML en het schrijven van een LINQ-query.

Maar hoe komt u aan een dergelijk schema? Heel eenvoudig, Visual Basic leidt dat af uit het bestand *KoersGegevens.xml* met behulp van de sjabloon *XML to Schema*.



### Nog geen SP1?

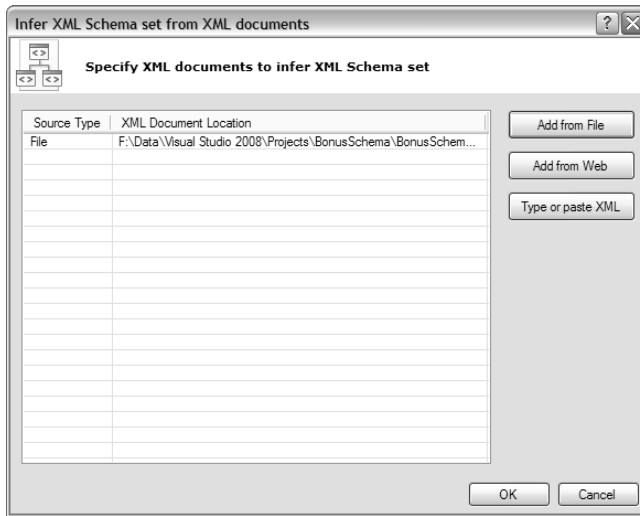
De sjabloon XML to Schema is pas met ingang van Visual Basic 2008 SP1 beschikbaar. Gebruikt u nog geen SP1, dan kunt u de sjabloon ook in de vorm van *XML to Schema Inference Wizard voor Visual Studio 2008* downloaden op de website van Microsoft: [www.microsoft.com/Downloads/details.aspx?familyid=9AC5A653-563A-4204-A4EB-DDDCAE80B244&displaylang=en](http://www.microsoft.com/Downloads/details.aspx?familyid=9AC5A653-563A-4204-A4EB-DDDCAE80B244&displaylang=en). En mocht u geen zin hebben om deze lange URL in te kloppen, zoek dan op microsoft.com naar *XML to Schema Inference Wizard*.

U hoeft overigens het bestand niet op uw eigen computer te hebben. U kunt een schema afleiden van een XML-bestand op internet als u beschikt over de juiste URL. En dat doet u als volgt:

- 1 Maak een nieuw Windows Forms-project, waarbij het projecttype eigenlijk geen rol speelt (het mag dus ook een WPF-toepassing zijn). De naam van het project mag u zelf kiezen.
- 2 Voeg een nieuw item toe aan het project: **Project, Add New Item** en kies de sjabloon **XML to Schema**. Geef het de naam *KoersGegevens.xsd* en klik op **Add**.
- 3 Kies in het dialoogvenster voor **Add from Web**. Stel dat dit bestand te vinden is op <http://www.valutakoersen.nl/koersgegevens.xml>, geef dan dit HTTP-adres op en klik op **OK**. Als alles goed is, is het schemabestand nu een onderdeel van het project. U kunt overigens ook een bestaand bestand gebruiken of rechtstreeks XML-gegevens invoeren. Voor het schema maakt het niet uit.
- 4 Nu kunt u de LINQ-query bouwen. Voeg op het formulier een knop toe en voeg in de `Click`-eventprocedure de volgende opdracht in:

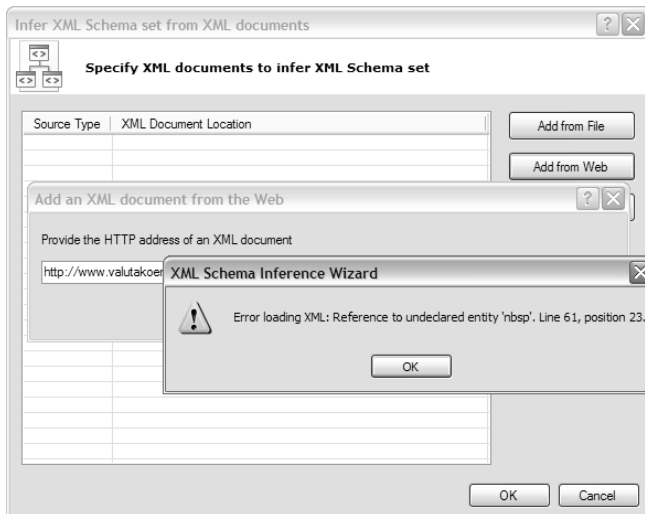
```
Dim XDoc = XDocument.Load("http://www.valutakoersen.nl/koersgegevens.xml")
```

- 5 Stop, even een stapje terug. Aangezien de hier bovengenoemde URL geen echt internetadres is, maar slechts als voorbeeld dient, krijgt u bij stap drie al een foutmelding als u deze URL invoert.



**Afbeelding 15.6** Hier geeft u de bron van de XML-gegevens op, dat kan een bestand zijn op uw computer of op internet, maar u kunt ook de XML-gegevens hier invoeren.

Maar gebruikt u een bestand rechtstreeks van internet, dan kunt u de bovenstaande opdracht gebruiken (maar dan wel met een werkend adres en uiteraard een werkende internetverbinding).



**Afbeelding 15.7** Als het internetadres niet klopt of het bestand een andere naam heeft, krijgt u een foutmelding in plaats van een schema.

- 6 Omwille van het voorbeeld gebruikt u voor deze oefening echter het bestand *KoersGegevens.xml*, dat u bij de voorbeeldbestanden vindt. Voeg het XML-bestand toe aan het project met **Project, Add Existing Item** – zorg dat u alle bestanden te zien krijgt – selecteer het bestand dat u wilt toevoegen en klik op **Add**. Selecteer het XML-bestand in de **Solution Explorer** en ga naar het venster Properties, wijzig de eigenschap Copy to Output Directory in Copy if newer.
- 7 Kies in het menu **Project, Add New Item** en selecteer **XML to Schema**, geef het bestand een passende naam en selecteer in het volgende dialoogvenster **Add from File**, selecteer het bestand *KoersGegevens.xml* en klik op **OK**. Voeg dan een knop toe en voeg in de Click-eventprocedure van de knop de volgende opdrachten toe:

```
Dim XDoc = XDocument.Load("KoersGegevens.xml")
Dim USDKoers = (From D In XDoc...<Valuta> Where D.@ISO = "USD" Select D...<Koers>.Value).SingleOrDefault
MessageBox.Show(String.Format("De koers van de USD: {0}", USDKoers))
```

De eerste opdracht laadt het bestand in een XDocument-object, onafhankelijk of dat bestand op internet staat of op uw eigen computer. De volgende opdracht is een LINQ-query die het <Valuta>-element lokaliseert waarvan het attribuut ISO de waarde USD heeft en retourneert de koers. IntelliSense biedt u een keuzelijst wanneer u begint met het invoeren van de punten en bij de eerste vishaak krijgt u een keuzelijst met de XML-elementen. En dat allemaal dankzij het XML Schema.



**Afbeelding 15.8** Dankzij het afgeleide XML Schema biedt IntelliSense u de XML-elementen aan in een keuzelijst. Dat vergemakkelijkt het maken van een LINQ-query.

## Knooppunten vinden met XPath

XPath is een standaard binnen de XML-familie, waarmee u knooppunten in een XML-boomstructuur aan de hand van hun *pad* kunt lokaliseren. Omdat er voor het lokaliseren van elementen met de klassen XDocument en XElement eenvoudige alternatieven zijn, speelt XPath in Visual Basic 2008 niet meer de belangrijke rol als de oudere klassen XmlDocument en XmlNode.

U kunt XPath-expressies gebruiken in samenhang met XElement-elementen. Hiervoor gebruikt u de uitbreidingsmethode XPathSelectElement, die echter pas beschikbaar is als u de namespace System.Xml.XPath importeert.

De volgende opdracht retourneert alle Boek-elementen waarvan de Prijs hoger is dan 25 euro:

```
Dim DureBoeken = XDoc.XPathSelectElements("//Boek[Prijs>25]")
```

Omdat ook .NET 3.5 alleen XPath 1.0 ondersteunt en deze ondertussen bijna verouderde standaard slechts een minimum aan vergelijkingsoperatoren biedt, is LINQ gewoonlijk de betere keuze:

```
Dim DureBoeken = From B in XDoc...<Boek> Where B...<Prijs>.Value > 25 Select B
```

Het kan gebeuren dat deze query alle Boek-elementen retourneert, onafhankelijk van hun Prijs. De oorzaak van dit raadsel ligt in het decimaalteken dat u hebt gebruikt en de landinstelling van het besturingssysteem. In Nederland is het scheidingsteken voor decimalen de komma, maar in andere landen kan dat een punt zijn. Hebt u een bestand waarin het verkeerde teken is gebruikt als decimaalteken, dan wordt de prijs niet goed geïnterpreteerd. Dit lost u op met de klasse XmlConvert in de namespace System.Xml met de methode ToSingle die u direct kunt inbouwen in de LINQ-query:

```
Dim DureBoeken = From B in XDoc...<Boek> Where XmlConvert.ToSingle(B...<Prijs>.Value) > 25 Select B
```

Dit zijn kleinigheden die u gemakkelijk langere tijd over het hoofd ziet en die tot onontdekte fouten leiden.

## XML valideren met XML Schema

Zonder een XML Schema zijn XML-gegevens meestal niet al te veel waard, omdat aan de hand van een XML Schema eenvoudig is vast te stellen of een XML-document de juiste structuur heeft. Met andere woorden, of alle verwachte elementen aanwezig zijn die voor verdere bewerking noodzakelijk zijn, of deze elementen van het verwachte datatype zijn en bijvoorbeeld in de verwachte volgorde voorhanden zijn. Alleen een schema kan de geldigheid van een XML-document bevestigen. Deze gang van zaken noemt men validering.

Een schema is vergelijkbaar met een taalreferentie van een programmeertaal. Elk tekstbestand komt als invoer voor de Visual Basic-compiler in aamerking, maar de compiler kan alleen een tekstbestand met de juiste opbouw foutvrij compileren. Of het resultaat foutvrij uitgevoerd kan worden, is een andere vraag. Ook een XML Schema garandeert niet dat een XML-document de juiste inhoud heeft. Bij de validatie van een XML-document met een XML Schema gaat het uitsluitend om de structuur van het XML-document.

Een XML Schema is geheel in XML opgebouwd. Voor XML Schema's bestaat een standaard, die XML Schema 1.0 heet. Op het eerste gezicht lijkt een XML Schema rijkelijk complex, wat het echter niet is (de massa XML-markeeringen en XML-namespaces is in het begin misschien wat overweldigend). Een voorbeeldje verduidelijkt dit wellicht.

Gegeven is een heel eenvoudige XML-structuur die als volgt is opgebouwd:

```
<Boeken>
  <Boek BoekNr="1234">
    <Titel>Alles duidelijk</Titel>
    <Auteur>Alles duidelijk</Auteur>
    <AantalBladzijden>1234</AantalBladzijden>
    <Prijs>19,95</Prijs>
  </Boek>
</Boeken>
```

Een daarbij passend XML Schema zou er als volgt uit kunnen zien (er zijn meerdere varianten mogelijk):

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Boeken">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Boek">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Titel" type="xs:string" />
              <xs:element name="Auteur" type="xs:string" />
              <xs:element name="AantalBladzijden" type="xs:unsignedShort" />
              <xs:element name="Prijs" type="xs:string" />
            </xs:sequence>
            <xs:attribute name="BoekNr" type="xs:unsignedShort" use="required" />
          </xs:complexType>
        </xs:element>
```

```
</xs:sequence>  
</xs:complexType>  
</xs:element>  
</xs:schema>
```

Naast de gebruikelijke XML-kop heeft het XML Schema natuurlijk een root-element met de naam `schema`. De volledige naam is echter `xs:schema`, omdat de namespace `xs` aan de naam voorafgaat. Deze namespace `xs` wordt in het rootelement gedeclareerd:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

De XML-namespaces komen verderop aan bod.

Alle schema-elementen worden voorafgegaan door de namespace (die overigens niet per se `xs` hoeft te heten). Als volgende wordt het element `Boeken` gedefinieerd, dit is een complex element omdat het kinderelementen heeft. Dat er slechts een element `Boek` aanwezig is, doet niet ter zake. Ook het element `Boek` is een complex element dat de kinderelementen `Title`, `Auteur`, `AantalBladzijden` en `Prijs` bezit en een attribuut `BoekNr`. Elk van deze kinderelementen (en het attribuut) hebben een datatype toegewezen gekregen. Deze datatypes zijn echter alleen aanbevelingen, die resulteren uit de afleiding van het XML Schema uit de XML-tekst. U kunt de datatypes wijzigen als dat nodig is, maar het gaat erom dat het duidelijk is dat bij een XML Schema elk element een datatype krijgt toegewezen, zelfs als die vaak eenvoudig van het type `string` is. XML Schema gebruikt een eigen set datatypes die met de datatypes van Visual Basic slechts voorwaardelijk compatibel zijn (zo is er een groot verschil bij de datum).

De huidige vorm van het XML Schema geeft aan dat de kinderelementen `Title`, `Auteur`, `AantalBladzijden` en `Prijs` exact in deze volgorde moeten voorkomen (met `<xs:sequence>` `</xs:sequence>`), maar daarvoor bestaat in het algemeen geen reden, dus kunt u dat maar beter veranderen. U kunt het XML Schema achteraf altijd aan uw wensen aanpassen, hoewel dat meestal niet nodig is.

De volgende vraag is, hoe vinden XML-gegevens en het XML Schema elkaar? Met andere woorden, hoe bereikt u dat Visual Basic bij de invoer van XML-gegevens een aanwezig XML Schema gebruikt? Dat is bij Visual Basic verbaazingwekkend eenvoudig, want u hoeft alleen het schema bij de eigenschappen van het XML-document te selecteren.

De onderstaande oefening toont u precies hoe dat gaat:

- 1 Start Visual Basic en maak een nieuw project. Type en naam zijn onbelangrijk, maar sla het project wel op.
- 2 Voeg aan het project via **Project, Add New Item** een XML-bestand toe met de naam `Boeken.xml` en voer hier na de kopregel de korte XML-tekst in van het begin van deze paragraaf.
- 3 Voeg via **Project, Add New Item** een XML Schema toe via de sjabloon **XML to Schema** en geef het schema de naam `Boeken.xsd`, klik op **Add**.
- 4 Selecteer het bestand `Boeken.xml` in het volgende dialoogvenster en klik op **OK**.
- 5 Het gemaakte schema is nog niet helemaal klaar voor gebruik. U moet in het schema een kleinigheid veranderen:

```
<xs:element name="Boek" maxOccurs="unbounded">
```

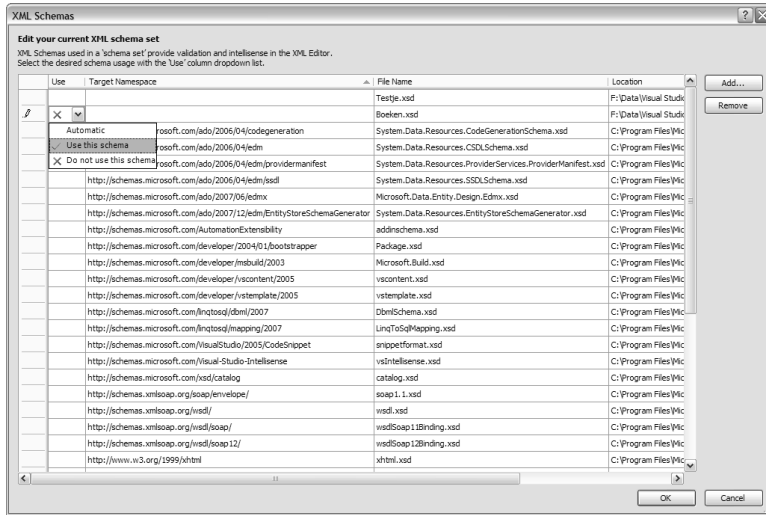
Deze kleine verandering heeft grote gevolgen, want ze zorgt ervoor dat het element `Boek` meermaals mag voorkomen.



#### XML Schema aanpassen

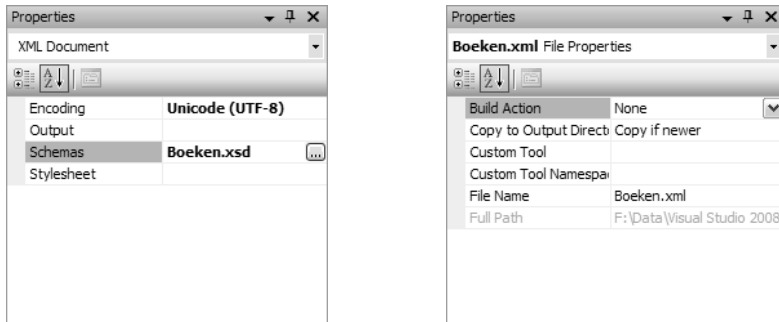
Het is meestal een beetje onhandig dat bij het afleiden van een XML Schema de kindelementen altijd tussen `<xs:sequence>` `</xs:sequence>` staan, wat betekent dat ze in de vastgelegde volgorde moeten voorkomen en ook dat elk kindelement moet voorkomen, zodat de XML voldoet aan de validatie. Het is flexibeler als er, in plaats van `<xs:sequence>` een `<xs:all>` te gebruiken, waardoor de volgorde geen rol speelt. Wilt u bereiken dat een kindelement niet verplicht hoeft voor te komen, geef het kindelement dan het attribuut `minOccurs="0"`.

- 6 Maar ondertussen weet het XML-bestand nog steeds niets van het schema. Open het XML-bestand in de editor en druk op F4. In het venster `Properties` krijgt u nu de eigenschappen van het XML-document te zien en niet de eigenschappen van het XML-bestand (die krijgt u te zien als u het bestand selecteert in de `Solution Explorer`). Klik op de knop bij de eigenschap `Schemas` en selecteer het toe te passen schema in het dialoogvenster `XML Schemas` met een klik op de pijlknop in de kolom **Use** voor het gekozen schema. Selecteer dan **Use this schema** en klik op **OK**. Het geselecteerde schema is nu toegevoegd.



**Afbeelding 15.9** Selecteer het schema dat u aan het XML-document wilt koppelen.

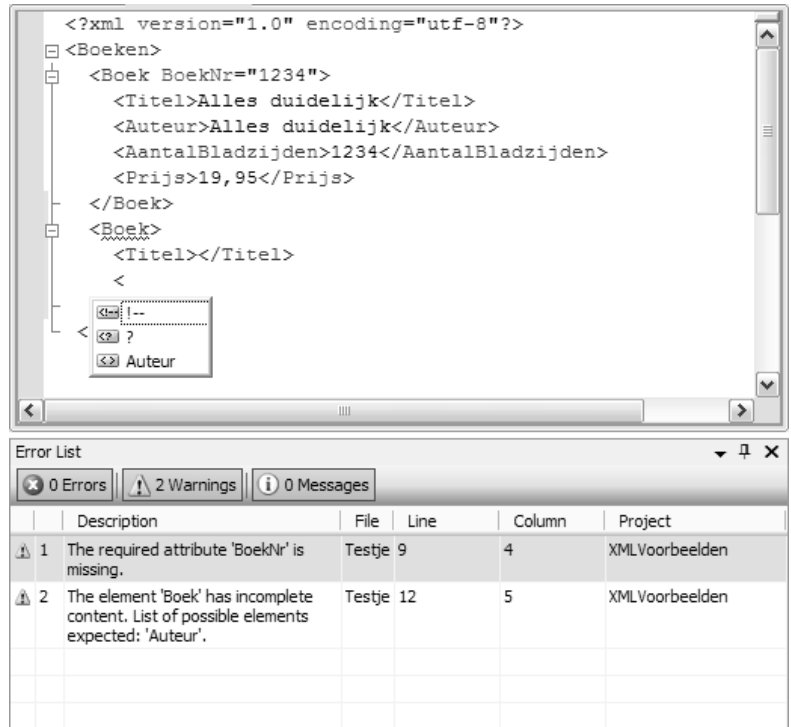
Nu wordt het spannend, wat is het resultaat van de aanwezigheid van het XML Schema? Probeer het rustig uit en voeg na het laatste Boek-element een nieuw element in. U zult zien dat het element Boek in de keuzelijst staat en de invoer van een ander element leidt tot een waarschuwing en een vermelding in het venster Error List.



**Afbeelding 15.10** De eigenschappen van het XML-document (links) zijn iets anders dan de eigenschappen van het XML-bestand (rechts).

Maar ook de invoer van een element Boek leidt tot een waarschuwing en wel omdat het kindelement Titel ontbreekt. Voert u Titel in, dan blijft de waarschuwing staan, want nu ontbreekt Auteur. De waarschuwing verdwijnt pas als alle verplichte elementen zijn ingevoerd.





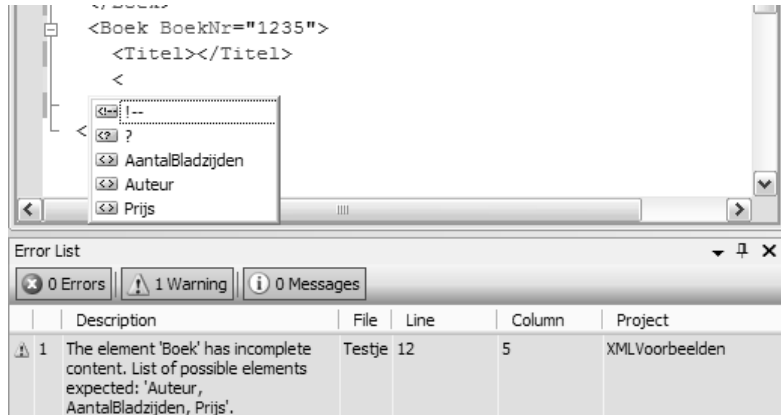
**Afbeelding 15.11** Waarschuwingen en IntelliSense tonen u slechts het volgende element dat u dient toe te voegen.

Vindt u dat niet prettig werken en wilt u wat flexibeler aan de gang? Vervang dan het element `<sequence>` in het XML Schema door het element `<all>`, dan speelt de volgorde geen rol meer en toont IntelliSense alle elementen van Boek.

```
<xs:element name="Boek" maxOccurs="unbounded">
  <xs:complexType>
    <xs:all>
      <xs:element name="Titel" type="xs:string" />
      ...
    </xs:all>
  </xs:complexType>
</xs:element>
```

Wilt u een bepaald element niet verplicht maken, gebruik dan het attribuut `minOccurs` bij dat element:

```
<xs:element name="AantalBladzijden" type="xs:unsignedShort" minOccurs="0" />
```



**Afbeelding 15.12** IntelliSense toont u de resterende elementen voor Boek.

Dit wil zeggen dat het element AantalBladzijden niet verplicht is en dus mag ontbreken.

Een vraag die vooral beginnende lezers stellen, is natuurlijk of XML Schema's echt nodig zijn. Vereist zijn XML Schema's niet, want XML-gegevens zijn ook aanspreekbaar als er geen XML Schema voorhanden is. De validatie met een XML Schema is in uw eigen programma's steeds een vrijwillige aangelegenheid.

XML Schema's stellen de geldigheid van de structuur van een XML-document vast en verzekeren daarmee dat alle vereiste XML-elementen voorhanden zijn. Bovendien krijgt elk XML-element een datatype toegevoegd, wat in bepaalde situaties een voordeel is.

### Een XML Schema direct met een XML-bestand verbinden

De keuze van het schema via het venster Properties is natuurlijk niet de officiële weg, omdat het koppelen van een XML-bestand met een XML Schema ook zonder Visual Basic functioneert. Volgens de officiële weg geeft u de naam van het schema (met de extensie .xsd) op in de kop van het XML-bestand. Daarvoor gebruikt u de namespace xsi (dat staat voor *XML Schema Instance*).

De juiste formulering hangt af van het feit of het schema een namespace gebruikt of niet.

De eenvoudigere variant is zonder namespace. Gebruikt het schema geen namespace, dan verwijst het attribuut `noNamespaceSchemaLocation` of `SchemaLocation` naar het XSD-bestand. De kop van het XML-bestand luidt dan als volgt:

```
<Boeken xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="Boeken.xsd">
```

Het volstaat om de naam van het XSD-bestand op te geven. Dit heeft hetzelfde effect als het toevoegen van het schema via het venster Properties.

Is er een namespace in het spel, dan is het wat ingewikkelder. Bekijk de onderstaande kopregel van een XML Schema met de declaratie van een namespace met de naam `boe`:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="unqualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:boe="urn:XMLVoorbeeld"
  targetNamespace="urn:XMLVoorbeeld">
```

Naast de namespace `xs` declareert het schema een tweede namespace `boe` met de waarde `urn:XMLVoorbeeld`. Hierbij staat *urn* voor Uniform Resource Name en dit is bij een XML Schema de officiële weg om een unieke (eenduidige) naam aan te geven, in dit geval `XMLVoorbeeld`. Het attribuut `targetNamespace` legt de namespace `boe` vast als de namespace voor alle (globale) schema-elementen die niet expliciet gekwalificeerd zijn (dus waar geen namespace expliciet is aangegeven). Belangrijk is in dit verband ook het attribuut `elementFormDefault`. Daar het hier de waarde `unqualified` (het alternatief is `qualified`) heeft, hoeft u niet per se de naam van de namespace voor de kindelementen van `Boeken` te zetten.

Dat was een hoop voorbereidend werk, dat in het XML Schema noodzakelijk was. Dus hoe koppelt u nu het schema aan het XML-bestand? Dat doet u zo:

```
<boe:Boeken xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:boe="urn:XMLVoorbeeld"
  xsi:schemaLocation="urn:XMLVoorbeeld Boeken.xsd">
```

Niet alleen gebruikt u in plaats van het attribuut `noNamespaceSchemaLocation` het attribuut `schemaLocation`, maar de waarde hiervan is tweeledig. Het eerste deel is de naam van de namespace, het tweede deel is de naam van het XSD-bestand. In de declaratie van het rootelement `Boeken` staat de naam van de namespace `boe` voor `Boeken`, maar niet aan de elementen `Boek` en diens kindelementen, dankzij het attribuut `elementFormDefault="unqualified"`. Ook deze werkwijze resulteert in een toekenning van `Boeken.xsd` aan de eigenschap `Schemas` van het XML-document.

## De rol van XML Namespaces

Ook bij XML bestaan namespaces, zoals u in de vorige paragraaf al hebt gezien. Een namespace is een set namen waarin alle namen uniek en eenduidig zijn. Elk XML-element krijgt via een XML-namespace een categorie toegewezen. Daarmee vervullen XML-namespaces dezelfde rol als de .NET-klassenbibliotheek, waar klassen respectievelijk algemene typen een categorie krijgen toegewezen. Elk XML-document heeft vanaf het begin de default namespace, die u niet hoeft te declareren en daarom ook geen directe rol speelt. Alle overige namespaces declareert u in het rootelement als attribuut:

```
<?xml version="1.0" encoding="utf-8" ?>  
<ns0:Boeken xml:ns:ns1="http://xmlvoorbeeld1" xml:ns:ns2="http://xmlvoorbeeld2"  
  xml:ns:ns0="http://xmlvoorbeeld0">
```

In het rootelement `Boeken` declareert u drie namespaces (`ns0`, `ns1` en `ns2`). Het element `Boeken` zelf behoort tot namespace `ns0`.

U gebruikt namespaces om gelijknamige, afzonderlijke elementen van een unieke naam te voorzien:

```
<ns1:Boek>  
  <Titel>Boek 1000</Titel>  
</ns1:Boek>  
<ns2:Boek>  
  <Titel>Boek 2000</Titel>  
</ns2:Boek>  
<Boek>  
  <Titel>Boek 9000</Titel>  
</Boek>
```

Per namespace kunt u verschillende ‘vocabulaires’ declareren, waarmee elk element aan een vocabulaire (*vocabulary*) kan worden gekoppeld. Geeft u bij het aanspreken van de XML-gegevens een vocabulaire aan, dan komen alleen de elementen uit dit vocabulaire in aanmerking. Een toepassing waarop we in dit hoofdstuk echter niet verder ingaan, zou zijn om alleen de inhoud te filteren uit een Word 2007-document dat in XML-formaat voorhanden is. De inhoud herkent u omdat deze behoort tot de WordML-vocabulaire in de namespace `w`.

Tipje: verander de extensie `.docx` in `.zip` en dubbelklik op het bestand om de inhoud in Verkenner te bekijken.

In het volgende voorbeeld maakt u een consoletoepassing die met behulp van de klasse XDocument de inhoud van alle alinea's van een Word 2007-document uitvoert. Om ervoor te zorgen dat het DOCX-bestand niet met de klassen in de namespace System.IO.Packaging wordt aangesproken, kopieert u vooraf het document.xml-bestand uit het DOCX-bestand, waarbij de extensie van het DOCX-bestand was gewijzigd in .zip (dit is niet de normale werkwijze).

```
Imports System.Xml.Linq
Imports <xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
Module Module1
    Sub Main()
        Dim WmlPad As String = _
            Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) & "\Document.xml"
        Dim XDoc = XDocument.Load(WmlPad)
        Dim Alineas = From a In XDoc...<w:document>...<w:body>...<w:p> Select a
        For Each a In Alineas
            Console.WriteLine(a.Value)
        Next
        Console.ReadLine()
    End Sub
End Module
```

## XML-gegevens direct invoeren

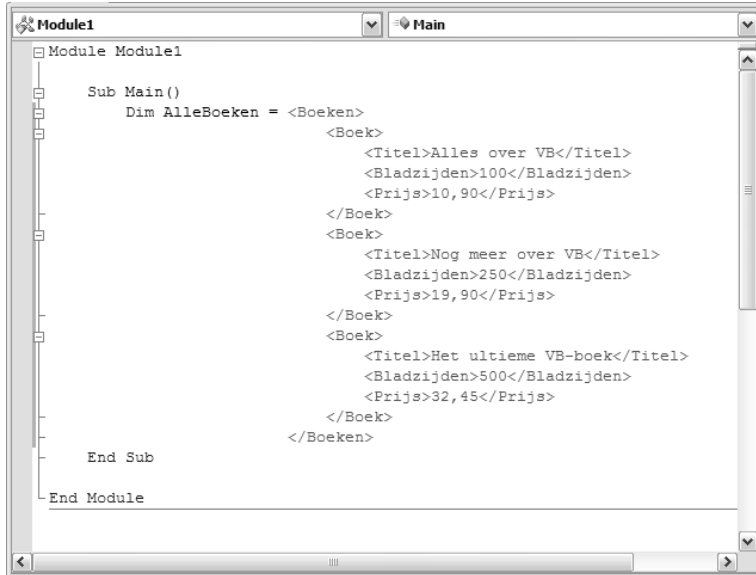
Visual Basic heeft sinds versie 2008 een bijzondere mogelijkheid: u kunt direct XML invoeren in de brontekst van het programma. Probeer het zelf maar eens, maak een nieuw project en geef in de eventprocedure Form\_Load (van een Windows Forms Application) of in Sub Main (van een consoletoepassing) de volgende code in:

```
Dim AlleBoeken = <Boeken>
    <Boek>
        <Titel>Alles over VB</Titel>
        <Bladzijden>100</Bladzijden>
        <Prijs>10,90</Prijs>
    </Boek>
    <Boek>
        <Titel>Nog meer over VB</Titel>
        <Bladzijden>250</Bladzijden>
        <Prijs>19,90</Prijs>
    </Boek>
```

## Hoofdstuk 15 – XML als universeel dataformaat

```
<Boek>
  <Titel>Het ultieme VB-boek</Titel>
  <Bladzijden>500</Bladzijden>
  <Prijs>32,45</Prijs>
</Boek>
</Boeken>
```

Nee, nee, het is geen drukfout en er ontbreken geen aanhalingstekens. U kunt XML-code direct aan een variabele toewijzen zonder dat de XML-gegevens tussen aanhalingstekens moeten staan. Deze variant werkt omdat achter de schermen reguliere XELeMent-objecten schuilgaan, waarbij een formele declaratie niet noodzakelijk is, omdat de Visual Basic-compiler al tijdens de invoer de typen herkent en de XML-boomstructuur kan opbouwen. Dat betekent dat een query van deze XML-gegevens zich in niets onderscheidt van een query van formeel gedeclareerde XML-gegevens.



**Afbeelding 15.13** XML-gegevens direct invoeren in de brontekst, zonder aanhalingstekens en geen foutmelding of waarschuwing te bekennen.

De volgende query retourneert de som van de waarden van alle Prijs-elementen:

```
Dim PrijsAlleBoeken = (From B In AlleBoeken...<Boek> Select CType(B...<Prijs>.Value,
Decimal)).Sum()
```

Om ervoor te zorgen dat u de functie `Sum` kunt toepassen, dient u met `Select` een numerieke waarde te selecteren. Daarom gebruikt u `CType` voor de conversie van `String` naar `Decimal`.



### Punten en komma's

Mocht u in het element `Prijs` een punt in plaats van een komma hebben gebruikt, dan dient u in plaats van `CType` de methode `ToDecimal` van de klasse `XmlConvert` te gebruiken:

```
Dim PrijsAlleBoeken = (From B In AlleBoeken...<Boek>
    Select XmlConvert.ToDecimal(B...<Prijs>.Value)).Sum()
```

De volgende query geeft het gemiddelde aantal bladzijden van alle `Boek`-elementen:

```
Dim GemiddeldAantalBladzijden =
    (From B In AlleBoeken...<Boek> Select XmlConvert.ToDecimal(B...<Bladzijden>.Value)).Average
```

En vergeet niet om de namespace `System.Xml` te importeren, anders is de klasse `XmlConvert` niet beschikbaar.

## Direct ingevoerde XML als tekst opslaan

De directe invoer van XML genereert automatisch `XElement`-objecten en dat maakt het mogelijk om de XML-tekst probleemloos op te slaan. De volgende opdracht toont de XML-tekst van `AlleBoeken`:

```
MessageBox.Show(AlleBoeken.ToString())
```

Of voor een consoletoepassing:

```
Console.WriteLine(AlleBoeken.ToString())
```

En de volgende opdracht slaat de XML-tekst op in een bestand:

```
AlleBoeken.Save("AlleBoeken.xml")
```

## Expressies inbouwen

Wilt u in direct ingevoerde XML ook variabelen en andere expressies opnemen, dan zet u deze uitdrukkingen tussen vishaken met procenttekens en een isgelijktteken: `<%=expressie%>` (kenners van ASP en ASP.Net zal dit vertrouwd voorkomen). Deze techniek is vooral heel praktisch wanneer u de XML-gegevens uit andere gegevens samenstelt:

## Hoofdstuk 15 – XML als universeel dataformaat

```
Dim BTW As Byte = 6
Dim Boek = <Boek>
    <NettoPrijs><%= 19.9 * 1 - BTW / 100 %></NettoPrijs>
</Boek>
```

Het volgende voorbeeld stelt met WMI (Windows Management Instrumentation) enkele basisgegevens over draaiende services samen, waarvan de term *SQL* in de naam voorkomt en maakt hiervan een XML-constructie. Het project dient wel te refereren aan de assembly `System.Management.dll` (bijvoorbeeld via **Project, Add Reference**).

```
Dim Mc As New Management.ManagementClass("Win32_Service")
Dim ServiceLijst As Management.ManagementObjectCollection = Mc.GetInstances()
Dim AantalSQLServices As Integer
Dim SQLServices As New XElement("SQLServices")
For Each Mo As Management.ManagementObject In ServiceLijst
    If Mo.Properties("Name").Value.ToString.Contains("SQL") Then
        Dim SQLService As New XElement("SQLService")
        SQLService.Value = Mo.Properties("Name").Value.ToString
        SQLServices.Add(SQLService)
        AantalSQLServices += 1
    End If
Next
Dim Samenvatting = <Samenvatting>
    <AantalServices><%= AantalSQLServices %></AantalServices>
</Samenvatting>
SQLServices.Add(Samenvatting)
Console.WriteLine(SQLServices.ToString)
Console.ReadLine()
```

### Namespaces inbouwen

Ook in direct ingevoerde XML kunnen namespaces voorkomen. Als voorbeeld nemen we nog een keer de inmiddels vertrouwde XML-constructie waarin we het Boek-element van een namespace voorzien.

```
Dim AlleBoeken = <boe:Boeken>
    <boe:Boek>
        <Titel>Alles over VB</Titel>
        <Bladzijden>100</Bladzijden>
        <Prijs>10,90</Prijs>
    </boe:Boek>
```



```

<boe:Boek>
  <Titel>Nog meer over VB</Titel>
  <Bladzijden>250</Bladzijden>
  <Prijs>19,90</Prijs>
</boe:Boek>
<boe:Boek>
  <Titel>Het ultieme VB-boek</Titel>
  <Bladzijden>500</Bladzijden>
  <Prijs>32,45</Prijs>
</boe:Boek>
</boe:Boeken>

```

Op dit punt toont de compiler een foutmelding, omdat de namespace `boe` nog niet is gedeclareerd. Dat probleem lost u snel op, want sinds versie 2008 kunt u met `Imports` ook XML-namespaces binnenhalen:

```
Imports <xmlns:boe="http://boeken">
```

De namespace dient u natuurlijk ook te gebruiken bij een LINQ-query:

```
Dim PrijsAlleBoeken = (From B In AlleBoeken...<boe:Boek> Select CType(B...<Prijs>.Value,
    Decimal)).Sum()
```

## Een complete XML-toepassing

Tot slot van dit hoofdstuk een XML-toepassing die het samenspel van XML-query's, XML-namespaces en vooral XML Schema demonstreert. Het gaat hier om een WPF-toepassing die een lijst met documenten beheert. U kunt XML-berichten laden, waarbij een bericht ofwel een aanvraag voor een document is, dan wel een upload van een document voorstelt. Via het XML Schema van het geladen XML-document wordt bepaald over welk soort bericht het gaat. U vindt dit programma bij de voorbeeldprogramma's.

In het voorbeeld worden in totaal drie namespaces via `Imports`-opdracht aan het begin van het programma toegevoegd:

```
Imports <xmlns="http://xmlvoorbeeld0">
Imports <xmlns:ns1="http://xmlvoorbeeld1">
Imports <xmlns:ns2="http://xmlvoorbeeld2">
```

## Hoofdstuk 15 – XML als universeel dataformaat

De volgende opdracht geeft alle Boek-elementen uit de namespace n2 aan de WPF-ListBox:

```
ListBox1.ItemsSource = (From B In XDoc.Root.<ns2:Boek> Select B).ToList
```

De validatie gebeurt in de volgende regel:

```
XDoc.Validate(BookSchemas, Nothing, False)
```

BookSchemas is een XmlSchemaSet-object dat vooraf is gemaakt:

```
Dim Xsd1 As SmlSchemaSet = BookSchemas.Add("urn:WPFBoekWinkel:Boekaanvraag", "Boekaanvraag.xsd")
```

Lukt de validatie niet, dan is een exception het gevolg. Als alternatief kunt u deze een eventhandler toewijzen die bij een validatiefout wordt aangeroepen.

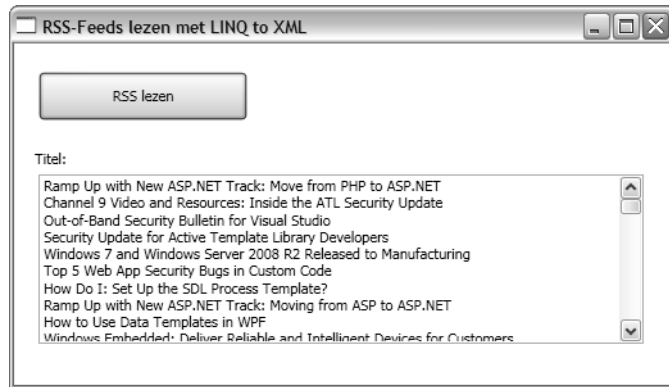
### RSS-feeds lezen

Een geliefd voorbeeld om de eenvoud te tonen waarmee u XML-gegevens – dankzij de klasse XElement en LINQ – kunt verwerken, is een RSS-feed-reader. Een RSS-feed is een eenvoudige XML-structuur die gewoonlijk berichten bevat die afkomstig zijn van een website en die min of meer regelmatig beschikbaar zijn in de vorm van een XML-bestand. Zeer geliefd zijn de RSS-feeds van websites als **Flickr.com**, die het resultaat van een query retourneren en bijvoorbeeld de URL's van de opgevraagde foto's bevatten.

Een RSS-feedreader leest deze XML-inhoud en toont de inhoud. Er zijn verschillende RSS-standaards, zoals ATOM, RSS 1.0 en RSS 2.0, die slechts verschillen in de structuur van de XML-inhoud en andere details. Een RSS-feedreader hoeft daarom slechts de XML-gegevens te laden en de verschillende elementen aan te spreken om bijvoorbeeld de titels van de in de feed opgeslagen berichten te tonen.

Hoe simpel dat in de praktijk is, toont de volgende code wel, die de RSS-feed van de Visual Basic-teams bij Microsoft in Redmond laadt en via het Title-element de verschillende berichten in een WPF-ListBox toont.

```
Dim RssFeedUrl As String = "http://msdn.microsoft.com/vbasic/rss.xml"  
Dim RssFeed As XElement = XElement.Load(RssFeedUrl)  
Dim RssTitle = From Item In RssFeed.Elements("channel").Elements("item").Elements("title")  
Select Item.Value  
liTitle.ItemsSource = RssTitle.ToList
```



**Afbeelding 15.14** De RSS-feeder aan het werk. Hier ziet u waar de Visual Basic-teams in Redmond mee bezig zijn.

Eerst wordt de hele XML-inhoud in een `XElement`-object geladen. Aansluitend vat een simpele LINQ-query alle `Title`-elementen samen in een lijst die aan de `ListBox` wordt gekoppeld.

## Samenvatting

XML is sinds jaren een gevestigde naam als standaard voor het doorgeven van gegevens zowel tussen verschillende toepassingen als ook als tussenformaat voor internet. Met de klassen `XDocument`, `XElement` en `XName` zowel met de mogelijkheid om XML-code direct in de brontekst in te voeren, is Visual Basic goed toegerust voor de verwerking van XML. De mogelijkheid om uit willekeurige XML een schema af te leiden, rondt het XML-comfort zinvol af. In het bijzonder voor Silverlight-browsertoepassingen, waar XML-gegevens uit veelsoortige bronnen (RSS-feeds, webquery's, JavaScript-objecten en dergelijke) moeten worden samengevat, zijn deze capaciteiten van belang.



## Oplossingen van de oefeningen

**I**n deze bijlage vindt u de antwoorden op de vragen aan het eind van elk hoofdstuk. Neem de tijd en werk de antwoorden een-voor-een door. U leert zo nog een paar dingen over het programmeren in Visual Basic die in het boek niet aan de orde komen.

## Hoofdstuk 1

- 1 Daarmee slaat u alle bij het project behorende bestanden op in de geselecteerde map. Doet u dat niet, dan slaat Visual Basic ze op in een tijdelijke map, wat wil zeggen dat ze na een crash van Visual Basic relatief moeilijk (of helemaal niet) zijn te repareren.
- 2 In de map *Visual Studio 2008\Projects* als submap van *Mijn Documenten*.
- 3 Visual Basic bouwt (*build*) het project en runt het in Debug-modus. Dat wil zeggen, Visual Basic compileert alle bij het project behorende brontekstbestanden. Dit resulteert in een EXE-bestand, de compiler slaat dit op in de defaultmap voor uitvoer in de Debug-modus (*bin\debug*). Daarna start de uitvoering van het project in de debugger.
- 4 Nee, omdat de resources bij het bouwen deel van het EXE-bestand worden.
- 5 Bij de projecteigenschappen (**Project**, *<projectnaam>* **Properties**) op het tabblad **Application** bij **Assemblyname**. Dit is standaard de gekozen projectnaam.
- 6 Alleen als op de andere pc .NET Runtime al geïnstalleerd is. Aangezien de actuele versie van .NET Runtime 3.5 ongeveer 200 MB groot is, is het een goed idee om dat bestand als bijlage mee te sturen (dat zou de vriendschap wel eens op de proef kunnen stellen). In plaats daarvan kan hij beter een link naar de Microsoft-website in zijn bericht opnemen, met het verzoek om indien nodig .NET Runtime te downloaden en te installeren.

## Hoofdstuk 2

- 1 Een project is een kunstmatig begrip dat alleen in het kader van de ontwikkelingsomgeving van Visual Basic bestaat. Het project staat voor alle bij het programma behorende bestanden. Het projectbestand heeft de extensie *.vbproj* en bevat naast de namen van de tot het project behorende bestanden ook de projectinstellingen. Visual Studio-projectbestanden zijn XML-bestanden bedoeld voor *MSBuild*, een onderdeel van .NET Framework dat zorgt voor de uiteindelijke bouw van de applicatie. In het projectbestand is geregeld hoe de applicatie uiteindelijk moet worden samengesteld.
- 2 Een solutionbestand (met de extensie *.sln*) vat een of meer projecten samen. Visual Basic maakt standaard een map voor de solution en een submap met dezelfde naam voor het project. De solutionmap kan verschillende projectmappen (met daarin verschillende projecten) bevatten. In de defaultinstelling laat Solution Explorer de solutionmap echter niet zien. U kunt deze instelling wijzigen via het menu **Tools**, **Options**, bij **Projects and Solutions**. Zet een vinkje bij **Always show solution**.

In eerste instantie worden de bestanden van een nieuw project opgeslagen in een tijdelijke map (voor XP is dat *%UserProfile%\Local Settings\Application Data\Temporary Projects*, voor Vista *%UserProfile%\AppData\Local\Temporary Projects*). Slaat u het project op via **Files, Save All**, dan gebeurt dat op de locatie die is opgegeven bij **Tools, Options, Projects and Solutions, Projects location**. Uiteraard kunt u deze locatie altijd wijzigen. Wilt u het project direct automatisch opslaan, zet dan een vinkje bij **Tools, Options, Project and Solutions, Save new projects when created**.

- 3 Een Visual Studio-solution correspondeert met een map op de vaste schijf. Alle bij de solution behorende projecten en bestanden zijn opgeslagen in deze map (en submappen). Als u in Solution Explorer een submap maakt voor bijvoorbeeld bitmaps, dan maakt u fysiek op de vaste schijf een submap die onderdeel wordt van de solution of het project.
- 4 Met F5 start u het project in de debugger. Als er breakpoints zijn gezet, dan draait het programma tot het volgende breakpoint. Met Ctrl+F5 start u het project zonder de debugger, eventuele breakpoints worden genegeerd.
- 5 Het is in Visual Studio heel eenvoudig om elke menuopdracht te voorzien van een sneltoets. U doet dat via het menu **Tools, Options**, onder het kopje **Environment** gaat u naar **Keyboard**, selecteer daar de gewenste menuopdracht, bijvoorbeeld **Project.Properties**, geeft de gewenste sneltoets in en klik op **OK** om uw keuze te bevestigen.

## Hoofdstuk 3

- 1 Een Visual Basic-formulier is een object van de klasse `Form` dat een venster representeert. De klasse `Form` is een model van een typisch Windows-venster, dat met zijn eigenschappen en methoden de functionaliteit van een venster ter beschikking stelt.
- 2 Kopieer de control op het klembord (Ctrl+C) en plak de control vervolgens net zo vaak als nodig op het formulier (Ctrl+V).
- 3 Gebruik het menu **Format** met de verschillende opdrachten voor het aanpassen van de grootte, positie en afstand.
- 4 Een eventprocedure is een Visual Basic-procedure die gekoppeld is aan een bepaalde event. .NET Runtime roept de eventprocedure automatisch aan wanneer deze event optreedt.
- 5 Met een dubbelklik op de gewenste event in het venster Properties, dat bij Visual Basic 2008 naast eigenschappen ook events toont.
- 6 Rechtsklik op de Toolbox en kies **Reset Toolbox**.

## Hoofdstuk 4

- 1 Het datatype van een variabele bepaalt welke soort gegevens in de variabele kan worden opgeslagen. Belangrijke datatypes zijn `Integer` (gehele getallen tot ongeveer twee miljard, positief en negatief), `Single` (drijvendekommagetallen met tot zeven plaatsen achter de komma) en `String` (tekenreeksen tot ongeveer twee miljard tekens lang). Ook als een datatype niet dwingend is voorgeschreven, dient u bij de declaratie van een variabele altijd een datatype op te geven. Bij lokale variabelen die bij de declaratie een waarde toegewezen krijgen, is Visual Basic in staat om het datatype te herkennen, zodat een expliciete opgave in die situatie overbodig is. Maar ook dan dient u het datatype op te geven, daar dit de leesbaarheid van de broncode verbetert.
- 2 Heel eenvoudig, het datatype `Byte` reserveert een geheugenruimte van 1 byte voor de variabele. De maximale numerieke waarde die in 1 byte (= 8 bit) kan bevatten is het binaire getal 11111111, dat is decimaal  $2^8 - 1 = 255$ . Een variabele van het type `Byte` kan dus alleen waarden aannemen van 0 tot en met 255. De waarde 1000 valt buiten het bereik van deze variabele.
- 3 `Meetwaarde.Length` geeft het aantal elementen in de array `Meetwaarde`. Het eerste element van een array heeft altijd de index 0, als er tien elementen in de array aanwezig zijn, dan heeft het laatste element in de array de index 9. De fout die de programmeur hier maakt, is dat de lus een keer teveel wordt doorlopen. De juiste formulering is dus:

```
For n As Integer = 0 To Meetwaarde.Length-1
```

- 4 Aan de variabele `Factor` is geen waarde toegewezen, dus deze krijgt de defaultwaarde toegewezen en dat is 0. Binnen de lus vindt de vermenigvuldiging van `Factor` met `Waarde` plaats, maar het resultaat is – ongeacht de waarde van `Waarde` – altijd 0, dus de afbreekvoorwaarde wordt nooit bereikt.
- 5 Het berekenen van priemgetallen is een geliefd programmeervoorbeeld. Het onderstaande programma berekent de priemgetallen in het bereik 3 tot en met 100:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        Handles Button1.Click
            Dim IsPriemGetal As Boolean
            For i As Short = 3 To 100 Step 2
                IsPriemGetal = True
                For j As Short = 3 To Math.Sqrt(i) Step 2
```



```

        IsPriemGetal = (i Mod j) = 0
    If IsPriemGetal = False Then Exit For
Next
    If IsPriemGetal Then
        MessageBox.Show(i)
        ListBox1.Items.Add(i)
    End If
Next
End Sub
End Class

```

In de geneste lussen zijn een paar optimalisaties opgenomen:

- Aangezien gehele getallen (met uitzondering van het getal 2) geen priemgetallen kunnen zijn, worden alleen de oneven getallen getest, vandaar dat de lus een stapgrootte van 2 gebruikt.
- Een getal hoeft slechts door alle getallen tot zijn vierkantswortel gedeeld te worden, anders worden de delingen slechts herhaald (bijvoorbeeld 15 is deelbaar door 3 en door 5, want  $3 * 5 = 5 * 3 = 15$ ).
- Als een deling een rest van 0 oplevert, dan is het geen priemgetal en wordt de For-lus voortijdig verlaten.

Het bovenstaande voorbeeld zou in principe ook al met Visual Basic 1.0 werken. De huidige versie van Visual Basic biedt andere mogelijkheden. Bijvoorbeeld een functie die de waarde True retourneert als de meegegeven argument van het type Integer een priemgetal is:

```

Function IsPriem(ByVal n As Integer) As Boolean
    For m = 2 To Math.Sqrt(n)
        If n Mod m = 0 Then Return False
    Next
    Return True
End Function

```

Aan de functie is weinig bijzonder op te merken. De wijze waarop deze functie sinds Visual Basic 2008 valt aan te roepen is wel bijzonder:

### 1 Als Predicate:

```

Dim CheckPriemP1 As New Predicate(Of Integer)(AddressOf IsPriem)
Dim Getallen = Enumerable.Range(3, 100).ToArray()
Dim Priemgetallen = Array.FindAll(Getallen, CheckPriemP1)
For n = 0 To Priemgetallen.Count - 1
    ListBox1.Items.Add(Priemgetallen(n))
Next

```

## Hoofdstuk C – Oplossingen van de oefeningen

Een *Predicate delegate* staat voor een functie die een parameter van het opgegeven type verwacht en die een waarde van het type `Boolean` retourneert.

### 2 Als *lambda* expressie:

```
Dim CheckPriemP2 As New Func(Of Integer, Boolean)(AddressOf IsPriem)
Dim Getallen = Enumerable.Range(3, 100).ToArray()
Dim Priemgetallen = Getallen.Where(CheckPriemP2)
For i = 0 To Priemgetallen.Count - 1
    ListBox1.Items.Add(Priemgetallen(i))
Next
```

Een *lambda* expressie is een expressie die voor een functie staat. De methode `Where` van de array `Getallen` krijgt een lambda expressie mee, wat tot gevolg heeft dat de functie voor elk getal in de array eenmaal wordt aangeroepen. De enumeration `Priemgetallen` krijgt alleen die elementen van `Getallen` toegewezen waarvoor de functie de waarde `True` retourneert.

### 3 Als *LINQ* expressie:

```
Dim CheckPriemP3 As New Func(Of Integer, Boolean)(AddressOf IsPriem)
Dim Getallen = Enumerable.Range(3, 100).ToArray()
Dim Priemgetallen = From g In Getallen Where IsPriem(g)
For i = 0 To Priemgetallen.Count - 1
    ListBox1.Items.Add(Priemgetallen(i))
Next
```

Deze variant lijkt op variant 2, maar is wat eleganter dankzij de LINQ-operatoren `From`, `In` en `Where`.



**Afbeelding C.1** Het programmaatje *Priem* biedt alle vier de varianten om priemgetallen te genereren.

In alle drie de varianten bevat de enumeration `Priemgetallen` aan het eind de priemgetallen tussen 3 en 100 en in alle drie de gevallen is de array `Getallen` niet in een lus gevuld, maar via `Enumerable.Range(3, 100)`. Deze drie varianten zijn niet beter, maar moderner en een beetje effectiever. U vindt de oplossing en de drie varianten bij de voorbeeldprogramma's.

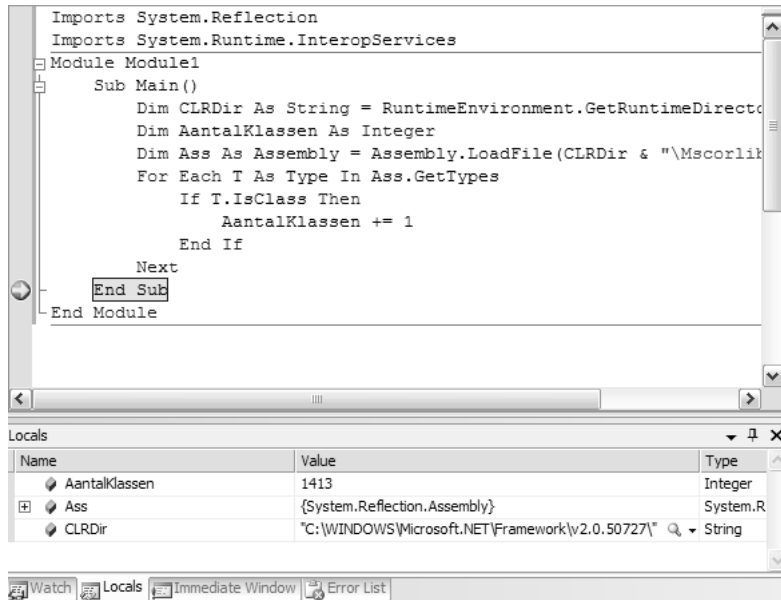
## Hoofdstuk 5

- 1 Een klasse definieert een object, een object is een instantie van een klasse.
- 2 De procedure `New` wordt automatisch aangeroepen bij de instantiatie van de klasse. De procedure `New` noemt men ook wel de *constructor*. Wanneer de procedure `New` als `Shared` is gedeclareerd, dan behoort de constructor tot de klasse en niet tot de instantie van de klasse. Een `Shared`-constructor roept u slechts eenmaal aan, namelijk wanneer de klasse voor de eerste maal wordt aangesproken.
- 3 Een veld is een normale variabele. Een eigenschap wordt gedeclareerd met het keyword `Property` en bezit een `Get`- en `Set`-accessor. De `Get`-accessor gebruikt u om de waarde van een eigenschap uit te lezen, de `Set`-accessor gebruikt u om een waarde aan een eigenschap toe te wijzen. `ReadOnly`-eigenschappen bezitten alleen een `Get`-accessor.
- 4 Als er twee procedures, functies of eigenschappen zijn met identieke namen maar een verschillende signature, dan is dat geen probleem. Dit noemt men overladen. De signature is het aantal parameters en hun datatype.
- 5 De verschillen tussen een interface en een abstracte basisklasse zijn:
  - Een interface moet volledig geïmplementeerd worden, bij een abstracte basisklasse alleen de members die met `MustOverride` zijn gedeclareerd.
  - Een interface kan van verschillende andere interfaces erven, een abstracte basisklasse – zoals iedere klasse – kan van slechts een enkele klasse erven.
  - Een abstracte basisklasse kan ook reguliere members hebben en daarmee programmacode, een interface bestaat uitsluitend uit declaraties.
- 6 De listing bevat de volgende fouten:
  - De eigenschap `RekeningSaldo` is `ReadOnly`, u kunt deze eigenschap geen waarde toewijzen.
  - De constructor van de klasse verwacht geen string als argument, dus mag u bij de instantiatie ook geen string als argument meegeven.
  - De eigenschap `BankNaam` is weliswaar `WriteOnly`, maar u kunt hieraan toch geen waarde toewijzen, aangezien ze niet als `Shared` is gedeclareerd en dus niet direct via de klasse valt aan te spreken.

- In de eigenschap BankNaam wordt de waarde die met het Value-argument is meegegeven, niet aan de variabele mBankNaam toegewezen
- Aangezien er geen constructor is gedeclareerd, kan de eigenschap RekeningNummer niet worden uitgelezen, de variabele mRekeningNummer heeft immers nooit een waarde toegewezen gekregen.

## Hoofdstuk 6

- 1 Er zijn enkele duizenden klassen in de .NET-klassenbibliotheek voorhanden. Het exacte aantal krijgt u door ze te tellen, maar het is niet precies duidelijk welke namespaces precies tot de kern van de .NET BCL (*Base Class Library*) behoren en welke niet. Elke namespace bestaat weer uit verschillende assembly's. De namespace `System.Reflection` biedt een mogelijkheid om assembly's te laden en alle klassen te doorlopen. De onderstaande consoleapplicatie telt bijvoorbeeld de klassen in de assembly `mscorlib.dll`:



**Afbeelding C.2** Dit is de basis, in deze versie bevat de assembly `mscorlib.dll` 1413 klassen.

```
Imports System.Reflection
Imports System.Runtime.InteropServices
Module Module1
    Sub Main()
        Dim CLRDir As String = RuntimeEnvironment.GetRuntimeDirectory()
        Dim AantalKlassen As Integer
        Dim Ass As Assembly = Assembly.LoadFile(CLRDir & "\mscorlib.dll")
        For Each T As Type In Ass.GetTypes
            If T.IsClass Then
                AantalKlassen += 1
            End If
        Next
    End Sub
End Module
```

Aan u de taak om het programmaatje uit te breiden, zodat het alle assembly's in de map van .NET Runtime doorloopt en het totaal naar het beeldscherm uitvoert.

- 2 Importeer de namespace `System.Windows.Forms`, dan kunt u deze in de andere opdrachtregels weglaten:

```
Imports System.Windows.Forms

Dim F As New Form()
F.Controls.Add (New Button)
```

Met `Global` kunt u ook de top van de namespacehiërarchie aanspreken. Dat maakt het wat makkelijker een complete namespace in het programma aan te geven.

- 3 De reden hiervan is, dat de klasse `StringBuilder` onderdeel is van de namespace `System.Text`, dus de programmeur uit Roodeschool dient de namespace met de opdracht `Imports` toe te voegen of de aanroep van de klasse telkens vooraf te laten gaan door de namespace.
- 4 De namespace `My` heeft de belangrijkste functies van de klassenbibliotheek aan boord, zodat deze eenvoudig zijn aan te spreken. De volgende opdracht test of een netwerkverbinding aanwezig is:

```
If My.Computer.Network.IsAvailable Then
```

- 5 Gebruik de volgende code in een consoletoepassing. Deze code bekijkt alle bestanden in de map `C:\Temp` en gebruikt de datum waarop het bestand voor het laatst is benaderd. Is het jaartal kleiner dan 2000, dan wordt het bestand gewist:

## Hoofdstuk C – Oplossingen van de oefeningen

```
Imports System.IO
Module Module1
    Sub Main()
        Dim TempPad As String = "C:\Temp"
        Dim AantalGewist As Integer
        For Each Fi As FileInfo In New DirectoryInfo(TempPad).GetFiles()
            If Fi.LastAccessTime.Year < 2000 Then
                Fi.Delete()
                AantalGewist += 1
            End If
        Next
        Console.WriteLine(String.Format("{0} Bestand(en) gewist.", AantalGewist))
        Console.ReadLine()
    End Sub
End Module
```

Bedenk wel dat elke benadering van het bestand meetelt, dus ook als het bestand is bekeken of gekopieerd. Wilt u de bestanden wissen die voor 2000 zijn gemaakt, gebruik dan `LastWriteTime.Year` in plaats van `LastAccessTime.Year`.

Het pad naar de map tijdelijke bestanden van de aangemelde gebruiker krijgt u overigens met de volgende opdracht:

```
Dim TempPad As String = System.IO.Path.GetTempPath
```

- 6 De consoletoepassing hieronder verruult de extensie `.jpg` voor `.jpeg` en toont de gewijzigde bestanden op het scherm:

```
Imports System.IO
Module Module1
    Sub Main()
        Dim PictPad As String = Environment.GetFolderPath(Environment.SpecialFolder.MyPictures)
        Dim AantalGewijzigd As Integer
        Dim NieuweNaam As String
        For Each Fi As FileInfo In New DirectoryInfo(PictPad).GetFiles("*.jpg")
            NieuweNaam = Path.ChangeExtension(Fi.FullName, "jpeg")
            Console.WriteLine(String.Format("Nieuwe bestandnaam: {0}", NieuweNaam))
            Rename(Fi.FullName, NieuweNaam)
            AantalGewijzigd += 1
        Next
        Console.WriteLine(String.Format("{0} Bestand(en) gewijzigd.", AantalGewijzigd))
        Console.ReadLine()
    End Sub
End Module
```

De functie `ChangeExtension` alleen is niet genoeg, want die schrijft de gewijzigde bestandsnaam niet weg. Dat doet de methode `Rename`.

- 7 De volgende consoletoepassing genereert zes toevalsgetallen tussen 1 en 49 en slaat deze op in een array, sorteert de getallen en schrijft ze dan eenmaal in een binair bestand weg en eenmaal getal voor getal als tekstbestand.

```
Imports System.IO
Module Module1
    Sub Main()
        Dim LottoGetallen(5) As Byte
        Dim R As New Random()
        Dim z As Byte
        For i As Integer = 1 To 6
            Do
                z = R.Next(1, 50)
            Loop Until Array.IndexOf(LottoGetallen, z) = -1
            LottoGetallen(i - 1) = z
        Next
        Array.Sort(LottoGetallen)
        Process.Start("Notepad", "LottoGetallen.dat")
        My.Computer.FileSystem.WriteAllBytes("LottoGetallen.dat", LottoGetallen, True)
        Using Tw As StreamWriter = My.Computer.FileSystem.OpenTextFileWriter("Lottogetallen.txt"
            False)
            For i As Integer = 1 To 6
                Tw.WriteLine(LottoGetallen(i - 1))
            Next
        End Using
        Process.Start("Notepad", "Lottogetallen.txt")
    End Sub
End Module
```

Het binaire bestand bevat zes bytes, één byte voor elk getal. Het tekstbestand gebruikt een byte voor elk cijfer en elk ander teken, zoals de regelomhaal. Het tekstbestand is dus per definitie groter. Op de vaste schijf valt dit bij zulke kleine bestanden echter niet te zien, omdat beide veel kleiner zijn dan de minimumruimte die een bestand op de vaste schijf inneemt.

- 8 De oplossing ligt in het recursieve gebruik van de functie `ListFileSize`, die de grootte van alle bestanden in een map optelt en daarna ook de submappen doorloopt en zichzelf hierbij opnieuw aanroept. De consoletoepassing hierna laat zien hoe dit werkt:

```
Imports System.IO
Module Module1
    Sub Main()
        Dim StartDir As String = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
        ' Voor het geval de map MyDocuments te groot is
        ' StartDir = "C:\Temp"
        Dim TotaleGrootte As Long = ListFileSize(StartDir)
        Console.WriteLine("Totale grootte: {0:n}", TotaleGrootte)
        Console.ReadLine()
    End Sub

    Function ListFileSize(ByVal Dir As String) As Long
        Dim TempSom As Long = GetDirSize(Dir)
        ' Nu alle submappen doorlopen
        For Each SubDir As DirectoryInfo In New DirectoryInfo(Dir).GetDirectories
            TempSom += GetDirSize(SubDir.FullName)
        Next
        Return TempSom
    End Function

    Function GetDirSize(ByVal Dir As String) As Long
        Dim TempSom As Long
        For Each Fi As FileInfo In New DirectoryInfo(Dir).GetFiles()
            TempSom += Fi.Length
        Next
        Return TempSom
    End Function
End Module
```

## Hoofdstuk 7

- 1 De compiler herkent een lambda expression aan het keyword `Function` rechts van het isgelijktken. De juiste expressie is dus:

```
Dim LeeftijdsControle = Function(P) P > 60
```

- 2 De lambda expression zou er zou uit kunnen zien:

```
Dim LeeftijdsControle = Function(Pers1 As Persoon, Pers2 As Persoon) Iff(Pers1.Leeftijd >
Pers2.Leeftijd, Pers1, Pers2)
```

- 3 Toegegeven, de opgave is wat gekunsteld, maar toont wel hoe flexibel de omgang met lambda expressions is.

```
Module Module1
```



```

Sub Main()
    Dim ControleerOfOneven = Function(z As Integer) z Mod 2 = 1
    Dim ControleerOfEven = Function(z As Integer) z Mod 2 = 0
    Dim ControleerGetal() As Func(Of Integer, Boolean) = {ControleerOfOneven,
        ControleerOfEven}
    Dim Getallen() As Integer = {12, 32, 44, 55, 66, 72}
    For i As Integer = 0 To Getallen.Length - 1
        Dim z As Integer = New Random(Now.Millisecond).Next(0, ControleerGetal.Length)
        If z = 0 Then
            Console.WriteLine("Test of oneven: {0}={1}", Getallen(i),
                ControleerGetal(0)(Getallen(i)))
        Else
            Console.WriteLine("Test of even: {0}={1}", Getallen(i),
                ControleerGetal(1)(Getallen(i)))
        End If
        Console.ReadLine()
    Next
End Sub
End Module

```

- 4 Voor objecten met een uniek kenmerk (*key*) is de Dictionary-collection het meest geschikt:

```
Dim Team As New Dictionary(Of Integer, Voetballer)
```

De key is in dit geval van het type Integer, hiervoor gebruikt u bijvoorbeeld het rugnummer van de speler of het lidmaatschapsnummer. De clubnaam komt niet in aanmerking, omdat die niet uniek is voor elke voetballer.

- 5 De LINQ-query luidt als volgt:

```
Dim Getallen() As Integer = {11, 16, 20, 25, 36, 100}
Dim Kwadraten = From z in Getallen Where Math.Sqrt(z) Mod 1 = 0
```

- 6 Bij de LINQ-query is aan het eind het Select-statement vergeten, dat specificeert welk object de query retourneert. Visual Basic voert de query echter toch correct uit. Helemaal correct luidt de query:

```
Dim UitleenTitel = From B in Boeken Where B.Uitgeleend = True Join P in Personen On
    Boeken.PersoonsNr Equals P.PersoonsNr Select B
```

## Hoofdstuk 8

- 1 Zet de onderstaande opdrachten in de `KeyDown`-eventprocedure van de `TextBox`:

```
If e.KeyCode = Keys.Escape Then
    Sender.Text = ""
End If
```

Of met de juiste typeconversie:

```
If e.KeyCode = Keys.Escape Then
    CType(Sender, TextBox).Text = ""
End If
```

- 2 `Down`, `KeyPress`, `KeyUp`.
- 3 Daarvoor plaatst u in de `Validating`-eventprocedure de volgende opdrachten:

```
If IsNumeric(Sender.Text) = False then
    e.Cancel = True
End If
```

- 4 De component `ErrorProvider` en diens methode `SetError` leveren een elegante foutmelding met een knipperend uitroepteken naast de fout.
- 5 De volgende code voegt de namen van de maanden toe aan de `ListBox` `liMaanden`:

```
For m As Short = 1 to 12
    liMaanden.Items.Add(Monthname(m))
Next
```

- 6 Dat is relatief eenvoudig. Voeg de volgende opdrachten toe aan de `DoubleClick`-eventprocedure van de control `ListView`:

```
Dim Mp3Naam As String = ListView1.SelectedItems(0).Text
Process.Start(MuziekPad & "\" & Mp3Naam)
```

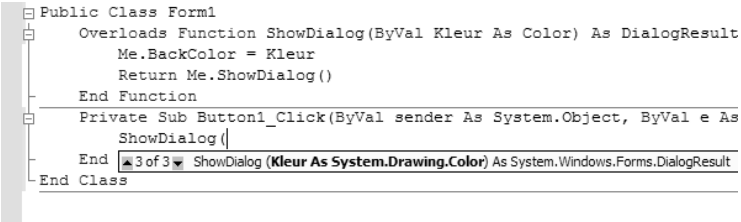
Het is interessant dat u geen mediaplayer hoeft te starten, dat gebeurt indirect door het openen van het MP3-bestand. Daardoor start de mediaplayer die gekoppeld is aan de bestandsextensie `.mp3`.

## Hoofdstuk 9

- 1 In principe kan een formulier een willekeurig aantal werkbalken en menubalken bevatten. Een `GroupBox` is een container die u kunt voorzien van een menubalk en een werkbalk.
- 2 De methode `ShowDialog` kunt u alleen de eigenaar van het venster als optionele parameter meegeven (wat meestal niet relevant is), maar u hebt de mogelijkheid om de methode te overladen. Dat is een eenvoudige techniek, omdat u in de formulierklasse alleen maar een procedure met de naam `ShowDialog` hoeft te declareren die een of meer parameters meekrijgt. De onderstaande methode `ShowDialog` verwacht een kleurwaarde:

```
Overloads Function ShowDialog(ByVal Kleur As Color) As DialogResult
    Me.BackColor = Kleur
    Return Me.ShowDialog()
End Function
```

Het mooie van overlading is dat er nu drie versies van `ShowDialog` ter beschikking staan, iets dat ook de Visual Basic Editor laat zien.



```
Public Class Form1
    Overloads Function ShowDialog(ByVal Kleur As Color) As DialogResult
        Me.BackColor = Kleur
        Return Me.ShowDialog()
    End Function
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
        ShowDialog(|
    End
    3 of 3 ShowDialog (Kleur As System.Drawing.Color) As System.Windows.Forms.DialogResult
End Class
```

**Afbeelding C.3** De methode `ShowDialog` is driemaal overladen en IntelliSense laat u voor elke versie zien welke argumenten de methode verwacht.

De retourwaarde van de methode `ShowDialog` is de button waarmee de gebruiker het formulier heeft gesloten. De eigenschap `AcceptButton` bepaalt welke knop de functie van `OK` vervult (een druk op de `Enter`-toets heeft hetzelfde resultaat als een klik op deze knop) en de eigenschap `CancelButton` bepaalt welke knop de functie van `Cancel` vervult (een druk op de `Esc`-toets heeft hetzelfde resultaat als een klik op deze knop).

- 3 De programmeur heeft een kleinigheidje over het hoofd gezien, namelijk dat `ShowDialog` een blokkerende aanroep is. Dat wil zeggen dat de volgende opdracht pas wordt uitgevoerd als het formulier weer is gesloten. Dus de programmeur kan het formulier aanroepen met de niet-blokkerende methode `Show`, of – en dat is natuurlijk de echte oplossing – hij wijst de titel toe voor de aanroep van het formulier.

- 4 De vraag waarom het hier gaat, is: bestaat een formulier nog als object nadat de gebruiker het heeft gesloten? Het antwoord hangt ervan af of er nog een variabele is die een referentie aan het formulierobject bevat. In het volgende geval staat de inhoud van de textboxes nog ter beschikking:

```
Dim D1g As New Form2
D1g.ShowDialog()
MessageBox.Show(D1g.TextBox1.Text)
```

Sluit de gebruiker het formulier, dan treedt een `Form_Closed`-event op en het venster sluit, maar het object blijft in het werkgeheugen en daarmee ook de inhoud van de textboxes, ook als deze niet zichtbaar zijn. Maar voert u voor het uitlezen van de textboxes de opdracht `D1g = Nothing` uit, dan is de inhoud van de textboxes niet langer aanspreekbaar.

## Hoofdstuk 10

- 1 Het antwoord op deze vraag is vrij eenvoudig, want al tijdens het invoeren krijgt u een waarschuwing dat er iets niet klopt: *Catch block never reached, because System.IO.FileNotFoundException inherits from System.Exception*. Dit wil zeggen dat het tweede catchblok nooit bereikt wordt omdat het eerste altijd actief is, daar wordt namelijk de meest algemene van alle exceptions getest. Wisselen de twee catchblokken van plaats, dan werkt het zoals bedoeld.

```
Sub BestandOpenen()
    Try
        Using Sr As New StreamReader("C:\BestaatNiet.txt")
            '....
        End Using
    Catch Ex As Exception
        '....
        Catch Ex As FileNotFoundException
        'Catch' block never reached, because 'System.IO.FileNotFoundException' inherits from 'System.Exception'
    End Try
End Sub
```

**Afbeelding C.4** *IntelliSense waarschuwt niet alleen dat er iets niet in orde is, maar zegt ook wat er mis is.*

- 2 De waarde van de eigenschap `StackTrace` is een erg lange string, die aan het einde ook het regelnummer bevat. Met een beetje creativiteit en het gebruik van stringfuncties lukt dat op de volgende manier:

```
Dim St As String = Ex.StackTrace
```

```
Dim RegelNr1 As Integer = CType(St.Substring(St.IndexOf("vb:regel ") + 9), Integer)
```

`IndexOf` geeft de positie van *vb:regel* in de string terug, daarbij telt u 9 op, want dat is de lengte van *vb:regel* gevolgd door een spatie en de methode `SubString` retourneert alle tekens tot het eind van de string. Dat de string met een punt eindigt, geeft in dit geval niet, omdat het voor `CType` een toegestaan teken in een getal is.

Een variant hierop maakt gebruik van de zogenoemde *regular expressions*, waarmee u een tekstpassage met behulp van zogenaamde joker-tokens zoekt. Aangezien dit een typisch gevorderd thema is, blijft het bij het volgende voorbeeld en de tip dat de .NET-klassenbibliotheek in de namespace `System.Text.RegularExpressions` een paar nuttige klassen – en in het bijzonder `Regex` – te vinden zijn.

```
Dim St As String = Ex.StackTrace
```

```
Dim Rx As New Regex(".+vb:[a-z]+\s*(?<RegelNr>[0-9]{1,3})", RegexOptions.IgnoreCase)
```

```
Dim RegelNr2 As Integer = CType(Rx.Match(St).Groups("RegelNr").Value, Integer)
```

De regular expression luidt: `.+vb:[a-z]+\s*(?<RegelNr>[0-9]{1,3})` en vindt een uitdrukking die begint met een willekeurig teken (de punt met de +), waarop *vb:* volgt, waarop tenminste een letter volgt (*[a-z]*), waarop nul of meer spaties volgen (*\s\**), waarop drie cijfers tussen 0 en 9 volgen, die in een groep met de naam *RegelNr* worden samengevat.

- 3 De oplossing van de vorige vraag werkt niet als het regelnummer geen deel uitmaakt van de `StackTrace`-string. Dat is het geval als het EXE-bestand zonder Debug-informatie wordt uitgevoerd.
- 4 Of een programma in de Debug-modus draait, kunt u vaststellen door de interne constante `DEBUG` met een opdracht voor voorwaardelijke compilering uit te lezen. In het volgende voorbeeld compileert de compiler de `Console.WriteLine`-regel alleen als het programma in de Debug-configuratie draait:

```
#If DEBUG Then
    Console.WriteLine("Debug...")
#End if
```

Deze regels worden bij de compilering van de brontekst uitgevoerd, in het uiteindelijke programmabestand zijn ze niet meer aanwezig.

- 5 Visual Basic slaat breakpoints op met het project. U hoeft deze dus niet opnieuw in te stellen als u het project opnieuw laadt. Alle breakpoints verwijdert u met **Debug, Delete All Breakpoints** voordat u het project definitief compileert. In Visual Studio – maar niet in de Express Edition – kunt u ook alle breakpoints deactiveren, dat is handig als u nog aan het testen bent en niet alle breakpoints wilt verwijderen.

## Hoofdstuk 11

- 1 Een *thread* is een soort subprocess, dat voor Visual Basic een procedure uitvoert die via de operator `AddressOf` wordt meegegeven. Is de procedure klaar, dan eindigt ook de thread. Elke draaiende thread loopt parallel aan de *main thread* op dezelfde processor respectievelijk dezelfde core. Visual Basic 10.0 zal pas echte parallele verwerking bieden.
- 2 De processor wordt effectiever en efficiënter ingezet, langere operaties kunnen op een eigen thread in de achtergrond draaien, terwijl de main thread de gebruikersinterface en gebruikersactiviteiten afhandelt. Zo wordt voorkomen dat een lang proces de gebruikersinterface bevroest.
- 3 In principe willekeurig veel, op dit punt is geen officiële limiet vastgesteld.
- 4 De klasse `Thread` heeft een Shared-eigenschap `CurrentThread` die de actuele draaiende thread bevat. De klasse `Thread` heeft ook een nieuwe eigenschap `ManagedThreadId` die de ID van de actuele thread bevat. Het is echter veel praktischer de thread aan het begin een naam toe te wijzen. Dan ziet u deze tijdens een programmaonderbreking in de lijst met threads van de IDE terug – behalve bij Visual Basic 2008 Express.
- 5 De programmeur heeft over het hoofd gezien dat een tweede thread niet rechtstreeks naar een stuulement kan schrijven, dat is aan de main thread voorbehouden. Daarom kan de tweede thread de progressbar benaderen, want dat leidt tot een exception. Er zijn twee mogelijkheden om dit bij Windows Forms te voorkomen:
  - Richt een procedure in voor de schrijffactie met een delegate en de methode `Invoke` van de progressbar.
  - Start de tweede thread met de component `BackgroundWorker` en werk de voortgang bij in de methode `ReportProgress`. Dat bespaart u het maken van een delegate en het gebruik van de methode `Invoke`.

Beide varianten zijn in hoofdstuk 11 aan de orde geweest, dus volstaan we hier met de korte versie. U kopieert het bestand met de zelfgeschreven procedure `FileCopy`. De procedure leest het bestand deel voor deel in een `Byte`-buffer en schrijft aan het eind de buffer als geheel naar het doelbestand:

```
Sub FileCopy()  
    ' Bestand in delen in de buffer lezen  
    Dim FiBron As New FileInfo(tbBronbestand.Text)  
    Dim DoelBuf(FiBron.Length - 1) As Byte  
    Dim PortieGrootte As Integer = 65356  
    Dim AantalGelezen As Integer  
    Using FsBron As FileStream = File.OpenRead(tbBronbestand.Text)  
        Do
```

```

    If AantalGelezen + PortieGrootte > DoelBuf.Length Then
        PortieGrootte = DoelBuf.Length - AantalGelezen
    End If
    AantalGelezen += FsBron.Read(DoelBuf, AantalGelezen, PortieGrootte)
    If ckThread.Checked Then
        BackgroundWorker1.ReportProgress(AantalGelezen)
    Else
        ProgressBar1.Value = AantalGelezen
    End If
    If ckThread.Checked = False Then
        StatusMelding(String.Format("{0} Bytes gelezen", AantalGelezen))
    End If
    If FsBron.Position = FiBron.Length Then Exit Do
    ' Let op: Kleine vertraging als "demonstratie-effect"
    Threading.Thread.Sleep(500)
Loop
End Using
' Nu de gehele buffer wegschrijven
Using FsDoel As FileStream = File.OpenWrite(Path.Combine(tbDoelmap.Text,
    Path.GetFileName(tbBronbestand.Text)))
    FsDoel.Write(DoelBuf, 0, DoelBuf.Length)
End Using
If ckThread.Checked = False Then
    ProgressBar1.Value = 0
    StatusMelding(String.Format("Wegschrijven bestand - {0} Bytes", DoelBuf.Length))
End If
End Sub

```

Hoewel de klasse `FileStream` een methode `BeginRead` heeft, die de inhoud van een bestand vanaf het begin asynchroon en daarmee op een tweede thread leest, gebruiken we deze mogelijkheid niet, omdat het in dit voorbeeld in eerste instantie gaat om de inzet van de component `BackgroundWorker`. De bestandsinhoud wordt daarom met de methode `Read` en niet met de methode `BeginRead` ingelezen. Het belangrijkste aspect is in dit verband dat de procedure `FileCopy` zowel op de main thread als op de `BackgroundWorker`-thread kan worden aangeroepen. De gebruiker bepaalt welke variant wordt gebruikt met de `CheckBox` (`ckThread`). Aangezien bij gebruik van de `BackgroundWorker`-thread in `FileCopy` geen directe schrijftoegang naar het sturelement mogelijk is, wordt waar nodig een passende controle ingebouwd. Dit voorkomt pogingen tot schrijftoegang en de daarmee gepaard gaande exceptions. Er is een korte vertraging ingebouwd, zodat er echt een verschil merkbaar is tussen het kopiëren op de main thread en op de extra thread.

## Hoofdstuk C – Oplossingen van de oefeningen

Daardoor treedt het typische effect op dat bij het kopiëren op de main thread het venster al snel bevriest (en dat is ook een manier om het nut van een bepaalde techniek te demonstreren). Maar waarom zou u niet de methode `CopyFile` gebruiken? Daarmee kunt u een bestand kopiëren in een enkele aanroep:

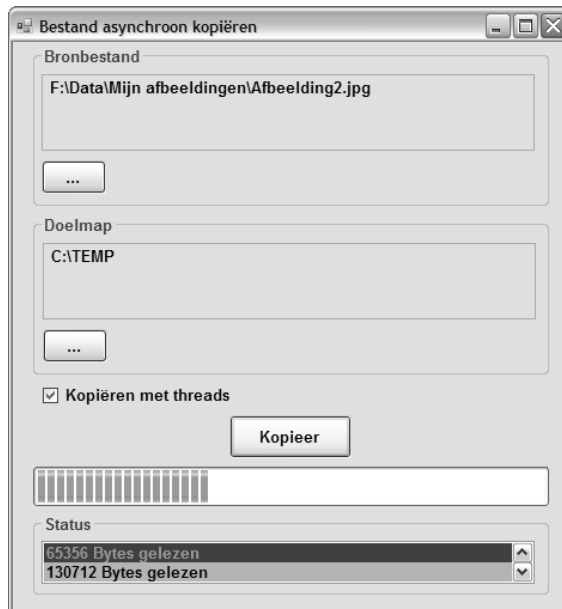
```
My.Computer.FileSystem.CopyFile(tbBronbestand.Text, tbDoelmap.Text)
```

Dat is duidelijk gemakkelijker, maar aangezien het bestand hier in een klap wordt gekopieerd, is er geen mogelijkheid om de voortgangsbalk bij te werken, wat natuurlijk jammer is.

Met de aanroep van `ReportProgress` wordt de voortgangsbalk met het aantal van de al geladen bytes geactualiseerd en wordt het aantal ingelezen bytes getoond:

```
Private Sub BackgroundWorker1_ProgressChanged(ByVal sender As Object, ByVal e As System.ComponentModel.ProgressChangedEventArgs) Handles BackgroundWorker1.ProgressChanged  
    ProgressBar1.Value = e.ProgressPercentage  
    StatusMelding(String.Format("{0} Bytes gelezen", e.ProgressPercentage))  
End Sub
```

Het volledige voorbeeld staat op de website.



**Afbeelding C.5** De `BackgroundWorker`-thread kopieert het bestand.



- 6 Wanneer twee threads parallel lopen, valt niet te voorspellen welke thread als eerste klaar is, daar de processor elke thread een bepaalde processortijd toekent op grond van de prioriteit van de thread. Threads kunnen elk hun eigen prioriteit hebben, zodat het kan gebeuren dat een thread met een lagere prioriteit wordt ingehaald door een thread met een hogere prioriteit. Als deze threads met dezelfde code en variabelen werken, kunnen er fouten optreden (ook wel *race condition* genoemd). Als het belangrijk is dat twee threads in een bepaalde volgorde eindigen, dan moet de ene thread op de andere wachten. Gebruik hiervoor een signaal waarop een thread wacht en dat de tweede thread instelt. Het volgende programmavoorbeeld is gebaseerd op een consoletoepassing. In de procedure `Main` start u naast de main thread nog twee threads die telkens een tellusje aanroepen. Om ervoor te zorgen dan thread T2 steeds na T1 eindigt, moet T2 op T1 wachten. Dat bereikt u het gemakkelijkst met de opdracht `T1.Join()` die voor de start van T2 wordt uitgevoerd en die ervoor zorgt dat de main thread niet eerder doorgaat (en daarmee niet eerder T2 start) dan dat de thread T1 klaar is.

```
Imports System.Threading
Module Module1
    Private wachtSignaal As New EventWaitHandle(False, EventResetMode.AutoReset)
    Sub Main()
        Console.Title = "Thread-synchronisatie met WaitHandle"
        Thread.CurrentThread.Name = "Main thread"
        Dim TS1 As New ParameterizedThreadStart(AddressOf DoeIets)
        Dim TS2 As New ParameterizedThreadStart(AddressOf DoeIets)
        Dim T1 As New Thread(TS1)
        T1.Name = "T1"
        Dim T2 As New Thread(TS2)
        T2.Name = "T2"
        T1.Start(25)
        ' Main thread wacht op beëindiging van T1
        T1.Join()
        T2.Start(25)
        ' Main thread wacht op beëindiging van T2
        T2.Join()
        DoeIets(5)
        Console.ReadLine()
    End Sub

    Sub DoeIets(ByVal Aantal As Integer)
        Dim T As Thread = Thread.CurrentThread
        For i As Integer = 1 To DirectCast(Aantal, Integer)
            Console.WriteLine("{0} zegt: {1}", T.Name, i)
        End For
    End Sub
End Module
```

## Hoofdstuk C – Oplossingen van de oefeningen

```
Next
    Console.WriteLine("{0} is klaar...", T.Name)
End Sub

End Module
```

In een Windows Forms- of WPF-toepassing liggen de verhoudingen wat anders. Hier zou het wachten van de main thread op de beëindiging van een andere thread de hele toepassing lamleggen, wat niet wenselijk is. Hier hebt u een alternatief voorhanden met `WaitHandle`. `WaitHandle` werkt als een verkeerslicht dat door verschillende threads naar keuze op rood of groen kan worden gezet en die ervoor zorgt dat een thread zolang wacht als het verkeerslicht op rood staat en pas verdergaat als het verkeerslicht op groen springt. De volgende code toont de inzet van `WaitHandle` in een erg eenvoudig voorbeeldje, waarin twee threads twee verschillende procedures starten en de ene procedure het verkeerslicht op rood zet, zodat ze niet eerder kan eindigen, dan totdat de tweede procedure het verkeerslicht weer op groen heeft gezet.

```
Imports System.Threading
Module Module1
    Private Verkeerslicht As New EventWaitHandle(False, EventResetMode.AutoReset)
    Sub Main()
        Dim T1 As New Thread(AddressOf DoeIets1)
        T1.Name = "T1"
        Dim T2 As New Thread(AddressOf DoeIets2)
        T2.Name = "T2"
        T1.Start()
        T2.Start()
        Console.ReadLine()
    End Sub

    Sub DoeIets1()
        Dim T As Thread = Thread.CurrentThread
        For i As Integer = 1 To 25
            Console.WriteLine("{0} Zegt: {1}", T.Name, i)
        Next
        Verkeerslicht.Set()
        Console.WriteLine("{0} is klaar...", T.Name)
    End Sub

    Sub DoeIets2()
        Dim T As Thread = Thread.CurrentThread
        For i As Integer = 1 To 25
```

```

        Console.WriteLine("{0} zegt: {1}", T.Name, i)
    Next
    Verkeerslicht.WaitOne()
    Console.WriteLine("{0} is klaar...", T.Name)
End Sub
End Module

```

## Hoofdstuk 12

- 1 Beide databasetypen verschillen fundamenteel van elkaar, ondanks hun sterk op elkaar lijkende namen. Een SQL Server Compact-database is een simpele database gebaseerd op een bestand dat alleen lokaal kan worden aangesproken. Er zijn geen gebruikersbeheer, geen opgeslagen procedures en andere geavanceerde kenmerken. Daar staat tegenover dat deze ook op mobiele apparaten met Windows Mobile draait. Verder is het een voordeel dat deze database hetzelfde SQL-dialect spreekt als grote broer Microsoft SQL Server en als vast onderdeel van .NET Framework 3.5 overal te gebruiken is waar .NET 3.5 Runtime voorhanden is. SQL Server Express is de ietwat uitgekede versie van Microsoft SQL Server, een van de capabelste databasemanagementsystemen op de markt. Visual Basic 2008 Express kan SQL Server Databases binnen de ontwikkelomgeving alleen als databasebestanden aanspreken, niet als Server-databases.

Ook een programma waarvan Visual Basic 2008 Express de broncode uitvoert, kan een SQL Server-database in het netwerk benaderen.

Daartoe dient u de connection string aan te passen, geef in plaats van `.\SQLEXPRESS` de naam van de servercomputer op (en activeer in de SQL Server-configuratie het *Named Pipes*-protocol). Het moge duidelijk zijn dat ook voor de omgang met SQL Server Express een zekere basiskennis van SQL Server- en Windows systeembeheer nodig is. SQL Server-databasebestanden (MDF-bestanden) zijn in principe alleen lokaal aan te spreken.

- 2 De connection string moet beslist het pad naar het MDF-bestand bevatten, uiteraard voorafgegaan door de naam van de lokale computer (die geeft u op met een punt) met daarachter de naam van de SQL Server instantie:

```

Dim CnSt As String = "Data Source=.\SQLServerEx08;Integrated
Security=True;AttachDbFileName=|DataDirectory|\Financieel.mdf;User Instance=true"

```

De plaatshouder |DataDirectory| levert het relatieve pad naar de map waar de toepassing standaard gegevens opslaat (bijvoorbeeld de map Mijn Documenten) en waar de MDF-bestanden zijn opgeslagen. Normaal gesproken heet een instantie van SQL Server Express *SQLEXPRESS*, maar theoretisch kan deze bij de installatie ook een andere naam krijgen, hoewel dit in de praktijk zelden voorkomt.

- 3 De data-adapter verbindt de database via een command-object met een `DataTable`, waarin de via het command-object opgevraagde records worden opgeslagen, zodat deze via databinding beschikbaar zijn voor het programma.
- 4 `CommandBuilder` is de eenvoudigste variant om een data-adapter met de voor het uitvoeren van delete-, insert- en update-operaties noodzakelijke command-objecten te voorzien. Voor elk command-object bouwt `CommandBuilder` een SQL-opdracht en leidt de namen van de velden af uit het al beschikbare select-command-object.  
`CommandBuilder` kan niet gebruikt worden wanneer de data-adapter met een command-object is verbonden dat voor een SQL-opdracht staat die records uit verschillende tabellen koppelt. In dat geval moet de programmeur zelf de opdracht declareren.
- 5 De component `BindingSource` is verantwoordelijk voor de binding van een gegevensbron – in het eenvoudigste geval een `DataTable` – aan een stuelelement dat via de eigenschap `DataSource` of `DataBinding` met de component `BindingSource` is verbonden. De component `BindingSource` is niet uniek voor databases, want ook een collection komt als databron in aanmerking. De component `BindingSource` beheert ook de recordpointer die vastlegt welke veldinhoud een gekoppeld enkelvoudig stuelelement (zoals een textbox of een label) toont.
- 6 De oorzaak van de fout is niet onmiddellijk duidelijk, omdat de opdrachten in de `Click`-eventprocedure van de Update-knop correct zijn. Het probleem is dat in de `Form_Load`-eventprocedure de verbinding in het kader van een `Using`-opdracht wordt geopend en dat betekent dat `End Using` de verbinding impliciet sluit. En met een gesloten verbinding kan de data-adapter niet functioneren. Het is in de meeste gevallen raadzaam de verbinding voor de gehele duur van het programma open te houden, wat wil zeggen dat u het connection-object niet in het kader van een `Using`-statement maakt. Ook als u regelmatig hoort dat een verbinding zo snel als mogelijk weer gesloten moet worden om systeemresources te sparen, dat geldt niet voor een typische Windows-toepassing.

## Hoofdstuk 13

- 1 WPF is niet de hoogvlieger die alle voorgangers ver achter zich laat, maar het heeft een aantal duidelijke voordelen ten opzichte van Windows Forms:
  - Er is een duidelijke scheiding tussen vormgeving en programmacode.
  - WPF biedt verregaande grafische mogelijkheden met vectorafbeeldingen en DirectX.
  - WPF biedt een fundamentele moderne aanzet die ertoe leidt dat de interface duidelijk effectiever door de toepassing gestuurd kan worden dan wat bij Windows Forms mogelijk is.
 Maar ook Windows Forms heeft zijn voordelen:
  - Windows Forms biedt een vertrouwd en overzichtelijk programmeermodel.
  - Visual Basic 2008 heeft een zeer goede designer, die op alle punten goed voldoet.
  - Windows Forms is veelzijdig en biedt onder andere 2D-afbeeldingen, omvangrijke bitmapbewerkingen en andere speeltjes die ook ervaren programmeurs lang niet allemaal kennen.
- 2 De XAML-code zou bijvoorbeeld zo kunnen luiden:

```
<StackPanel>
  <TextBox>Textbox1</TextBox>
  <TextBox>Textbox2</TextBox>
  <TextBox>Textbox3</TextBox>
</StackPanel>
```

Wanneer u in plaats van een Grid een StackPanel als container gebruikt, worden de textboxes automatisch onder elkaar geplaatst.

- 3 Om ervoor te zorgen dat de drie textboxes er niet zo bleekjes uitzien, declareert u een passende Style in bijvoorbeeld <Windows.Resources>:

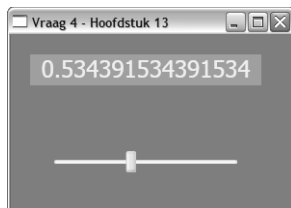
```
<Style TargetType="TextBox">
  <Setter Property="Background" Value="LightBlue"/>
  <Setter Property="Margin" Value="10,10,10,10"/>
  <Setter Property="TextAlignment" Value="Center" />
  <Setter Property="Width" Value="200" />
  <Setter Property="FontSize" Value="16"/>
  <Setter Property="FontFamily" Value="Arial"/>
  <Setter Property="FontWeight" Value="Bold"/>
</Style>
```

- 4 Het verbazingwekkende bij WPF is dat deze opdracht compleet in XAML-code te realiseren is. Het steekwoord daarbij is *property binding*. Property binding koppelt de waarde van een eigenschap aan de waarde van iets anders, bijvoorbeeld een variabele of een expressie. In dit geval koppelt u de waarde van de eigenschap `Opacity` van de container `Canvas` aan de eigenschap `Value` van de control `Slider`. De wijze waarop de eigenschap `Opacity` wordt gebonden speelt een doorslaggevende rol:

```
<Canvas Background="DarkBlue" >
  <Canvas.Opacity>
    <Binding ElementName="KleurSlider" Path="Value"/>
  </Canvas.Opacity>
</Canvas>
```

Verandert de eigenschap `Value` van de control `Slider`, dan krijgt de eigenschap `Opacity` automatisch de actuele waarde van `Value`. Hier is de volledige XAML-code, waarbij ook een `TextBlock`-control aan de eigenschap `Value` van de control `Slider` wordt gekoppeld:

```
<Window x:Class="Window1" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Vraag 4 - Hoofdstuk 13"
  Height="300" Width="300">
  <Canvas Background="DarkBlue" >
    <Canvas.Opacity>
      <Binding ElementName="KleurSlider" Path="Value"/>
    </Canvas.Opacity>
    <Slider Minimum="0.2" Maximum="1" Value=".5" Canvas.Left="40" Canvas.Top="120"
      Width="200" Name="KleurSlider">
    </Slider>
    <TextBlock Foreground="Yellow" FontSize="24" Background="red" Canvas.Top="20"
      Canvas.Left="20" Height="32" Width="240" TextAlignment="Center" Name="OpacityWaarde">
      <TextBlock.Text>
        <Binding ElementName="KleurSlider" Path="Value"/>
      </TextBlock.Text>
    </TextBlock>
  </Canvas>
</Window>
```



**Afbeelding C.6** De slider.