

Eine Einführung in XML

Karlheinz Zeiner

Inhalt

Eine Einführung in XML	1
1 XML-Grundlagen	1
1.1 Einfaches XML-File erzeugen	1
1.2 XML Syntax	1
2 XML Validierung und Schemas	3
2.1 "Wohl geformte" (" <i>Well Formed</i> ") XML Dokumente	3
2.2 "Gültige" XML Dokumente	3
2.3 XML Schemas	3
2.4 Validieren von XML-Files	5
3 XML Files anzeigen	6
3.1 Textform eines XML Files	6
3.2 CSS Style-Sheet	6
3.3 XSL	7
4 XPath	7
5 XML mit C# und .NET	9
5.1 XML Dateien erzeugen und lesen	9
5.1.1 XmlTextWriter (.NET 1.1)	9
5.1.2 XmlTextReader	10
5.1.3 Lesen über XmlDocument (DOM)	11
5.2 Validierung im Programm	13
6 XML in einer beliebigen Programmierumgebung	14
7 XSLT	15
7.1 Transformation eines XML Dokuments in ein Textfile.	16
7.2 XML Dokument in ein HTML File transformieren	18
7.3 XML nach XML	20
7.4 XSLT im Programm	20

Literatur, Links:

<http://www.w3schools.com/default.asp>

XSLT, Doug Tidwell, O'Reilly

Professional XML, 2nd Edition, WROX

1 XML-Grundlagen

1.1 Einfaches XML-File erzeugen

XML-Files können mit einem Editor wie Notepad, speziellen XML-Editoren oder XML-Werkzeugen (wie z.B. XML-Spy) erstellt werden. Auch der Visual Studio Editor ist zur nArbeit mit XML-Files gut geeignet.

[New File](#) → [Categories: General](#) → [XML File](#).

Hinweis: Zum Kopieren von XML oder HTML Text über die Zwischenablage wählt man im Visual-Studio beim Einfügen "Paste as HTML".

Das folgende Beispiel zeigt ein XML-File:

```
<?xml version="1.0" encoding="utf-8"?>
<Figures>
  <Figure Id="F1">
    <Name>Circle1</Name>
    <Type>Circle</Type>
    <Size>20.5</Size>
  </Figure>
  <Figure Id="F2">
    <Name>Square</Name>
    <Type>Square</Type>
    <Size>20.5</Size>
  </Figure>
  <Figure Id="F3">
    <Name>Triangle</Name>
    <Type>Triangle</Type>
    <Size>34.6</Size>
  </Figure>
</Figures>
```

Der Aufbau erinnert an ein HTML-File. XML verwendet "Tags" um Elemente mit einer Anfangs- und Endmarke zu gruppieren. XML Editoren bieten meist komfortable Unterstützung bei der Erstellung, sei es durch automatische Vervollständigung oder durch spezielle Darstellungen der Struktur. Aber auch automatische Einrückungen machen die verschachtelte hierarchische Baumstruktur eines XML-Files sichtbar.

1.2 XML Syntax

XML-Tags sind nicht vordefiniert, man muss sie selber festlegen. Die Syntax für ein XML-File ist einfach.

Die erste Zeile im Dokument - die XML Deklaration - definiert die XML Version und die Codierung der Zeichen im File. Die nächste Zeile beschreibt das Wurzelement (*root-element*) des Dokuments. Jedes XML-Dokument muss genau ein Wurzelement enthalten. Dessen Name soll das Dokument charakterisieren, z.B. hier in der Form: "Dieses Dokument ist ein XRio Dokument". Die Elemente sind in einer Baumstruktur angelegt. Ein Element kann Kindelemente (child elements) enthalten. Block ist ein Kindelement des Wurzelements und die Parameter Elemente sind Kindelemente des Block-Elements.

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Verwendet man gute Namen für die XML-Tags, so ist ein XML File gut lesbar.

Einige wichtige Regeln:

- Alle XML Elemente müssen mit einem schließenden Tag (closing tag) abgeschlossen sein.
- XML-Tags sind groß-klein sensitiv.
- Alle XML Elemente müssen richtig verschachtelt sein.
- Ein XML Dokument darf nur einen Wurzelknoten haben.

Elemente haben einen Inhalt

Elemente können unterschiedlichen Inhalt haben. Zu einem XML Element gehört alles vom Start-Tag bis zum Ende-Tag ((einschließlich der Tags). Ein Element kann weitere Elemente, einfachen Inhalt oder gemischten Inhalt enthalten oder kann auch leer sein. Ein Element kann Attribute enthalten. Die Id's von Block und Parameter sind Attribute. Attribute müssen apostrophiert (*quoted*) sein. Es gibt keine Regel, ob Daten eines Elements als Subelemente oder als Attribute modelliert werden

Im Beispiel hat das Figure-Element ein weitere Elemente als Inhalt, Type, Name und Size hat einfachen Inhalts (simple content or text content) weil diese Elemente nur noch Text enthalten.

<Unit></Unit> oder <Unit/>

ist ein Beispiel für ein leeres Element.

Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <simplecontent>This is the content!</simplecontent>
  <mixedcontent>An example for mixed element content. First text
and then a nested element<nested_element>Content of nested
element</nested_element>
</mixedcontent>
  <emptycontent></emptycontent>
  <emptycontent/>
  <element at1="Attribute 1" at2="Attribute 2"></element>
  <!-- This is a comment -->
</root>
```

Elemente versus Attribute

Wie bereits erwähnt kann man Daten in Elementen oder Attributen unterbringen.

Beispiel:

```
<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>

<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

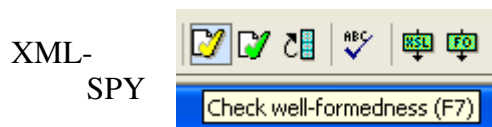
Im ersten Beispiel ist sex ein Attribut. Im zweiten ist sex ein Kind-Element. Die Information ist dieselbe.

Es gibt breite Diskussionen, aber keine Regel über die Verwendung von Elementen und/oder Attributen. Im Zweifelsfall entscheide ich mich eher für Elemente.

2 XML Validierung und Schemas

2.1 "Wohl geformte" ("Well Formed") XML Dokumente

Ein "wohl geformtes" XML Dokument erfüllt die XML Syntax. XML-Editoren beachten diese Syntax, bzw. erlauben eine Überprüfung der Syntax.



Wichtig: XML Applikationen brechen die Beareitung für ein nicht wohl geformtes XML Dokument ab. Dies ist in den W3C XML Spezifikationen so festgelegt: "A program should not continue to process an XML document if it is not well-formed. The reason is that XML software should be easy to write, and that all XML documents should be compatible."

2.2 "Gültige" XML Dokumente

Ein "gültiges" (*valid*) XML Dokument ist ein wohl geformtes XML Dokument, das zusätzlich einer Dokument Type Definition (DTD) oder einem Schema entspricht. Sowohl DTD's als auch Schemas definieren die erlaubte Struktur als auch die Daten der Elemente. Hier werden nur (die wichtigeren) Schemas besprochen.

XML Schemas werden ebenfalls in einem XML File definiert. Einfacher erstellt man Schemas jedoch mit einem entsprechenden Werkzeug.

2.3 XML Schemas

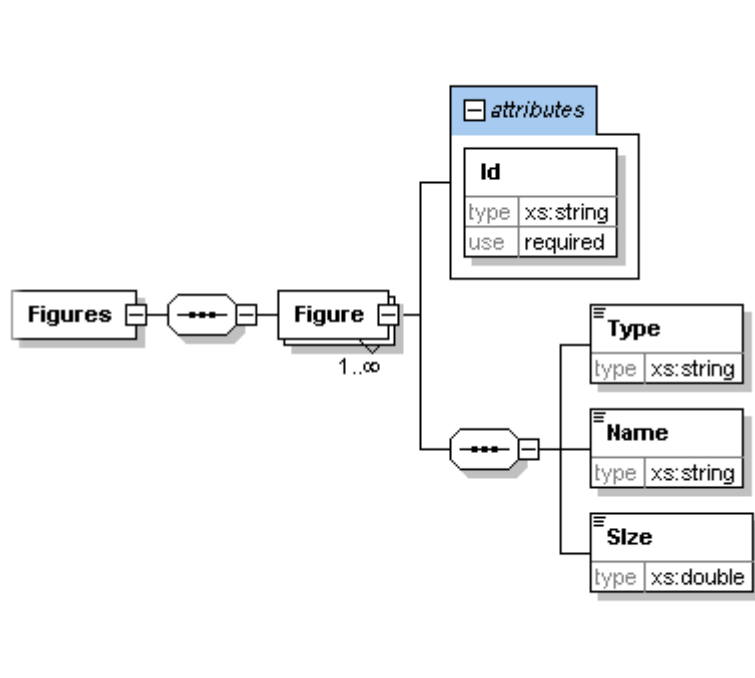
Ein Schema beschreibt die erlaubte Struktur eines XML Dokuments. Im Detail bedeutet das hauptsächlich:

- die Struktur des XML Dokumentes

- die Attribute eines Elements
- den Datentyp der Textknoten
- Ob Elemente oder Attribute vorhanden sein müssen oder optional sind

Beispiel:

Schema zu unserem Beispiel mit einer Liste von Figuren:



XML-Spy verwendet eine geschickte, gut lesbare Darstellung:

Der Wurzelknoten **Figures** enthält eine Sequenz aus **Figure** Elementen. Die Zahl der **Figure** Elemente ist mindestens 1 und nach oben unbegrenzt. Die Endknoten enthalten die eigentlichen Daten. Für den Text dieser Knoten sind einfache Datentypen definiert. Die Liste der Datentypen umfasst alle aus den Programmiersprachen bekannten Typen. Der Inhalt des Elements (InnerText) ist dann jedoch immer eine Zeichenkette. Der Datentyp `xs:double` z.B. erlaubt die üblichen Darstellungen eines Double-Wertes inklusive den Mustern INF, -INF und NaN für $+\infty$, $-\infty$ und "nicht definiert".

Man kann selber Datentypen definieren. Einfachen Typen basieren meist auf einem der vordefinierten Typen und definieren Einschränkungen zu diesem Typ. Schon unser einfaches Schema kann diese Möglichkeiten sinnvoll verwenden. Als **Id** sollen z.B. nur Wörter verwendet werden dürfen, die mit einem Buchstaben beginnen und dann nur noch Buchstaben und/oder Zahlen enthalten. Schemas sind selber auch wieder XML-Dateien. In Textform lautet die Definition des Typs **Id**:

```

<xs:simpleType name="tIdentifier">
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z]{1}[a-zA-Z0-9_]*"/>
  </xs:restriction>
</xs:simpleType>

```

Die Namen der Tags und die vordefinierten Datentypen sind hier inklusive einem Namensraum (Namespace) xs angegeben, der am Beginn des Dokuments definiert ist. Der voll qualifizierte Name eines Elements ist bei Verwendung eines Namensraumes namespace:name.

Der Typ tType definiert für den erlaubten Inhalt des Typ-Elements nur die in einer Aufzählung genannten Wörter.

```

<xs:simpleType name="tType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Circle"/>
    <xs:enumeration value="Square"/>
    <xs:enumeration value="Triangle"/>
  </xs:restriction>
</xs:simpleType>

```

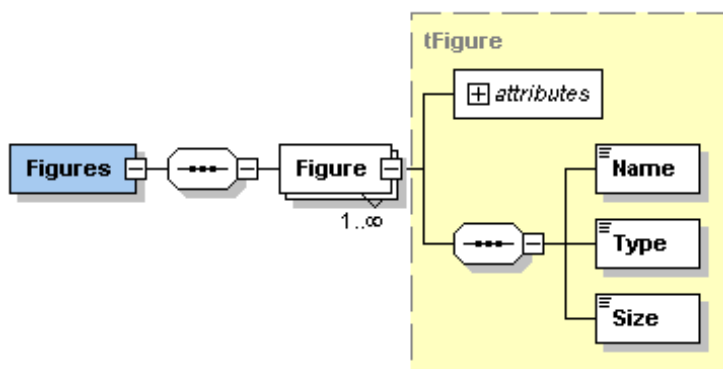
Für den Wert Size ist noch ein Typ definiert, der nur positive Werte zulässt.

```

<xs:simpleType name="tReal">
  <xs:restriction base="xs:double">
    <xs:minInclusive value="0"/>
  </xs:restriction>
</xs:simpleType>

```

Ein komplexer Datentyp erlaubt die Zusammenfassung von mehreren Knoten und Subknoten zu einem neuen Typ. Im folgenden Beispiel sind die Elemente für eine Figure als neuer Typ tFigure definiert.



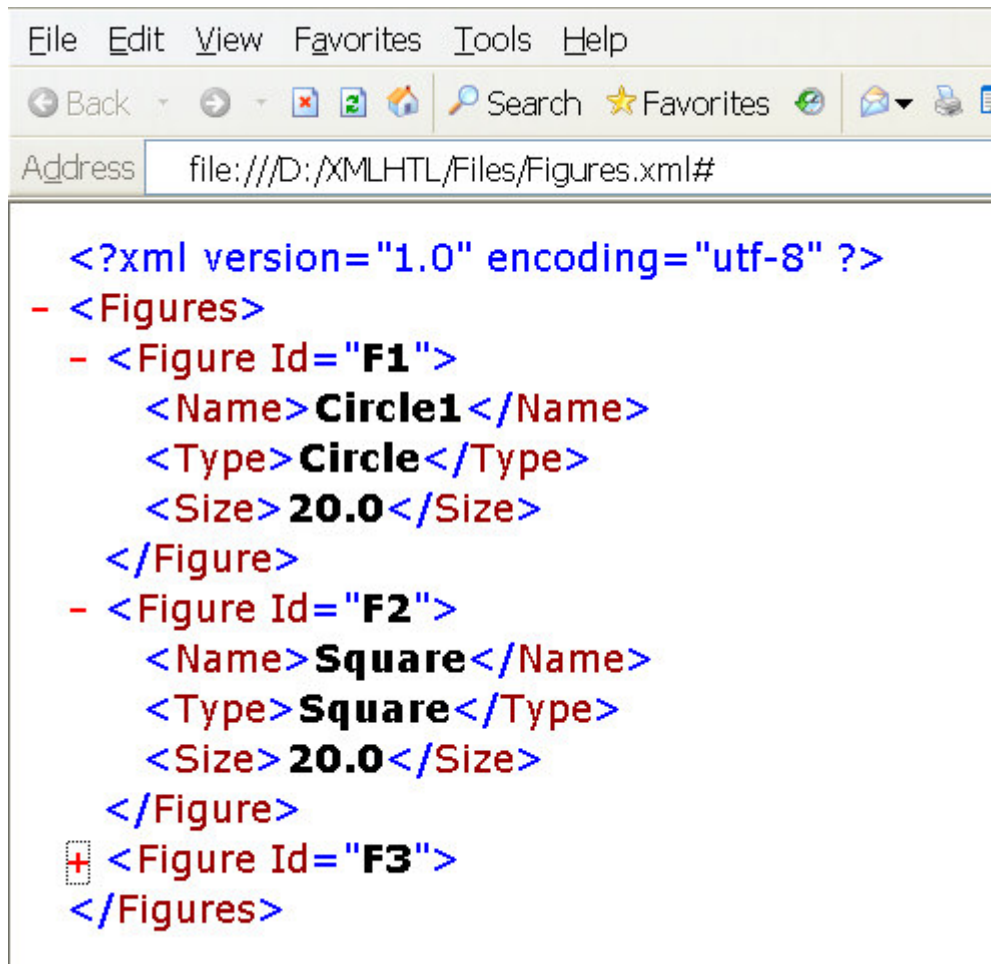
2.4 Validieren von XML-Files

Gute XML-Editoren können ein File gegen ein zugehöriges Schema validieren.

3 XML Files anzeigen

3.1 Textform eines XML Files

XML Dokumente können mit dem Internet-Explorer IE 5.0 (oder höher) oder einem anderen Browser angezeigt werden. Nach einer typischen Installation ist der IE für XML-Files zuständig. Das Dokument wird mit farbigen Tags und durch Einrückungen strukturiert angezeigt. Man kann Elemente expandieren oder kollabieren.



```
<?xml version="1.0" encoding="utf-8" ?>
- <Figures>
  - <Figure Id="F1">
    <Name>Circle1</Name>
    <Type>Circle</Type>
    <Size>20.0</Size>
  </Figure>
  - <Figure Id="F2">
    <Name>Square</Name>
    <Type>Square</Type>
    <Size>20.0</Size>
  </Figure>
  + <Figure Id="F3">
</Figures>
```

Will man das File editieren so geht das nur über Ansicht → Quelltext. Anschliessend Speichern nicht vergessen!

3.2 CSS Style-Sheet

Der "*Cascading Style Sheets*" Standard definiert eine Sprache zur Definition von Formateigenschaften für HTML oder XML Tags. Man kann CSS Style Sheets einem XML File zuordnen. Der Browser verwendet dann die Definitionen im Style Sheet um den Text-Inhalt des XML Files anzuzeigen. Ein Beispiel finden Sie unter

http://www.w3schools.com/xml/xml_display.asp

3.3 XSL

XSL ist die Abkürzung für eXtensible Stylesheet Language. Dieser Standard dient speziell der Darstellung von XML Dokumenten. XSL ist ein sehr mächtiges Konzept und kann verwendet werden um XML Dokumente in beliebige andere Dokumente umzuwandeln, z.B. in HTML-Files, PDF Files, SVG Files, Programmcode und vieles mehr.

XSL ist eine Entwicklung des World Wide Web Consortium (W3C).

4 XPath

XPath stellt eine Syntax zur Auswahl von Teilen eines XML-Dokuments dar. XPath ist für alle Techniken, die im Zusammenhang mit XML verwendet werden, eine wichtige Grundlage. XPath betrachtet ein XML-Dokument als einen Baum mit Knoten. Dieser Baum ist sehr ähnlich dem DOM-Baum. Man unterscheidet verschiedene Arten von Knoten:

- Wurzelknoten
- Elementknoten
- Attributknoten
- Textknoten
- Kommentarknoten
- Verarbeitungsanweisungen
- Namensraumknoten

Zur Auswahl von Dokumentteilen kann man absolute und relative Pfadangaben machen.

Beispiele für XPath Ausdrücke und der damit erfassten Knotenmenge für das Figures Dokument.

```
<?xml version="1.0" encoding="utf-8"?>
<figures>
  <figure Id="F1">
    <Name>Circle1</Name>
    <Type>Circle</Type>
    <Size>20.0</Size>
  </figure>
  <figure Id="F2">
    <Name>Square</Name>
    <Type>Square</Type>
    <Size>20.0</Size>
  </figure>
  <figure Id="F3">
    <Name>Triangle</Name>
    <Type>Triangle</Type>
    <Size>34.6</Size>
  </figure>
</figures>
```

XPath	Bemerkung	Knotenmenge
/	der Wurzelknoten Er enthält das gesamte Dokument. Er enthält das Element Figures und ist nicht das Element Figures selbst.	
/Figures		Figures
/Figures/*	Alle Kinder des Figures Knoten	Figure Id=F1 Figure Id=F2 Figure Id=F3
/Figures/*[@Id='F2']	Auswahl eines Kindelements mit einem bestimmten Wert des Id-Attributes	Figure Id=F2

Nun sei <Figures> das aktuell ausgewählte Element und die folgenden Angaben sind relative Pfade.

*[@Id='F2']	Auswahl eines Kindelements mit einem bestimmten Wert des Id-Attributes	Figure Id=F2
Figure[2]	Auswahl über einen Index (beginnt bei 1)	Figure Id=F2
Figure[2]/Name	der Elementknoten Name des zweiten Figure Knotens	Name
Figure[2]/Name/text()	Der Text des Elements	"Square"
.	der aktuelle Knoten	
..	der Elternknoten	

In den Beispielen wurde die abgekürzte Syntax verwendet. Die nicht abgekürzte Syntax verwendet Schlüsselwörter wie child, parent, ancestor-or-self um insgesamt 13 so genannte Achsen auszuwählen.

Beispiele ausgehend vom aktuellen Knoten (=Kontextknoten) Name des dritten Figure Elements.

abgekürzte Schreibweise	volle Schreibweise	Resultat
.	self::*	Name
..	parent::*	Figure
ancestor::*	Alle Elternknoten des Kontextknotens	Figure Figures

5 XML mit C# und .NET

XML ist ein wesentlicher Teil des Konzeptes von .NET und den .NET Sprachen. Konfigurationsdateien und die Ressource Dateien (*.resx) sind XML-Dateien, es gibt XML-Attribute zu den Feldern einer Klasse, die eine fast automatische Serialisierung ermöglichen, man kann ein DataGridView an ein XML-Schema koppeln usw. Die Erfahrung mit einem ziemlich großen Projekt hat gezeigt, dass man auch mit den elementaren Methoden die XML-Welt gut abdecken kann.

5.1 XML Dateien erzeugen und lesen

Sowohl für die Erzeugung als auch für das Lesen von XML-Dokumenten hat man grundsätzlich zwei Möglichkeiten.

- Sequenzielle Verfahren (XmlTextWriter, XmlTextReader)
- DOM (XmlDocument)

5.1.1 XmlTextWriter (.NET 1.1)

Das sequenzielle Verfahren eignet sich besonders zur Serialisierung eines Dokuments, das intern durch eine oft hierarchische Sammlung von Objekten repräsentiert wird.

Für das Beispiel mit den geometrischen Figuren dient eine von ArrayList abgeleitete Klasse FigureCollection zur Verwaltung einer Sammlung von Figuren. Es ist nahe liegend in dieser Klasse eine Methode zur Serialisierung in eine Xml-Datei zur Verfügung zu stellen.

```
public void SerializeToXMLFile(string fileName)
{
    XmlTextWriter xw;
    xw = new XmlTextWriter(fileName, System.Text.Encoding.UTF8);
    xw.Formatting = Formatting.Indented;
    xw.Indentation = 1;
    xw.IndentChar = '\t';

    xw.WriteStartDocument();
    xw.WriteStartElement("Figures");           // writes <Figures>

    for (int i = 0; i < base.Count; i++)
    {
        Figure f = (Figure) base[i];
        xw.WriteStartElement("Figure");       // Write <Figure>
        xw.WriteAttributeString("Id", f.Id);
        xw.WriteElementString("Type", f.Type);
        xw.WriteElementString("Size", XmlConvert.ToString(f.Size));
        xw.WriteElementString("Color",
            XmlConvert.ToString(f.ColorNr));
        xw.WriteEndElement();                // Write </Figure>
    }
}
```

```

xw.WriteEndElement(); // writes </Figures>
xw.WriteEndDocument();

xw.Close();
}

```

Der Code bedarf wohl kaum einer weiteren Erläuterung.

5.1.2 XmlTextReader

Die Arbeit mit dem XmlTextReader erscheint mir etwas schwierig. Der folgende Code list trotzdem die Liste der Figuren wieder ein und erzeugt die Objekte.

```

public void SerializeFromXmlFile (string fileName)
{
    XmlTextReader xr = new XmlTextReader(fileName);
    xr.WhitespaceHandling = WhitespaceHandling.None;

    base.Clear();

    string msg;
    string id = "";
    string type = "Undefined";
    Figure f = null;
    string sRead = "";

    xr.MoveToContent();
    while (xr.Read())
    {
        if (xr.NodeType == XmlNodeType.Element)
        {
            string name = xr.Name;
            switch (xr.Name)
            {
                case "Figure":
                    id = xr.GetAttribute("Id");
                    break;
                case "Type":
                    type = xr.ReadString();
                    switch (type)
                    {
                        case "Circle":
                            f = new Circle();
                            break;
                        case "Square":
                            f = new Square();
                            break;
                        case "Triangle":
                            f = new Triangle();
                            break;
                        default:

```

```

        f = null;
        break;
    }
    f.Id = id;
    base.Add(f);
    break;
case "Size":
    sRead = xr.ReadString();
    f.Size = (float) XmlConvert.ToDouble(sRead);
    break;
case "Color":
    sRead = xr.ReadString();
    f.ColorNr = XmlConvert.ToInt32(sRead);
    break;
default:
    sRead = "";
    break;
}
msg = String.Format("{0} {1}", name, sRead);
Debug.WriteLine(msg);
}
}
xr.Close();
}

```

Sowohl beim Schreiben als auch beim Lesen werden die XmlConvert Methoden verwendet um die Textknoten zu erzeugen und in C# Datentypen umzuwandeln. Besonders beim Lesen ist zu beachten, dass nicht passende Daten zu einer Exception führen, die man besser abfangen müsste.

5.1.3 Lesen über XmlDocument (DOM)

Ein XmlDocument kann mit einer einzigen Anweisung mit dem Inhalt eines Xml-Files gefüllt werden. Dann kann man beliebig im Knotenbaum navigieren und den Inhalt lesen. Die wichtigsten Methoden sind dabei

SelectSingleNode und

SelectNodes.

Zur Auswahl eines Knotens oder einer Knotenmenge verwendet man XPath Ausdrücke.

```

public void ReadUsingDom (string fileName)
{
    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.Load(fileName);

    XmlNode node;
    node = xmlDoc.SelectSingleNode(@" /Figures");
    if (node == null) return;

    figureList.Clear(); // remove all entries in the collection
}

```

```

XmlNodeList nodeList = node.SelectNodes("Figure");
foreach (XmlNode child in nodeList)
{
    Figure f;
    XmlNode testNode;
    string type, sRead;
    testNode = child.SelectSingleNode("Type");
    type = testNode != null ? testNode.InnerText : "Undefined";
    switch (type)
    {
        case "Circle":
            f = new Circle();
            break;
        case "Square":
            f = new Square();
            break;
        case "Triangle":
            f = new Triangle();
            break;
        default:
            f = null;
            break;
    }

    if (f == null) continue;

    // get the properties and set the values

    testNode = child.SelectSingleNode("@Id");
    sRead = testNode != null ?
        testNode.InnerText : "UndefinedId";
    f.Id = sRead;

    testNode = child.SelectSingleNode("Size");
    sRead = testNode != null ? testNode.InnerText : "100.0";
    f.Size = (float) XmlConvert.ToDouble(sRead);

    testNode = child.SelectSingleNode("Color");
    sRead = testNode != null ? testNode.InnerText : "0";
    f.ColorNr = XmlConvert.ToInt32(sRead);

    figureList.Add(f);
    Debug.WriteLine(String.Format("{0} {1} {2:f3}",
        type, f.Id, f.Size));
}
// clean up, not really necessary
xmlDoc.RemoveAll();
xmlDoc = null;
}

```

5.2 Validierung im Programm

Um ein XML-File in einem Programm zu validieren, muss man das Schema in eine XmlSchemaCollection aufnehmen und das Xml-Dokument über einen ValidierungsReader lesen.

```
public bool ReadXmlFile(string xmlFileName, string
schemaFileName)
{
    xmlTextReader = new XmlTextReader(xmlFileName);
    XmlSchemaCollection schemas = null;

    schemas = new XmlSchemaCollection();
    schemas.Add(null, schemaFileName);

    XmlValidatingReader validationReader =
        new XmlValidatingReader(xmlTextReader);
    validationReader.Schemas.Add(schemas);

    // Set the validation event handler
    validationReader.ValidationEventHandler +=
        new ValidationEventHandler(ValidationCallback);

    xmlDoc.Load(validationReader);
    validationReader.Close();
    xmlTextReader.Close();
    ...
}
```

```
void ValidationCallback (object sender, ValidationEventArgs args)
{
    // split the sentences (delimiter is '.') in the message into
    lines.
    Regex splitToLines = new Regex(@"\. ");
    string [] messageLines = splitToLines.Split(args.Message);
    int n = messageLines.Length;
    string description = "";
    // do not use the last line (contains the file name)
    for (int i = 0; i < n - 1; i++)
    {
        messageLines[i] += ".";
        description += messageLines[i] + " ";
    }
    // get the current line and column from the xml reader
    string message;
    XmlSchemaException e = args.Exception;
    message =
        String.Format("Error in line {0}, column {1} ({2} {3} {4})",
            e.LineNumber, e.LinePosition, xmlTextReader.Name,
            xmlTextReader.Value, description);
    Debug.WriteLine(message);
}
```

```
errors.Add(message);  
}
```

6 XML in einer beliebigen Programmierumgebung

Der Großteil der an einem PC installierten Software stellt die Funktionalität der Software auch über eine Automatisierungsschnittstelle zur Verfügung. So kann man z.B. auch in VBA ein DOM-Objekt erzeugen und damit arbeiten.

```
' Using a DOMDocument object allows us to load an XML document  
' and to select different nodes  
' The XML objects are only available, if you select Microsoft XML  
x  
' in the list of available references Menu Tools -> References  
  
Public Const xmlFile As String =  
"d:\edu\xmlhtml\Files\Figures.xml"  
  
Public Sub XMLExample()  
  
    Dim xmlDoc As DOMDocument  
    Dim xmlNode As IXMLDOMNode  
    Dim xmlEndNode As IXMLDOMNode  
    Dim xmlNodeList As IXMLDOMNodeList  
    Dim success As Boolean  
    Dim i As Integer, n As Integer  
    Dim id As String  
    Dim ftype As String  
    Dim size As String  
    Dim mySheet As Worksheet  
  
    Set xmlDoc = New DOMDocument  
    success = xmlDoc.Load(xmlFile)  
    If Not success Then Exit Sub  
  
    Set mySheet = Sheets(1)  
  
    Set xmlNode = xmlDoc.selectSingleNode("Figures")  
    Set xmlNodeList = xmlNode.selectNodes("Figure")  
    n = xmlNodeList.Length  
    Debug.Print n, "nodes selected"  
    i = 1  
    For Each xmlNode In xmlNodeList  
        Set xmlEndNode = xmlNode.selectSingleNode("@Id")  
        id = xmlEndNode.Text  
        Set xmlEndNode = xmlNode.selectSingleNode("Type")  
        ftype = xmlEndNode.Text  
        Set xmlEndNode = xmlNode.selectSingleNode("Size")  
        size = xmlEndNode.Text  
        Debug.Print i, id, ftype, size
```



```

    mySheet.Cells(i, 1) = id
    mySheet.Cells(i, 2) = ftype
    mySheet.Cells(i, 3) = size
    i = i + 1
Next xmlNode

```

End Sub

7 XSLT

XSL ist mehr als eine *Style Sheet Language*.

XSL besteht aus drei Teilen:

- XSLT ist eine Sprache zur Transformation von XML Dokumenten
- XPath ist eine Sprache um Teile eines XML Dokument auszuwählen, bzw. in einem XML Dokument zu navigieren
- XSL-FO ist eine Sprache um XML Dokumente zu formatieren

XSL ist ein Konzept, das es ermöglicht XML nach XML, XHTML, Text etc. zu transformieren, es erlaubt die Filterung und Sortierung von XML Daten, eine datenabhängige Formatierung von XML Daten.

Die Quelle einer Transformation ist immer ein XML Dokument. Der XSLT Prozessor holt sich die Instruktionen aus einem XSLT File und schickt das Resultat der Transformation auf einen Ausgabestrom.

Ein XML-File kann das XSLT File als "*processing instruction*" definiert haben:

```
<?xml-stylesheet type="text/xsl" href="ToHtml.xslt" ?>
```

Wird ein XML-File mit einer solchen Anweisung von einem Browser geladen, so wird der XSLT Prozessor aufgerufen und das Resultat der Transformation angezeigt.

Die meisten XSLT Prozessoren sind auch als ausführbares Programm in einer Kommandozeilenversion verfügbar. Z.B. kann man den Microsoft XSLT Prozessor folgendermaßen verwenden:

```
C:\>msxsl -?
```

```
Microsoft (R) XSLT Processor Version 4.0
```

```
Usage: MSXSL source stylesheet [options] [param=value...] [xmlns:prefix=uri...]
```

```
Options:
```

```

-?           Show this message
-o filename  Write output to named file
-m startMode Start the transform in this mode
-xw         Strip non-significant whitespace from source and stylesheet
-xe         Do not resolve external definitions during parse phase
-v          Validate Documents during parse phase
-t          Show load and transformation timings
-pi         Get stylesheet URL from xml-stylesheet PI in source Dokument
-u version  Use a specific version of MSXML: '2.6', '3.0', '4.0'
-           Dash used as source argument loads XML from stdin
-           Dash used as stylesheet argument loads XSL from stdin

```

Mxsl findet man über Google leicht zum Download, er funktioniert allerdings nur mit der Version 4 der Microsoft Core XML Services (MSXML). Die Version 3 wird mit dem Internet Explorer installiert.

7.1 Transformation eines XML Dokuments in ein Textfile.

Wir verwenden wieder unser XML Beispiel mit einer Liste von Figuren. XSLT Transformationen sind ebenfalls XML-Files und können mit Werkzeugen wie XML-Spy oder mit dem Visual-Studio erstellt werden.

Je nach Werkzeug wird dann z.B. gleich mehr oder weniger des äußeren Fragments eines XSLT Files erzeugt.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Mit der Output-Anweisung wird das Ausgabeformat festgelegt:

```
<xsl:output method="text" encoding="ISO-8859-1"/>
```

Es ist einleuchtend, dass wir für eine Transformation zwei Dinge benötigen:

- Kommandos um Ausgaben zu erzeugen, d.h. auf den Ausgabestrom zu schreiben.
- Kommandos, um Elemente im XML Dokument auszuwählen.

Ein XSLT Style Sheet ist ein XML Dokument. Die Anweisungen für den XSLT Prozessor sind Elemente aus dem xsl: Namensraum.

Zuerst liest der XSLT Prozessor das Style Sheet und konvertiert es in eine interne Datenstruktur, so dass er schnell findet, was er braucht.

Dann muss er das Dokument lesen und ebenfalls in einer Datenstruktur, einem Baum abbilden.

Für jeden Verarbeitungsschritt ist ein so genannter Kontext definiert. Dies ist der aktuelle Verarbeitungsknoten oder eine Liste von Knoten. Zu Beginn ist der Kontext die Wurzel des XML Dokuments. Dann arbeitet der Prozessor nach folgendem Prinzip:

Solange der Kontext Knoten enthält führe aus:
Hole den nächsten Knoten des Kontexts. Gibt es Vorlagen (<i>templates</i>) die zu diesem Knoten passen? Falls ein oder mehrere Vorlagen passen, verwende die richtige und verarbeite sie.

Beachte, dass dies ein rekursives Verarbeitungsmodell ist.

Das folgende Beispiel transformiert unser Musterdokument XmlTestDokument.xml in ein Textfile.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="text" encoding="ISO-8859-1"/>
```

```

<xsl:template match="/">
  <xsl:call-template name="Text"></xsl:call-template>
  <xsl:apply-templates select="/Figures"/>
</xsl:template>

<xsl:template name="Text">:method text: Text nodes in the
template are directly send to the output stream.
<tag>XML-tags are not processed!</tag>
-----
</xsl:template>

<xsl:variable name="newLine">
<xsl:text>
</xsl:text>
</xsl:variable>

<xsl:variable name="tab">
<xsl:text>&#9;</xsl:text>
</xsl:variable>

<xsl:template match="Figures">
  <xsl:apply-templates select="Figure"/>
</xsl:template>

<xsl:template match="Figure">
  <xsl:text>Figure Id = </xsl:text>
  <xsl:value-of select="@Id"/>
  <xsl:value-of select="$tab"/>
  <xsl:text> Name = </xsl:text>
  <xsl:value-of select="Name"/>
  <xsl:value-of select="$tab"/>
  <xsl:text>Size = </xsl:text>
  <xsl:value-of select="Size"/>
  <xsl:value-of select="$newLine"/>
</xsl:template>

</xsl:stylesheet>

```

```
<xsl:output method="text" encoding="ISO-8859-1"/>
```

Diese Anweisung definiert eine von drei Ausgabe Methoden: text, xml oder html und die Codierung des erzeugten Files.

Die Vorlage

```

<xsl:template match="/">
  <xsl:call-template name="Text"></xsl:call-template>
  <xsl:apply-templates select="Figures"/>
</xsl:template>

```

bezieht sich auf die Wurzel des Dokuments und definiert, was zuerst passiert. Der Aufruf des Templates mit dem Namen "Text" produziert einige Ausgaben, ändert aber den Kontext nicht.

Die nächste Instruktion macht das Element Figures zum aktuellen Kontextknoten und ruft das zum Kontext passende Template auf. Im Template für Figures wird durch den XPath Ausdruck Figure eine Knotenliste mit allen Figure Elementen zum aktuellen Kontext und das Template zu Figure für alle Knoten dieser Liste ausgeführt.

Am Beginn des Style Sheets sind zwei globale "Variablen" definiert. Die Werte dieser Variablen sind die Steuerzeichen für einen Zeilenumbruch und den Tabulator.

```
<xsl:text>Figure Id = </xsl:text>
<xsl:value-of select="@Id"/>
<xsl:text>  Name = </xsl:text>
<xsl:value-of select="Name"/>
<xsl:value-of select="$newline"/>
```

Diese Anweisungen schreiben den Text "Figure Id = " gefolgt vom Wert des Id-Attributes auf den Ausgabestrom. Beachte, dass der aktuelle Kontext ein Figure-Knoten ist, deshalb selektiert der XPath Ausdruck "@Id" das Id-Attribut des "Figure" Elements. Der Wert des selektierten Elements "Name" ist der Inhalt des Elements "Name". Anschließend wird noch eine neue Zeile erzeugt.

7.2 XML Dokument in ein HTML File transformieren

Drei Beispiele zeigen die Transformation eines XML Dokuments in ein HTML-File.

ToHtml1.xslt, ToHtml2.xslt und ToHtml3.xslt.

Quelltext des ersten Beispiels:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="ISO-8859-1"/>

<xsl:template match="/">
<html>
  <head>
    <title>Result of XSLT</title>
  </head>
  <body>
    <h3>Figure List</h3>
    <table border="1" cellspacing="0" cellpadding="3"
width="60%">
      <tbody>
        <tr>
          <td><b>Id</b></td>
          <td><b>Name</b></td>
          <td><b>Size</b></td>
          <td><b>Area</b></td>
        </tr>
        <xsl:apply-templates select="/Figures/Figure"/>
      </tbody>
    </table>
  </body>
</html>
</xsl:template>
```

```

<xsl:template match="Figure">
  <tr>
    <td><xsl:value-of select="@Id"/></td>
    <td><xsl:value-of select="Name"/></td>
    <td><xsl:value-of select="Size"/></td>
    <xsl:variable name="s" select="Size"/>
    <xsl:choose >
      <xsl:when test="Type='Circle'">
        <td><xsl:value-of select=
          "format-number($s*$s*3.1415 div 4.0, '#.##')"/></td>
      </xsl:when>
      <xsl:when test="Type='Square'">
        <td><xsl:value-of select=
          "format-number($s*$s, '#.##')"/></td>
      </xsl:when>
      <xsl:when test="Type='Triangle'">
        <td><xsl:value-of select=
          "format-number(1.7321*Size*Size div 4.0,
            '#.##')"/></td>
      </xsl:when>
      <xsl:otherwise>
        <td>---</td>
      </xsl:otherwise>
    </xsl:choose>
  </tr>
</xsl:template>
</xsl:stylesheet>

```

Wenn die Ausgabemethode html oder xml ist, werden alle Tags außer den <xsl:...> Tags einfach an den Ausgabestrom weitergereicht.

7.3 XML nach XML

Dies ist sehr ähnlich wie die Transformation nach HTML. Transformationen von XML nach XML erlauben eine praktisch beliebige Umwandlung eines Quelldokuments in ein neues Dokument. Man kann diese Technik z.B. zum Sortieren von Listen oder zum Laden und Schreiben von XML-Dokumenten in verschiedenen Versionen verwenden.

Das folgende Beispiel liest wieder eine Liste von Figuren und transformiert das File in ein File, dass für jedes Figure Element den Namen löscht und ein ColorNr Element einfügt.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>

<xsl:template match="/">
  <Figures>
    <xsl:apply-templates select="/Figures/Figure"/>
  </Figures>
</xsl:template>

<xsl:template match="Figure">
  <Figure>
    <xsl:copy-of select="Type" />
    <xsl:copy-of select="Size" />
    <ColorNr>0</ColorNr>
  </Figure>
</xsl:template>

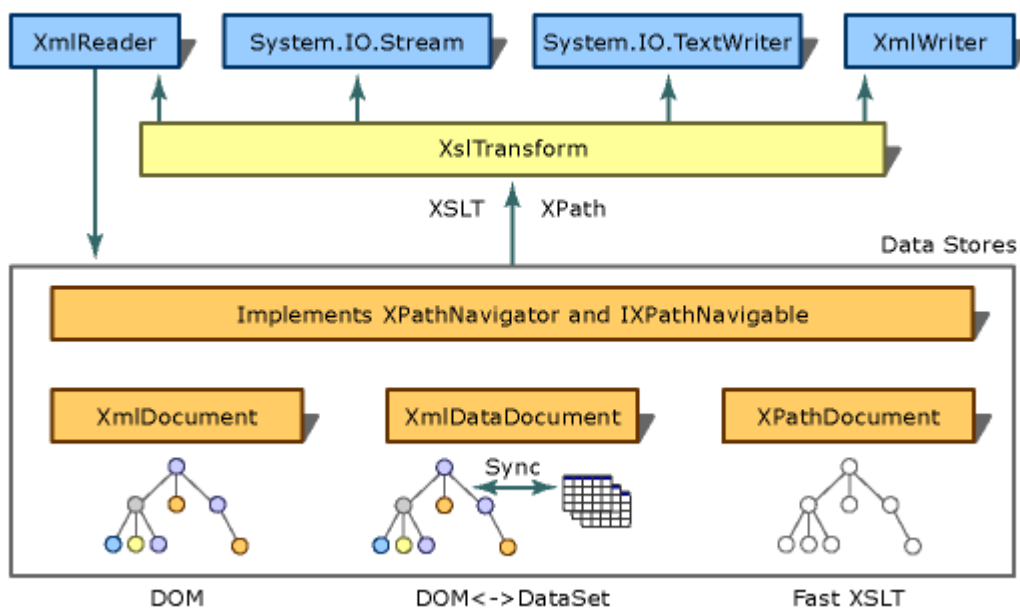
</xsl:stylesheet>
```

7.4 XSLT im Programm

Aus der Dokumentation zu Visual Studio:

The goal of the Extensible Stylesheet Language Transformation (XSLT) is to transform the content of a source XML document into another document that is different in format or structure. For example, to transform XML into HTML for use on a Web site or to transform it into a document that contains only the fields required by an application. This transformation process is specified by the W3C XSL Transformations (XSLT) Version 1.0 recommendation located at www.w3.org/TR/xslt. In the .NET Framework, the XslTransform class, found in the System.Xml.Xsl namespace, is the XSLT processor that implements the functionality of this specification. There are a small number of features that have not been implemented from the W3C XSLT Version 1.0 recommendation, listed in Outputs from an XslTransform. The following figure shows the transformation architecture of the .NET Framework.

Transformation Architecture



The XSLT recommendation uses XPath to select parts of an XML document, where XPath is a query language used to navigate nodes of a document tree. As shown in the diagram, the .NET Framework implementation of XPath is used to select parts of XML stored in several classes, such as an `XmlDocument`, an `XmlDataDocument`, and an `XPathDocument`. An `XPathDocument` is an optimized XSLT data store, and when used with `XslTransform`, it provides XSLT transformations with good performance.

Code example:

```
using System;
using System.IO;
using System.Xml;
using System.Xml.XPath;
using System.Xml.Xsl;

public class Sample
{
    private const String filename = "mydata.xml";
    private const String stylesheet = "myStyleSheet.xsl";

    public static void Main()
    {
        XslTransform xslt = new XslTransform();
        xslt.Load(stylesheet);
        XPathDocument xpathdocument = new
        XPathDocument(filename);
        XmlTextWriter writer = new XmlTextWriter(Console.Out);
        writer.Formatting=Formatting.Indented;

        xslt.Transform(xpathdocument, null, writer, null);
    }
}
```