Elmar Geese
Markus Heiliger
Matthias Lohrer

XML, XSLT, VB und ASP

Praktisches XML-Wissen für Webprojekte

Galileo Computing

4 Überblick über die XML-Bausteine

In diesem Kapitel wird ein Überblick über die XML-Bausteine geliefert. Es werden die technischen Grundlagen für das Verständnis von XML vermittelt.

4.1 XML-Grundregeln

4.1.1 Wohlgeformte und gültige Dokumente

Das Regelwerk von XML läßt sich in zwei Aufgabengebiete trennen:

- Das eine Aufgabengebiet befaßt sich mit der Struktur, die einem Dokument gegeben sein muß. Man spricht von *wohlgeformten* Dokumenten, wenn die entsprechenden Regeln eingehalten werden.
- Das andere Aufgabengebiet befaßt sich mit der Syntax. Wird diesem Teil des Regelwerks entsprochen, spricht man von *validen* oder *gültigen* Dokumenten.

Dokumente können wohlgeformt sein, aber dennoch syntaktisch invalide. (Der umgekehrte Fall ist natürlich ausgeschlossen.) XML-Parsern ist es möglich, Dokumente zu verarbeiten, die nicht dem gesamten Regelwerk entsprechen – aber wohlgeformt müssen die Dokumente mindestens sein. Auf das Arbeiten mit gültigen Dokumenten wird im Kapitel *DTD und Schema* genauer eingegangen.

4.1.2 Struktur eines XML-Dokuments

XML wird in einer Baumstruktur modelliert.

Nachfolgend die wichtigsten Regeln für den Umgang mit XML:

- Jedes XML-Dokument enthält einen eindeutig benannten Tag (Root-Tag), der alle anderen datenenthaltenden Tags umschließt.
- Tags müsssen immer geschlossen werden, im Gegensatz zu HTML, das leere Tags zuläßt. Hierbei muß die Benennung des öffnenden und schließenden Tags exakt übereinstimmen (case-sensitive).
  ```
  <Chapter>
  </Chapter>
  ```
- Für Tags, die keinen Wert umschließen, kann anstatt des vorangegangenen Beispiels auch der folgende Shortcut genutzt werden:
  ```
  <Chapter/>
  ```

- Elemente können geschachtelt werden. Überschneidungen von Elementen sind nicht möglich.

```
<Document>
   <Chapter>XML-Bausteine
      <Heading_2>Der Sprachstandard
      </Heading_2>
   </Chapter>
</Document>
```

- Jedes Element kann Eigenschaften haben.

```
<Document name="XML-Grundwissen" created="01-01-2000">
   <Chapter start="1" end="28240" style="Heading 1">XML Bausteine
      <Heading_2 start="25" end="140" style="Heading 1">
      Der Sprachstandard
      </Heading_2>
   </Chapter>
</Document>
```

- Eigenschaften werden immer in Anführungszeichen eingeschlossen.

```
<Document name="XML-Grundwissen" created="01-01-2000">
</Document>
```

- Eigenschaften können keine Elemente enthalten.
- Reservierte Zeichen können nicht innerhalb des Wertebereichs von Elementen bzw. Attributen genutzt werden.

```
< =&lt;
& =&amp;
> =&gt;
" =&quot;
' =&apos;
```

- Daten, die ungültige, d. h. für XML mißverständliche Zeichen beinhalten, werden in CDATA-Klammern eingeschlossen. Der Parser ignoriert dann die Anweisungen in diesen Blöcken.

4.2 Die Verarbeitung

Die XML-Spezifikation beinhaltet die Vorgaben, nach denen XML verarbeitende Software entwickelt wurde bzw. wird. XML ist für die meisten Betriebssysteme durch native oder Java-Lösungen verfügbar. Diese Basiskomponenten leisten folgende Features:

- Parser lesen und schreiben XML-Strukturen.
- Filter selektieren Elemente.

- Prozessoren interpretieren und transformieren XML-Dateien.
- Durch XML-Schemas können Dokumenttypen abgebildet werden.
- Parser können XML-Dateien gegen Schemas prüfen.

4.3 XSL-Stylesheets

Die Extensible Stylesheet Language (XSL) ist für die Aufbereitung der Inhalte von XML-Dokumenten zuständig. Hierbei gibt es keine Beschränkung hinsichtlich des Zielformats. Dies kann sowohl eine Textdatei als auch ein HTML-Dokument sein.

Die XSL-Anweisungen werden zur Laufzeit von einem XSL-Prozessor interpretiert. Dabei werden die Anweisungen auf eine verknüpfte XML-Datei angewendet. Die Zuordnung unterschiedlicher Stylesheets zu einer XML-Datei ermöglicht verschiedene Darstellungen der gleichen Daten.

Durch Filterausdrücke ist es möglich, Untermengen der Daten einer XML-Datei abzubilden.

4.4 XPath

XPath ist die Abfragesprache für XML-Dokumente. In XPath können Filterausdrücke formuliert werden, die Anwendung ermöglicht erst den produktiven Einsatz von XSL sowie die Navigation innerhalb eines XML-Dokuments mit einem XML-Parser.

4.5 DTD

Document Type Definitions (DTD) beschreiben die Struktur, der ein XML-Dokument entsprechen muß, wenn dieses auf eine DTD verweist.

Im Gegensatz zu SGML sind DTDs in XML optional.

Dieselbe Aufgabe wie DTDs können auch XML-Schemas übernehmen.

4.6 Schema

Schemas sind vergleichbar mit DTDs. Sie ermöglichen, daß Dokumente auf ihre Beschaffenheit geprüft werden können. Eine Besonderheit ist hierbei, daß Schemas, im Gegensatz zu DTDs, auch offene, d. h. erweiterbare Modelle beschreiben können. Schemas werden DTDs in einigen Bereichen ablösen, sobald ein einheitlicher Standard verabschiedet und in den Parsern implementiert ist. Ein großer Vorteil der Schemas ist das Format, in dem sie verfaßt sind. Im Gegensatz zu DTDs liegen sie im XML-

Format vor, was zum einen das Erlernen der Syntax erleichtert und zum anderen nicht die Existenz eines entsprechenden Editors voraussetzt. Die Bearbeitung kann in einem XML-Editor erfolgen.

4.7 RDF

RDF stellt den allgemeinsten Ansatz zur Beschreibung von Metadaten zur Verfügung. Das Ziel von RDF ist nicht ausschließlich die Beschreibung von Dokument-Schemas, sondern die Beschreibung beliebiger Ressourcen. Die Ziele von RDF:

- Bessere Recherche nach Web-Dokumenten
- Möglichkeit der Indexierung und Katalogisierung
- Beschreibung virtueller Dokumente
- Profiling

Für die Beschreibung von Schemas durch RDF wurde RDF-Schema erstellt. RDF-Schema ist ein Satz von RDF-Objekten mit dem Ziel, Dokument- und Objektbeschreibungen zu erstellen.

Durch Integration digitaler Signaturen soll RDF ein Standard für E-Commerce und kollaborative Anwendungen werden.

Das Konzept von RDF ähnelt dem eines Klassensystems einer objektorientierten Programmiersprache. Klassen können wiederverwendet und auch verfeinert werden. Eine Sammlung von Klassen einer Applikation in einer RDF-Datei wird als Schema bezeichnet. Für diesen Zweck stellt RDF eine eigene Spezifikation zur Verfügung, die die für die Beschreibung von Schemas erforderlichen Klassen enthält: RDF-Schema.

Da bei RDF die Entwicklung noch nicht abgeschlossen ist, ist es für den produktiven Einsatz vielleicht noch etwas früh. Dennoch ist RDF überall dort eine Alternative, wo nicht nur Schemas beschrieben, sondern Objekte modelliert werden sollen. RDF ist als Format für die Beschreibung beliebiger Ressourcen besonders für die Abbildung von Dokumenttypen und von Dokument-Metadaten geeignet.

4.8 XLink

Die XML Linking Language (XLink) erweitert die Möglichkeiten von XML um die Funktionalität von Links auf nicht im Dokument enthaltene Daten.

```
<Document name="XML Grundwissen" created="01-01-2000">
   <Chapter start="1" end="28240" style="Heading 1">XML Bausteine
      <Heading_2 start="25" end="140" style="Heading 1"> Der
```

```
Sprachstandard
      </Heading_2>
    <a xml:link="simple" href="spreadsheet.jpg" show="new" content-
role="basics">basicsspreadsheet
      </a>
   </Chapter>
</Document>
```

Des Weiteren werden durch XLink folgende Arten von Links zur Verfügung gestellt:

- Links, die zu mehreren Zielen führen
- bi-direktionale Links
- Links für Annotationen, zum Beispiel in schreibgeschützten Dokumenten
- Verknüpfungen zu Datenbanken
- Funktionalitäten, wie sie jetzt (wenn überhaupt) nur durch Skripts realisiert werden können wie »expand-in-place«, neues Fenster oder Zielrahmen, automatisches Folgen und weitere

4.9 XPointer

Die XML Pointer Language (XPointer) bietet im Vergleich zu XLinks einen detaillierten Zugriff. Hiermit ist nicht nur die Adressierung eines Dokuments, sondern auch die einer bestimmten Stelle im Inhalt des Dokuments möglich.

Sprungmarke für einen XPoint-Link ist hierbei eine ID.

```
<a id="begin-here">This is a sample.</a>
```

Der folgende XPointer-Link, bezogen auf die vorangegangene Codezeile, würde als Ziel den Buchstaben »i« des Wortes »is« haben.

```
id(begin-here).string(2,"i")
```

Durch XPointer werden hinsichtlich XML die folgenden Mehrwerte geschaffen:

- Links, die auf bestimmte Stellen innerhalb von Dokumenten zeigen, ohne daß diese vom Autor gekennzeichnet sein müssen
- Links können sich dabei sowohl auf einen Bereich beziehen als auch absolut oder relativ angegeben werden.
- einfach lesbare Adressierung

4.10 XHTML

Die bekannteste Auszeichnungssprache ist HTML, das als Sprache des Internets (und ausschließlich dort) große Verbreitung gefunden hat. Die Realisierung einer privaten Homepage erfordert sicherlich nicht den Einsatz von XML. Der Einsatz von HTML auf großen Sites ist jedoch problematisch, denn HTML wurde für andere Zwecke entworfen als die, für die es jetzt eingesetzt wird. Es war eigentlich nur für den Austausch von Dokumenten im wissenschaftlichen Bereich gedacht und wird heute mehr als Bindeglied zwischen Information, Daten und multimedialen Elementen verwendet.

Eine Verarbeitung von in HTML codierten Dokumenten durch Parser ist nur sehr begrenzt möglich. Weder die Struktur noch der Inhalt kann schlüssig aus einem HTML-Dokument abgeleitet werden. Recherchen sind nur über Volltext oder Meta-Tags möglich. Das typische Aussehen von HTML-Quellcode ist allgemein bekannt:

```
<font size="2">Hier ist etwas Text in HTML
<p>Dies ist ein Absatz in HTML
<li>Dies ist ein Listeneintrag in HTML
<p>Dies ist ein weiterer Absatz in HTML
</font>
</p>
```

HTML-Browser sind beispiellos tolerant, denn die meisten von ihnen können dieses Fragment problemlos anzeigen, obwohl es unsauber strukturiert ist. Die o. g. Probleme werden durch diese Toleranz jedoch verschärft. Durch XML kann dieses Dilemma auf zwei verschiedenen Wegen gelöst werden.

- Erstellung von sauber ausgezeichneten Webseiten durch aus XML generiertes HTML
- Einsatz von XHTML, das HTML ablösen wird

In XHTML wird HTML durch XML beschrieben. Dadurch finden die strikten Vorgaben von XML auf HTML Anwendung, so daß die Interpretation von XHTML-Dateien einfach möglich ist. Das obige Beispiel sähe in XHTML so aus:

XHTML

```
<font size="2">Hier ist etwas Text in HTML
    <p>Dies ist ein Absatz in HTML</p>
    <li>Dies ist ein Listeneintrag in HTML</li>
    <p>Dies ist ein weiterer Absatz in HTML</p>
</font>
```

Die Spezifikation zu XHTML finden Sie unter *http://www.w3.org/TR/xhtml1*. Hier finden Sie auch Hinweise dazu, wie man heute bereits XHTML einsetzen kann und dennoch von den Browsern verstanden wird.

4.11 Andere Erweiterungen

Mit der fortschreitenden Verbreitung von XML wuchs und wächst der Anteil von Sprachbestandteilen und Erweiterungen, wie die Implementierung XML-basierter Sprachen, die für spezielle Anwendungszwecke geschaffen werden.

Festgeschrieben werden diese Erweiterungen in DTDs.

Als Beispiel sind nachfolgend einige Erweiterungen aufgeführt.

4.11.1 SMIL

Durch die Synchronized Multimedia Integration Language (SMIL) können in einer XML-basierten Sprache multimediale Daten referenziert werden. Durch SMIL ist es Autoren möglich, Links an Medienobjekte zu binden oder das Layout einer Bildschirmpräsentation zu beschreiben.

Eingesetzt wird SMIL derzeit z. B. im RealPlayer. Hier können Inhalte mit der SMIL-Syntax erstellt werden, um sie dann mit dem RealPlayer wiederzugeben.

Durch Einsatz der SMIL-Syntax in anderen XML-Dokumenten können dort Timing und Synchronisation realisiert werden.

4.11.2 MathML

Ziel der Mathematical Markup Language (MathML) ist es, die Benutzung von mathematischen und wissenschaftlichen Ausdrücken zu erleichtern.

MathML kann zur Codierung von Sprachsynthetisierung eingesetzt werden und somit als Basis für sprachgesteuerte User Agents genutzt werden.

4.11.3 SVG

Das Scalable-Vector-Graphics- (SVG-)Format soll die Beschreibung von Vektorgrafiken vollständig in XML ermöglichen. Für die Web-Entwicklung bietet sich hier ein riesiges Potential, wenn mit Grafiken so dynamisch umgegangen werden kann wie jetzt mit Dokumenten. SVG-Viewer sind als Stand-alone-Anwendungen, aber auch als Plugins/ActiveX-Controls erhältlich, zum Beispiel von adobe (*http://www.adobe.com/svg/*).

4.11.4 UXF

Das UML-Exchange-Format (UXF) ist die Portierung der Unified Modelling Language in das XML-Format. Gründe für diese Portierung sind:

- ein standardisiertes Dateiformat für den Datenaustausch zwischen Entwicklungswerkzeugen unabhängig von der zugrundeliegenden Plattform
- verbesserte Möglichkeiten der Kommunikation zwischen Entwicklern. Insbesondere dann, wenn die Verbindung durch das Inter- bzw. Intranet gewährleistet wird
- die Transparenz des Formats, die erweiterten Möglichkeiten der Aufbereitung und Präsentation der Daten

4.11.5 WML

Die Wireless Markup Language (WML) ist ein Format, das nach dem Flop von WAP-Anwendungen zu zweifelhaftem Ruhm gekommen ist. Zukünftig wird es dennoch eine große Rolle spielen, sobald die Endgeräte den Standard auch hinreichend unterstützen und eine bessere Übertragungsbandbreite gegeben ist.

5 Eine Welt aus Knoten – Das Document Object Model

Das Document Object Model (DOM) stellt eine Schnittstellenbibliothek zur Verfügung, mit der Visual Basic und andere Programmiersprachen XML-Dokumente bearbeiten können.

5.1 Der Weg zum DOM

Vielleicht erinnern Sie sich an den »Browser-Krieg« zwischen dem Netscape Navigator und dem Microsoft Internet Explorer. Im Gefolge dieses Kampfes um Marktanteile kam es bei den Browsern zu inkompatiblen JavaScript-Objekthierarchien. Die JavaScript-Implementierung des Internet Explorers greift über das `all`-Objekt auf die Bestandteile einer HTML-Seite zu. Der Netscape Navigator benötigt aber die eigene Layer-Technologie für dynamisches HTML. Wer in beiden Browsern eine gute Figur machen will, muß daher auch heute noch die DHTML-Bestandteile doppelt programmieren, einmal für den Internet Explorer und einmal für den Netscape Navigator. Das ist ärgerlich und teuer.

Nicht zuletzt diese Fehlentwicklung hat das W3C veranlaßt, bei der Konzeption von XML nicht nur die Struktur von XML-Dokumenten zu standardisieren, sondern auch ein Schnittstellenmodell festzulegen, über das unterschiedliche Programmiersprachen plattformunabhängig auf diese Dokumente zugreifen können. Genau dieses Schnittstellenmodell bietet das DOM.

msxml 3.0 ist nahezu vollständig W3C-kompatibel

Der msxml-3.0-Parser implementiert die *DOM Level 1 Recommendation* des W3C vom 1.Oktober 1998. In dieser Empfehlung definiert das W3C eine fundamentale und eine erweiterte Gruppe von Schnittstellen. Die als fundamental bezeichnete Gruppe umfaßt solche Elemente, die unbedingt nötig sind, um mit XML- oder auch mit HTML-Dokumenten zu arbeiten. Die »Extended«-Gruppe enthält Schnittstellen, die nur bei der Arbeit mit XML-Dokumenten notwendig sind.

Es folgt die Gretchenfrage: Hat sich Microsoft an diesen Standard gehalten? Die Antwort lautet: Mit einer Ausnahme ja. Sowohl die fundamentalen als auch die erweiterten Schnittstellen sind in msxml 3.0 implementiert. Die Ausnahme betrifft das `DOMException`-Interface, das im msxml-3.0-Parser nicht implementiert ist und durch das `parseError`-Objekt ersetzt wird.

Außerdem hat Microsoft noch weitere Objekte, Methoden und Eigenschaften hinzugefügt. Beispiele sind etwa die `selectNodes`-Methode, die einen XPath-Ausdruck entgegennimmt, und die `transformNode`-Methode, mit der XSLT-Transformationen durchgeführt werden können. Auch das `parseError`-Objekt, das an die Stelle des fehlenden `DOMException`-Interfaces tritt, ist eine Microsoft-Erfindung, und selbst die Methoden `Load` und `Save`, mit denen XML-Objekte geladen und wieder gespeichert werden, sind nicht vom W3C standardisiert. Da man aber schwerlich ohne `Load` und `Save` wird auskommen können und `parseError` gute Dienste leistet, wird man in der täglichen Arbeit wahrscheinlich auch die Microsoft-Erweiterungen nutzen.

DOM zum ersten, zweiten, dritten ...

Das W3C hat die Entwicklung des DOM noch nicht abgeschlossen. Vom 13. November 2000 datieren die *DOM Level 2 Core Recommendation* sowie einige weitere Empfehlungen, die sich auf die Bereiche Views, Style, Events, und Traversal-Range beziehen. In Arbeit ist außerdem bereits die *DOM-Level-3-Spezifikation*. Als Programmierer sieht man diese Entwicklung mit einem lachenden und einem weinenden Auge. Das Versprechen einheitlicher Schnittstellen klingt gut. Wären da nur nicht diese vielen Zwischenschritte, die sich auf dem Weg in die Zukunft in Form von halbfertigen Parsern auf den Rechnern der Kunden einnisten würden.

5.2 DOM-Grundlagen

Die zentrale DOM-Schnittstelle bildet `IXMLDOMNode`. Im Prinzip kann jede Komponente eines XML-Dokuments über diese Schnittstelle angesprochen werden: Elemente, Attribute, Kommentare, Processing Instructions und sogar das komplette Dokument selbst bilden einen solchen Knoten. Manche Knoten können andere Knoten enthalten. Diese sind dann in einem Objekt vom Typ `IXMLDOMNodeList` aufgelistet, so daß das gesamte XML-Dokument in einer Baumstruktur dargestellt wird. Abgesehen von der allgemeinen `IXMLDOMNode`-Schnittstelle gibt es für die speziellen Knotentypen auch spezielle Schnittstellen, etwa `DOMDocument`, `IXMLDOMElement`, `IXMLDOMProcessingInstruction`, `IXMLDOMText` und weitere. Insgesamt gibt es zwölf spezielle Knotentypen, die in der Tabelle 5.2 genau aufgelistet werden. Objekttechnisch gesehen, erben diese Knoten die Methoden und Eigenschaften von dem grundlegenden `IXMLDOMNode`. Abgesehen von ihrer unterschiedlichen Funktion unterscheiden sich diese Knotentypen jeweils darin, in welchen Knotentypen sie enthalten sein können und welche Knotentypen sie selbst wieder enthalten können. Die Tabelle 5.2 macht hierzu genaue Angaben.

Um die diversen Knotentypen genauer kennenzulernen, nehmen wir als Beispiel zunächst das folgende kleine XML-Dokument (CD-LW:\DOM\neuesProdukt.xml):

```
<?xml version="1.0" encoding="iso8859-1"?>
<Kaufhaus>
  <Abteilung Name="Bekleidung">
     <Produkt>
       <Name>Mütze</Name>
       <Farbe>weinrot</Farbe>
       <Preis Währung="DM">50</Preis>
       <!-- eine schöne Mütze -->
     </Produkt>
   </Abteilung>
</Kaufhaus>
```

Nach dem Einlesen in Visual Basic wird dieses XML-Dokument folgendermaßen in DOM-Objekten repräsentiert:

```
|--das gesamte Dokument: DOMDocument
  '--XML-Deklaration: IXMLDOMProcessingInstruction
  '--Kaufhaus: IXMLDOMElement
    '--Abteilung: IXMLDOMElement
      '--Liste aller Attribute: IXMLDOMNamedNodeMap
      | '--Attribut "Name": IXMLDOMAttribute
      |   '--Bekleidung: IXMLDOMText
      '--Produkt: IXMLDOMElement
        '--Name: IXMLDOMElement
        | '--Mütze: IXMLDOMText
        '--Farbe: IXMLDOMElement
        | '--weinrot: IXMLDOMText
        '--Preis: IXMLDOMElement
        | '--Liste aller Attribute: IXMLDOMNamedNodeMap
        |   '--Währung: IXMLDOMAttribute
        |     '--DM: IXMLDOMText
        '--Kommentar: IXMLDOMComment
          '--eine schöne Mütze: IXMLDOMText
```

Deutlich ist die hierarchische Struktur zu erkennen. Knoten enthalten Knoten, die selbst wieder Knoten enthalten usw. An der Spitze steht ein `DOMDocument`-Objekt. An ihm »hängt« das gesamte XML-Dokument. Der XML-Prolog ist in einer `IXMLDOMProcessingInstruction` gefaßt. Unabhängig

von ihrer jeweiligen Verschachtelungstiefe ist das oberste Element `Kaufhaus` genauso in ein Objekt vom Typ `IXMLDOMElement` gefaßt wie die Elemente `Produkt`, `Name`, `Farbe` und `Preis`.

Zum Abschluß dieses Kapitels werden Sie einige Prozeduren erstellen, mit denen Sie die Struktur eines XML-Dokuments in einer ähnlichen Form ausgeben können.

5.3 Referenz auf mxsml 3.0

Wenn Sie in Visual Basic mit dem DOM arbeiten möchten, müssen Sie als erstes eine Referenz auf den XML-Parser setzen. In Visual Basic verwenden Sie hierfür den Menübefehl **Projekt · Verweise.** Hier selektieren Sie den Eintrag *Microsoft XML, v3.0* (die zugehörige Datei `msxml3.dll` ist standardmäßig unter `//WINNT/System32` zu finden). Wenn Sie vergessen sollten, diese Referenz einzufügen, wird Sie die in Abbildung 5.1 gezeigte Fehlermeldung daran erinnern.

Abbildung 5.1 Die typische Fehlermeldung, wenn Sie die Referenz auf den XML-Parser vergessen haben

Jedenfalls erscheint die Fehlermeldung dann, wenn Sie bei den Deklarationen oder in der übrigen Prozedur einige der DOM-Objekte verwendet haben, z.B. bei der Deklaration:

```
Dim xmlDoc As DOMDocument
```

5.4 Fehlerbehandlung mit der IXMLDOMParseError-Schnittstelle

Das Wichtigste beim Programmieren ist die Fehlerbehandlung. Gute Debug-Möglichkeiten sind besonders beim Einarbeiten in ein neues Gebiet äußerst nützlich. Noch bevor es richtig losgeht, sollen daher die Möglichkeiten der Fehlerbehandlung dargestellt werden.

DOMException-Interface nicht im msxml 3.0 implementiert

Der W3C-Standard sieht für die Fehlerbehandlung eine Schnittstelle mit Namen `DOMException` vor. Der Microsoft-Parser implementiert diese Schnittstelle nicht. Statt dessen bietet das `DOMDocument`-Objekt die `parseError`-Eigenschaft, die ein Objekt mit `IXMLDOMParseError`-Schnittstelle zurückgibt. Dieses Objekt verfügt über eine Reihe nützlicher Eigenschaften, die die Fehlersuche erleichtern. Üblicherweise wird direkt nach dem Laden eines XML-Dokuments geprüft, ob beim Einlesen und Parsen des Dokuments ein Fehler aufgetreten ist. Die entsprechende Programmierfloskel sieht wie folgt aus und kann nach eigenen Vorlieben erweitert werden:

```
xmlDoc.async = False
xmlDoc.Load "D:\DOM\neues Produkt.xml"
If xmlDoc.parseError <> 0 Then
   If xmlDoc.parseError.reason <> "" Then
      Debug.Print xmlDoc.parseError.reason
   End If
   Debug.Print xmlDoc.parseError.Line
End If
```

In der Zeile `If xmlDoc.parseError <> 0` wird die Default-Eigenschaft `errorCode` geprüft. Tabelle 5.1 listet die `parseError`-Eigenschaften auf. Alle Eigenschaften sind schreibgeschützt.

Eigenschaft	Bedeutung
`errorCode`	0, wenn kein Fehler aufgetreten ist. Sonst der Fehlercode. Default-Eigenschaft
`filepos`	Zeichenposition, an der der Fehler aufgetreten ist (»beim 234. Zeichen«)
`Line`	Zeilennummer, an der der Fehler aufgetreten ist
`linepos`	Zeichenposition innerhalb der Zeile
`Reason`	Erklärt die Fehlerursache
`srcText`	Die Zeile, die den Fehler verursacht hat
`url`	Die URL des XML-Dokuments, das den Fehler verursacht hat

Tabelle 5.1 Eigenschaften der `IXMLDOMParseError`-Schnittstelle

Die async-Eigenschaft steuert das Verhalten beim Laden eines XML-Dokuments. Mit der Standardeinstellung TRUE wird die Kontrolle an das Programm bereits zurückgegeben, noch bevor das Dokument komplett geladen ist. Über die readyState-Eigenschaft kann der Download-Status abgefragt und auf das onreadystatechange-Ereignis entsprechend reagiert werden.

Für unsere Zwecke ist es sinnvoller, erst den kompletten Download abzuwarten und anschließend mit der Bearbeitung fortzufahren, so daß wir die async-Eigenschaft stets auf False setzen.

5.5 Die Sonderstellung der Attribute

Am Anfang kann die Art und Weise verwirren, wie Attribute in der Objekthierarchie behandelt werden. Einerseits heißt es, daß alle Informationen eines XML-Dokuments in Knoten gefaßt und in einer Baumstruktur repräsentiert werden. Andererseits werden zwar auch Attribute in Knoten gefaßt, diese sind aber nicht in die Baumstruktur eingebettet. Nehmen wir beispielsweise dieses Fragment:

```
<Preis Währung="DM">50</Preis>
```

Das Objekt child enthalte dieses Element. Die Überprüfung child.hasChildNodes gibt »Wahr« zurück. Das Element enthält genau einen Knoten vom Typ NODE_TEXT mit dem Inhalt »50«. Und wo ist der Attributknoten?

Für den Zugriff auf die Attribute muß man die Attributes-Eigenschaft bemühen. Die Attributes-Eigenschaft gibt die Attribute in Form eines Objekts vom Typ IXMLDOMNamedNodeMap zurück. Wenn child.Attributes.Length > 0 ist, dann sind Attribute vorhanden. child.Attributes(0).xml enthält den Wert Währung="DM". child.attributes(0).nodeName enthält Währung. child.attributes(0).nodeValue enthält DM. Mit getNamedItem kann über den Namen auf ein Attribut zugegriffen werden. child.attributes.getNamedItem ("Währung").xml enthält also ebenfalls Währung ="DM".

So weisen wir der Variablen attrib das Attribut zu:

```
set attrib = child.attributes(0)
```

Kann ein Attribut-Knoten selbst wieder Kindknoten enthalten? Eigentlich doch nicht. Tatsächlich gibt aber attrib.hasChildNodes Wahr zurück. Der Grund: Der Wert des Attributs selbst wird als Textknoten behandelt, und attrib.childNodes(0).xml gibt DM zurück.

5.6 Zwölf Knoten sollt ihr sein

Das DOM definiert insgesamt zwölf verschiedene Typen von Knoten. Diese Knotentypen können jeweils bestimmte Arten von Informationen enthalten. Außerdem unterscheiden sie sich darin, in welchem Knotentyp sie selbst enthalten sein können und welche Knotentypen sie wiederum enthalten können. Über die Eigenschaft nodeType können Sie den jeweiligen Knotentyp abfragen.

5.6.1 Die sechs häufigsten Knotentypen

Die am häufigsten verwendeten Knotentypen haben Sie bereits kennengelernt. Einen Knoten vom Typ NODE_DOCUMENT benötigen Sie, um überhaupt auf das gesamte XML-Dokument zuzugreifen. Die Elemente sind vom Typ NODE_ELEMENT. Textinformationen, egal ob sie in Elementen oder in Attributen abgelegt sind, sind vom Typ NODE_TEXT. Für die Attribute gibt es den Typ NODE_ATTRIBUTE. Für den XML-Prolog und alle sonstigen Bestandteile, die in <? ...?> eingefaßt sind, gibt es den Typ NODE_PROCESSING_INSTRUCTION. Kommentare werden in Knoten vom Typ NODE_COMMENT gehalten. Auch die übrigen Knotentypen sollen nun noch kurz vorgestellt werden:

Knotentyp	Erläuterung	Kann enthalten sein in folgenden Knotentypen	Kann selbst folgende Knotentypen enthalten
NODE_ELEMENT	Ein Element	Document, DocumentFragment, EntityReference, Element	Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
NODE_ATTRIBUTE	Ein Attribut eines Elements	Erscheint nicht als Kindknoten eines anderen Knotens, sondern ist in einer Auflistung vom Typ IXMLDOMNamendNodeMap enthalten	Text, EntityReference
NODE_TEXT	Textinhalt	Attribute, DocumentFragment, Element, EntityReference	keine
NODE_CDATA_SECTION	CDATA-Abschnitt	DocumentFragment, EntityReference, Element	keine

Tabelle 5.2 Die zwölf Knotentypen des DOM

Knotentyp	Erläuterung	Kann enthalten sein in folgenden Knotentypen	Kann selbst folgende Knotentypen enthalten
NODE_ ENTITY_ REFERENCE	Entity Reference	Attribute, DocumentFragment, Element, EntityReference	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
NODE_ ENTITY	Enthält eine expanded entity	DocumentType	Text, EntityReference
NODE_ PROCESSING_ INSTRUCTION	Processing Instruction	Document, DocumentFragment, Element, EntityReference	Keine
NODE_ COMMENT	Kommentar	Document, DocumentFragment, Element, EntityReference	Keine
NODE_ DOCUMENT	Enthält das gesamte XML-Dokument	Keine, muß stets auf der obersten Ebene angesiedelt sein	Element (maximal eins), ProcessingInstruction, Comment, DocumentType
NODE_ DOCUMENT_TYPE	document type declaration, indicated by the <!DOCTYPE > tag	Document	Notation, Entity
NODE_ DOCUMENT_FRAGMENT	Fragment eines XML-Dokumentes	Keine	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
NODE_ NOTATION	Eine Notation in der Document Type Definition	DocumentType	keine

Tabelle 5.2 Die zwölf Knotentypen des DOM (Forts.)

5.6.2 NODE_DOCUMENT_FRAGMENT

Wenn ein Knoten dieses Typs mit einer Methode wie `insertBefore` oder `appendChild` an einen anderen Knoten angehängt wird, verhält er sich anders als die übrigen Knoten. Es wird nicht der Fragmentknoten selbst angehängt, sondern die Kinder des Fragmentknotens werden umgehängt. Auf diese Weise lassen sich bequem eine Reihe von Geschwisterknoten erzeugen und anschließend umhängen.

Nehmen wir wieder ein Beispiel. In folgendem XML-Dokument soll ein weiteres Produkt angefügt werden (CD-LW:\DOM\neuesProdukt.xml):

```
<?xml version="1.0" encoding="iso8859-1"?>
<Kaufhaus>
   <Abteilung Name="Bekleidung">
      <Produkt>
         <Name>Mütze</Name>
         <Farbe>weinrot</Farbe>
         <Preis Währung="DM">50</Preis>
      </Produkt>
   </Abteilung>
</Kaufhaus>
```

Hier ist der entsprechende VB-Code (CD-LW:\DOM\modDOM.bas):

```
Sub neuesProdukt()
Dim xmlDoc As New DOMDocument
Dim XPATHAusdruck As String
Dim fragmenthenkel As IXMLDOMDocumentFragment
Dim neuProd As IXMLDOMElement
Dim eTmp As IXMLDOMElement
Dim txtNode As IXMLDOMText
Dim ergebnisliste As IXMLDOMNodeList
   xmlDoc.async = False
   xmlDoc.Load "D:\DOM\neues Produkt.xml"
   If (xmlDoc.parseError <> 0) Then
      If xmlDoc.parseError.reason <> "" Then
         Debug.Print xmlDoc.parseError.reason
         Debug.Print xmlDoc.parseError.Line
      End If
      Exit Sub
   End If
' Folgendes Produkt soll hinzugefügt werden:
'     <Produkt>
'       <Name>Jacke</Name>
'       <Farbe>blau</Farbe>
'       <Preis Währung="DM">150</Preis>
'     </Produkt>
' Die Elemente "Name", "Farbe" und "Preis" werden zunächst
' an einem Fragmenthenkel zusammengetragen und dann vom
' Fragmenthenkel an den Produkthenkel umgehängt.
```

```
   Set fragmenthenkel = xmlDoc.createDocumentFragment
   ' "Name"-Element erzeugen
   Set eTmp = xmlDoc.createElement("Name")
   Set txtNode = xmlDoc.createTextNode("Jacke")
   eTmp.appendChild txtNode
   fragmenthenkel.appendChild eTmp
   ' "Farbe"-Element erzeugen
   Set eTmp = xmlDoc.createElement("Farbe")
   Set txtNode = xmlDoc.createTextNode("blau")
   eTmp.appendChild txtNode
   fragmenthenkel.appendChild eTmp
   ' Preis-Element mit Attribut erzeugen
   Set eTmp = xmlDoc.createElement("Preis")
   eTmp.setAttribute "Währung", "DM"
   Set txtNode = xmlDoc.createTextNode("150")
   eTmp.appendChild txtNode
   fragmenthenkel.appendChild eTmp
   ' Produkt-Element erzeugen
   Set neuProd = xmlDoc.createElement("Produkt")
   ' Elemente "umhängen"
   neuProd.appendChild fragmenthenkel
   ' Abteilungs-Element heraussuchen
   XPATHAusdruck = "//Abteilung[@Name='Bekleidung']"
   xmlDoc.setProperty "SelectionLanguage", "XPath"
   Set ergebnisliste = xmlDoc.selectnodes(XPATHAusdruck)
   ' Neues Produkt hinzufügen
   ergebnisliste.Item(0).appendChild neuProd
   ' Testausgabe
   Debug.Print vbCrLf & "-------------------------"
   Debug.Print xmlDoc.xml
End Sub
```

Um einen Knoten vom Typ NODE_DOCUMENT_FRAGMENT zu erzeugen, stellt das `DOMDocument`-Objekt die Methode `createDocumentFragment` zur Verfügung. Die Ausgabe zeigt das gewünschte Ergebnis:

```
<?xml version="1.0"?>
<Kaufhaus>
   <Abteilung Name="Bekleidung">
      <Produkt>
         <Name>Mütze</Name>
         <Farbe>weinrot</Farbe>
```

```
        <Preis Währung="DM">50</Preis>
    </Produkt>
    <Produkt><Name>Jacke</Name><Farbe>blau</Farbe><Preis
Währung="DM">150</Preis></Produkt></Abteilung>
</Kaufhaus>
```

Bei der Ausgabe fehlen lediglich die Einrückungen.

5.6.3 NODE_CDATA_SECTION

Dieser Knotentyp enthält einen CDATA-Abschnitt. Auf diese Weise lassen sich Textblöcke markieren, die Sonderzeichen wie < oder & enthalten, ohne daß diese Sonderzeichen als Markup interpretiert werden. Ein Beispiel:

```
<Formel>Das: <![CDATA[3 < 4]]> ist eine Formel.</Formel>
```

In einem VB-Programm referenziere die Variable `child` das Element `Formel`. Die Variable `child` verfügt dann über drei Kindknoten: Der erste Textknoten enthält »Das: «. Der zweite Kindknoten ist vom Typ `NODE_CDATA_SECTION` und enthält den Text »3 < 4«. Der dritte Knoten ist wieder vom Typ NODE_TEXT und enthält den Text » ist eine Formel.«.

In VB erhält man ein Objekt vom Typ NODE_CDATA_SECTION über die `createCDATASection`-Methode eines `DOMDocument`-Objekts:

```
Dim myCDATA As IXMLDOMCDATASection
Dim xmlDoc As New DOMDocument
...
Set myCDATA = xmlDoc.createCDATASection
```

5.6.4 NODE_ENTITY_REFERENCE

Zwei Arten von Entity References sind voneinander zu unterscheiden: die vordefinierten und die selbst definierten. (Tatsächlich gibt es noch viel mehr Arten von Entity References. An dieser Stelle reicht das Verständnis dieser beiden Arten aus.)

In den XML-Parser eingebaut ist das Verständnis für fünf vordefinierte und für numerische Entitäten, die in Tabelle 5.3 aufgeführt sind.

Entität	Zeichen
&lt;	<
&gt;	>
&amp;	&
&apos;	'
&quot;	«
&#x89AB;	Hexadezimalwert eines beliebigen Unicode-Zeichens
&#8901;	Dezimalwert eines beliebigen Unicode-Zeichens

Tabelle 5.3 Die vordefinierten Entitäten versteht jeder XML-Parser

Daneben können in einer DTD eigene Entitäten definiert werden. Damit lassen sich beispielsweise Textkonstanten abkürzen. Diese beide Arten werden unterschiedlich behandelt.

Die vordefinierten Entitäten werden nicht in eigene Knoten gefaßt, sondern gehen in einem Textknoten auf. Der Inhalt `3 < 4` würde also nicht als Folge von »Textknoten, Entity-Reference-Knoten, Textknoten« repräsentiert, sondern in einen einzelnen Textknoten gefaßt.

Anders verhält es sich bei den selbstdefinierten Entitäten. Diese werden tatsächlich in einem eigenen Knoten vom Typ Entity Reference gespeichert. Das folgende Beispiel definiert in der vorangestellten DTD eine Entität namens `myhomepage`, die im Dokument mit `&myhomepage;` referenziert wird.

```
<?xml version="1.0"?>
<!DOCTYPE Kaufhaus [
<!ENTITY myhomepage "http://www.myhomepage.de" >
<!ELEMENT Kaufhaus (#PCDATA) >
]>
<Kaufhaus>Sehen Sie selbst: &myhomepage;</Kaufhaus>
```

Wenn in einem VB-Programm die Variable `child` das Element `Kaufhaus` referenziert, dann ergibt sich das folgende Bild. Der Kommentar führt jeweils die Ausgabe an, die von Debug.Print erzeugt wird:

```
Debug.Print child.childNodes(0).xml ' "Sehen Sie selbst:"
Debug.Print child.childNodes(1).xml ' "&myhomepage;"
Debug.Print child.childNodes(1).nodeTypeString
' "entityreference"
```

```
Debug.Print child.childNodes(1).Text
' "http://www.myhomepage.de"
```

Anders sieht es beim Einsatz der vordefinierten Entity References aus. Wenn im XML-Dokument folgender Inhalt auftaucht:

```
<Kaufhaus>Sehen Sie selbst: 3 &lt; 4</Kaufhaus>
```

ergibt sich folgendes Bild:

```
Debug.Print child.childNodes(0).xml
' "Sehen Sie selbst: 3 &lt; 4"
Debug.Print child.childNodes(0).nodeTypeString
' "text"
Debug.Print child.childNodes(0).Text
' "Sehen Sie selbst: 3 < 4"
```

Vordefinierte Entity References werden also nicht als eigenständige Knoten abgespeichert, sondern gehen im umgebenden Textknoten auf.

5.6.5 NODE_DOCUMENT_TYPE

Ein Knoten von diesem Typ enthält die Document Type Definition (DTD). Das ist jener Bestandteil am Anfang eines XML-Dokuments, der in den `<!DOCTYPE ... >`-Tag gefaßt ist. Diese Angabe kann nur auf oberster Ebene erfolgen. Anders gesagt: Ein Knoten vom Typ NODE_DOCUMENT_TYPE muß ein Kindknoten vom NODE_DOCUMENT-Knoten sein.

5.6.6 NODE_NOTATION

Notationen sind spezielle Konstrukte in der Document Type Definition. Sie legen den Umgang mit Nicht-XML-Daten fest, wie etwa Grafiken oder Sound-Dateien. Nur zur Illustration folgendes Beispiel:

```
<!NOTATION avi SYSTEM 'mplayer.exe' >
```

Da Notationen nur innerhalb von DTDs vorkommen können, muß also ein entsprechender Knoten ein Kindknoten eines NODE_DOCUMENT_TYPE-Knotens sein.

5.6.7 NODE_ENTITY

In einer Document Type Definition können verschiedenen Arten von Entitäten definiert werden. Diese werden dann in einem Knoten vom Typ »NODE_ENTITY« abgelegt. Folgendes Beispiel als Illustration:

```
<!ENTITY myhomepage "http://www.mlohrer.de">
```

Knoten von diesem Typ können ebenfalls nur in Knoten vom Typ NODE_DOCUMENT_TYPE enthalten sein.

5.7 Ein XML-Dokument mit VB erstellen

5.7.1 Knoten zum Baum verknüpfen

Im Crashkurs-Kapitel ganz am Anfang des Buches wurde kurz angedeutet, wie man mit VB von Grund auf ein neues XML-Dokument erstellt. Mit ausführlicheren Erläuterungen soll das folgende XML-Dokument von Grund auf erstellt werden:

```
<?xml version="1.0" encoding="iso8859-1"?>
<Kaufhaus>
  <Produkt>
    <Name>Mütze</Name>
    <Preis Währung="Euro">50
      <!--DM-Preise nur noch auf Anfrage-->
    </Preis>
    <Formel><![CDATA[3 < 4]]></Formel>
  </Produkt>
</Kaufhaus>
```

Die Arbeit verrichtet diese VB-Prozedur, die auf der CD-ROM in CD-LW:\DOM\modDOM.bas enthalten ist.

```
Sub neuesXMLDokument()
Dim xmlDoc As New DOMDocument
Dim pi As IXMLDOMProcessingInstruction
Dim elem1 As IXMLDOMElement
Dim elem2 As IXMLDOMElement
Dim text As IXMLDOMText
Dim comment As IXMLDOMComment
Dim cdata As IXMLDOMCDATASection
Dim dateiname As String
   dateiname = "L:\galileo\xml-Buch\Kap-4-DOM-neuesDoku.xml"
   Set pi = xmlDoc.createProcessingInstruction("xml",
   "version='1.0' encoding='iso8859-1'")
   xmlDoc.appendChild pi
   Set elem1 = xmlDoc.createElement("Kaufhaus")
   xmlDoc.appendChild elem1
   Set elem2 = xmlDoc.createElement("Produkt")
   elem1.appendChild elem2
```

```
    Set elem1 = xmlDoc.createElement("Name")
    elem2.appendChild elem1
    Set text = xmlDoc.createTextNode("Mütze")
    elem1.appendChild text
    Set elem1 = xmlDoc.createElement("Preis")
    elem2.appendChild elem1
    Set text = xmlDoc.createTextNode("50")
    elem1.appendChild text
    elem1.setAttribute "Währung", "Euro"
    Set comment = xmlDoc.createComment("DM-Preise nur noch
    auf Anfrage")
    elem1.appendChild comment
    Set elem1 = xmlDoc.createElement("Formel")
    elem2.appendChild elem1
    Set cdata = xmlDoc.createCDATASection("3 < 4")
    elem1.appendChild cdata
    xmlDoc.Save dateiname
End Sub
```

Die Vorgehensweise ist im Prinzip einfach. Die Anweisung

```
Dim xmlDoc As New DOMDocument
```

deklariert und erzeugt eine Instanz eines `DOMDocument`-Objekts. Die Anweisung

```
Set elem1 = xmlDoc.createElement("Kaufhaus")
```

erzeugt ein Element mit dem Namen »Kaufhaus«. Auch für die übrigen Elemente eines XML-Dokumentes – Kommentare, CDATA-Abschnitte, Textknoten etc. – stellt das `DOMDocument`-Objekt jeweils eine entsprechende Methode zur Verfügung. Die Prozedur verwendet `createTextNode`, `createCDATASection`, `createProcessingInstruction` und `createComment`.

Die Methode `papa.appendChild baby` hängt den Knoten `baby` als Kindknoten an den Knoten `papa` an. Diese Methode funktioniert für alle Knotentypen, solange der Elternknoten den anzufügenden Knoten enthalten darf (siehe hierzu auch in der Tabelle 5.2 die Spalte »Kann selbst folgende Knotentypen enthalten«). Die `appendChild`-Methode nimmt auf den Elternknoten Bezug. Beim Einfügen eines Knotens in die Baumstruktur ist es alternativ möglich, sich mit der `insertBefore`-Methode auf einen Geschwisterknoten zu beziehen. `schwester.insertBefore bruder` fügt (links) vor die Schwester einen `bruder`-Knoten ein. Die Benennung der Objekte soll das Gemeinte verdeutlichen.

Das Beispiel-Dokument enthält fünf Elemente, die in der Prozedur mit Hilfe von nur zwei Variablen des Typs `IXMLDOMElement` erzeugt werden. Nachdem ein Element in den Baum eingehängt wurde, kann es später über die Baumstruktur wiedergefunden werden, so daß die Variable anschließend für ein neues Element-Objekt zur Verfügung steht. Im Prinzip sind zwei Variablen stets ausreichend. Die eine zeigt auf den gerade neu erzeugten Knoten und die andere auf einen Knoten im gesamten Baum. Man muß sich nur gut merken, welche Variable gerade wohin zeigt, und manchmal kann es natürlich ganz praktisch sein, ein paar mehr Variablen zur Verfügung zu haben.

Ein Element erhält über die `setAttribute`-Methode ein Attribut:

```
elem1.setAttribute "Währung", "Euro"
```

Mit

```
xmlDoc.Save dateiname
```

wird die Datei gespeichert.

5.7.2 XML-Daten laden und speichern

Eine XML-Datei kann mit der `Load`-Methode in ein `DOMDocument`-Objekt geladen und mit `Save` in eine Datei gespeichert werden. Das Laden erfolgt per Default-Einstellungen im asynchronen Modus. Das bedeutet, daß mit der Programmausführung fortgefahren wird, noch bevor der möglicherweise langwierige Ladevorgang beendet ist. Um erst den Abschluß des Ladevorgangs abzuwarten, kann die `async`-Eigenschaft auf `false` gesetzt werden. Eine typische Sequenz hat diese Form:

```
xmlDoc.async = False
xmlDoc.Load "c:\test.xml"
```

Der `Load`-Methode kann als Argument ein Pfadname, eine URL oder in ASP-Skripten auch das `Request`-Objekt übergeben werden. Ein asynchroner Download kann mit der `abort`-Methode des `DOMDocument`-Objekts abgebrochen werden.

Auch die `Save`-Methode ist entsprechend vielseitig. Als Argument kann ihr entweder ein Dateiname mit Pfadangabe übergeben werden, bei ASP-Skripten das `Response`-Objekt, ein anderes `DOMDocument`-Objekt oder ein weiteres Objekt, das Persistenz unterstützt.

Für die Verarbeitung innerhalb von Skripten stehen die Methode `loadXML` und die `xml`-Eigenschaft zur Verfügung. `loadXML` erhält als Argument einen String, der entweder ein komplettes XML-Dokument oder ein wohlgeformtes XML-Fragment enthält. Alle Knoten verfügen außerdem über die `xml`-Eigenschaft, die den Inhalt des Knotens als XML-String zurückgibt.

5.8 Von Ast zu Ast: Sich im XML-Baum bewegen

Nehmen wir einmal an, wir wären ein Zeiger auf einen Knoten in einem XML-Baum. Als solcher kennen wir den Knoten, auf den wir gerade zeigen, ganz gut, aber nun packt uns die Neugier, und wir wollen den ganzen XML-Baum kennenlernen. Im Prinzip können wir dann immer vier Wege einschlagen. Diese vier Wege sind:

- nach oben zum Elternknoten
- nach unten zu unseren eigenen Kindknoten
- nach links zu den vor uns stehenden Geschwisterknoten
- nach rechts zu den hinter uns stehenden Geschwisterknoten

5.8.1 Zum Elternknoten

Zunächst versuchen wir es beim Elternknoten.

```
set elternknoten = ichknoten.parentNode
```

Zeigt `elternknoten` jetzt stets auf den Elternknoten? Meistens schon. Wenn wir aber zufällig gerade das Wurzelelement sind, das selbst keinen Elternknoten mehr hat, dann nicht. In VB würde folgender Code diese Situation erzeugen:

```
xmlDoc.async = False
xmlDoc.Load "D:\DOM\kaufhaus.xml"
Set node = xmlDoc.parentNode ' node = nothing
If node Is Nothing Then
   MsgBox "Kein Elternknoten vorhanden."
End If
```

Auch ein Objekt vom Typ `DocumentFragment` und ein Attributknoten haben keinen Elternknoten. Ein frisch erzeugter Elementknoten, der noch in keine Baumstruktur eingefügt wurde, hat natürlich ebenfalls noch keinen Elternknoten (hier hört die Analogie zur Welt der Säugetiere definitiv auf).

5.8.2 Zu den Kindknoten

So testen wir, ob wir über Kindknoten verfügen:

```
Debug.Print xmlDoc.hasChildNodes
```

In der Regel wird das Dokument selbst einige Kindknoten enthalten, sonst wäre es ein leeres Dokument. Also erhalten wir hier `Wahr` als Antwort. Wie viele Kinder sind es?

```
Debug.Print xmlDoc.childNodes.Length
```

nennt die Antwort: 2. Also wollen wir das erste Kind näher kennenlernen.

```
Set node = xmlDoc.childNodes(0)
```

Alternativ funktioniert auch

```
Set node = xmlDoc.firstChild
```

Den letzten Kindknoten erhalten wir mit

```
Set node = xmlDoc.lastChild
```

Die `childNodes`-Methode gibt ein Objekt vom Typ `IXMLDOMNodeList` zurück.

5.8.3 Zu den Geschwisterknoten

Zwei Methoden stehen zur Verfügung, um die Geschwisterknoten zu besuchen: `nextSibling` und `previousSibling`. Das Beispiel zeigt die Verwendung:

```
Set nextnode = node.previousSibling
If nextnode Is Nothing Then
   MsgBox "kein vorhergehender Geschwisterknoten vorhanden."
End If
```

5.8.4 Den gesamten Baum durchsuchen

Unter Umständen kann es zu mühsam sein, ausgehend von einem einzelnen Knoten der Reihe nach alle anderen Knoten eines Baums »abzuklappern«. Wenn wir gezielt auf eine ganze Reihe von Knoten zugreifen wollen oder einen ganz bestimmten Knoten suchen, dann stehen einige spezielle Methoden zur Verfügung.

- `getElementsByTagName` listet alle Elemente eines Typs auf.

- `selectNodes` wertet einen XPath-Ausdruck aus und liefert eine Ergebnisliste.
- `selectSingleNode` wertet ebenfalls einen XPath-Ausdruck aus und liefert einen einzelnen Knoten.

getElements ByTagName

Die Methode `getElementsByTagName` steht dem Dokumentknoten und allen Elementknoten zur Verfügung. Die Methode versammelt alle Elemente eines bestimmten Typs, die unter den »Nachfahren« des Ausgangsknotens auftauchen. Wenn man beispielsweise alle Produkte bearbeiten möchte, die im XML-Kaufhaus angeboten werden, könnte die Anweisung lauten:

```
Dim produktliste As IXMLDOMNodeList
...
Set produktliste = xmlDoc.getElementsByTagName("Produkt")
```

Wenn die Methode nicht vom Dokumentknoten aus aufgerufen wird, sondern von einem Elementknoten, dann werden nicht alle Elemente des gesamten Dokuments gefunden, sondern nur die jeweiligen Nachfahren. Auf diese Weise lassen sich beispielsweise alle Produkte einer Abteilung zusammentragen:

```
Set produktliste = abteilung.getElementsByTagName("Produkt")
```

selectSingle Node

Man muß nur vorher dafür gesorgt haben, daß die Variable `abteilung` auf den richtigen Knoten zeigt. Dafür kann etwa die folgende `selectSingleNode`-Methode nützlich sein:

```
xmlDoc.setProperty "SelectionLanguage", "XPath"
Set start = xmlDoc.selectSingleNode("//
Abteilung[@Name='Bekleidung']")
```

Der `selectSingleNode`-Methode wird als Argument ein XPath-Ausdruck übergeben. Aus Gründen der Rückwärtskompatibilität wird außerdem eine Syntax namens XSLPattern unterstützt, die die Default-Einstellung bildet. Um ausdrücklich XPath zu wählen, muß dieser Standard zunächst mit der Zeile

```
xmlDoc.setProperty "SelectionLanguage", "XPath"
```

eingestellt werden. Bleibt noch der XPath-Ausdruck `//Abteilung[@Name='Bekleidung']`") zu klären. Er bedeutet hier so viel wie »Finde ein Element vom Typ Abteilung, dessen Attribut Name den Wert 'Bekleidung' hat«. XPath bildet eine kleine Welt für sich. In Kapitel 6 wird es ausführlich vorgestellt.

5.9 Die Knotenstruktur eines XML-Dokuments ausgeben

Die Online-Dokumentation zum msxml 3.0 enthält ein instruktives Beispiel für eine ASP-Seite, die ein XML-Dokument einliest und die Knoten-Struktur des Dokuments in ähnlicher Form ausgibt, wie es am Anfang des Kapitels gezeigt wurde. Dieses Beispiel haben wir im folgenden so angepaßt, daß es innerhalb von VB abläuft und das Ergebnis am Schluß mit `debug.print` ausgibt. Es ist recht aufschlußreich, einige Minuten (noch besser sind einige Stunden) mit dem Nachvollzug der drei Prozeduren zu verbringen. Im Anschluß an den Code werden die drei Prozeduren erläutert (CD-LW:\DOM\modDOM.bas).

```
Sub attribute_walk(node, indent, ausgabe)
Dim i As Integer
Dim attrib As IXMLDOMAttribute
Dim attribList As IXMLDOMNamedNodeMap
  For i = 1 To indent
    ausgabe = ausgabe & (" ")
  Next
  Set attribList = node.Attributes
  For Each attrib In attribList
   ausgabe = ausgabe & ("|--")
   ausgabe = ausgabe & (attrib.nodeTypeString)
   ausgabe = ausgabe & (":")
   ausgabe = ausgabe & (attrib.Name)
   ausgabe = ausgabe & ("--")
   ausgabe = ausgabe & (attrib.nodeValue)
   ausgabe = ausgabe & vbCrLf
  Next
End Sub

Sub tree_walk(node, indent, ausgabe)
Dim nodeName As String
Dim nodeList As IXMLDOMNodeList
Dim child As IXMLDOMNode
Dim i As Integer
   indent = indent + 2
   Set nodeList = node.childNodes
   For Each child In nodeList
     For i = 1 To indent
       ausgabe = ausgabe & (" ")
```

```
      Next
      ausgabe = ausgabe & "|--" & child.nodeTypeString & "--"
      If child.nodeType = NODE_ELEMENT Then
        ausgabe = ausgabe & (child.nodeName)
        ausgabe = ausgabe & vbCrLf
        If (child.Attributes.Length > 0) Then
          indent = indent + 2
          attribute_walk child, indent, ausgabe
          indent = indent - 2
        End If
      End If
      If child.hasChildNodes Then
        tree_walk child, indent, ausgabe
      Else
        ausgabe = ausgabe & child.Text
        ausgabe = ausgabe & vbCrLf
      End If
   Next
   indent = indent - 2
End Sub

Sub ZeigeKnoten()
Dim xmlFile As String
Dim xmlDoc As DOMDocument
Dim indent As Integer
Dim ausgabe As String
   xmlFile = "D:\DOM\kaufhaus.xml"
   indent = 0
   ausgabe = ""
   Set xmlDoc = New DOMDocument
   xmlDoc.validateOnParse = False
   xmlDoc.async = False
   xmlDoc.Load xmlFile
   If xmlDoc.parseError.errorCode = 0 Then
     ausgabe = ausgabe & ("-------------------") & vbCrLf
     tree_walk xmlDoc, indent, ausgabe
     ausgabe = ausgabe & ("-------------------")
   Else
      If xmlDoc.parseError.reason <> "" Then
         ausgabe = ausgabe & xmlDoc.parseError.reason
         ausgabe = ausgabe & xmlDoc.parseError.Line
```

```
      End If
      Exit Sub
   End If
   Debug.Print ausgabe
End Sub
```

5.9.1 Die Prozedur ZeigeKnoten

Die Startprozedur ist `ZeigeKnoten`. Sie lädt das XML-Dokument und beginnt das rekursive Traversieren (siehe hierzu Kapitel 6) des XML-Dokuments durch den Aufruf von `tree_walk` und die Übergabe des Wurzelknotens. Die Variable `indent` enthält die aktuelle Einrücktiefe. In der Variablen `ausgabe` wird das Ergebnis zusammengestellt. Beide Variablen werden an `tree_walk` und `attribute_walk` als Referenzparameter übergeben. Der Inhalt von `ausgabe` kann auf diese Weise kontinuierlich anwachsen und auch `indent` kann sich dynamisch anpassen. Wenn die Bearbeitung abgeschlossen ist, wird das Ergebnis mit `Debug.Print ausgabe` ausgegeben.

Falls es beim Einlesen des XML-Dokuments zu einem Fehler gekommen sein sollte (`xmlDoc.parseError.errorCode <> 0`), wird die Fehlerursache mit `xmlDoc.parseError.reason` und die Zeilenposition, an der der Fehler aufgetreten ist, mit `xmlDoc.parseError.Line` ausgegeben.

5.9.2 Die Prozedur tree_walk

Beim Eintritt in die Prozedur wird zunächst die Variable `indent` um den Wert 2 erhöht, damit die Einrücktiefe stimmt.

Eine Liste aller Kindknoten des aktuellen Knotens vom Typ `IXMLDOMNodeList` erhalten wir mit der Anweisung:

```
Set nodeList = node.childNodes
```

Anschließend bearbeitet die Prozedur mit

```
For Each child In nodeList
```

Der Wurzelknoten enthält das komplette Dokument in Form von Kindknoten

alle enthaltenen Knoten. Manch einer fragt sich jetzt vielleicht: Ja, aber wo bleibt denn der Knoten selbst? Fehlt nicht der Elternknoten, wenn nur die Kindknoten bearbeitet werden? Hier kommt es darauf an, mit welchem Knoten die Prozedur ganz am Anfang aufgerufen wird. `ZeigeKnoten` ruft `tree_walk` mit dem Wurzelknoten auf. Der Wurzelknoten ist vom Typ `NODE_DOCUMENT` (9) und enthält das gesamte XML-Dokument in Form von Kindknoten. Anders herum wird es noch deutlicher: Alle Kindknoten des

Wurzelknotens machen zusammengenommen das komplette XML-Dokument aus. Also wird das XML-Dokument auch komplett bearbeitet.

`tree_walk` gibt so viele Leerzeichen aus, wie in der Variablen `indent` vermerkt sind, um in der Ausgabe die Einrücktiefe korrekt darzustellen. Danach wird mit Hilfe der Eigenschaft `child.nodeTypeString` der Typ des aktuellen Knotens ausgegeben.

Wenn es sich bei dem Knoten um ein Element handelt (`If child.nodeType = NODE_ELEMENT`), wird der jeweilige Name ausgegeben. Der Name des Elements

```
<Farbe>schwarz</Farbe>
```

wäre beispielsweise `Farbe`. Anschließend wird mit `If child.Attributes.Length > 0` geprüft, ob das Element über Attribute verfügt. Wenn das Element über Attribute verfügt, wird `attribute_walk` mit dem aktuellen Knoten aufgerufen.

Wenn der aktuelle Knoten über weitere Kindknoten verfügt (`If child.hasChildNodes`) wird `tree_walk` mit dem aktuellen Knoten rekursiv aufgerufen.

Wenn der aktuelle Knoten nicht über weitere Kindknoten verfügt, wird mit `child.Text` der Inhalt dieses Knotens ausgegeben.

`tree_walk` ruft sich selbst auf. Der erste Aufruf von `tree_walk`, der noch in der Prozedur `ZeigeKnoten` erfolgt, kehrt erst dann zurück, wenn das gesamte Dokument bearbeitet wurde.

5.9.3 Die Prozedur attribute_walk

Auch `attribute_walk` gibt zunächst Leerschritte für die richtige Einrücktiefe aus.

Die `Attributes`-Eigenschaft eines Elementknotens gibt ein Objekt vom Typ `IXMLDOMNamedNodeMap` zurück, das eine Liste aller Attribute enthält. Der Deutlichkeit halber wird diese Liste zunächst mit der Variablen `attribList` referenziert (`Set attribList = node.Attributes`). Anschließend bearbeitet die Schleife

```
For Each attrib In attribList
```

die enthaltenen Attribute. Die Eigenschaft `attrib.nodeTypeString` gibt jeweils »`attribute`« zurück. `attrib.Name` enthält den Attributnamen, zum Beispiel »`Währung`«, und mit `attrib.nodeValue` greifen wir auf den Inhalt zu, beispielsweise »`Euro`«.