

Understanding XML: A Software Development Approach

Hossein Hassani



Hossein Hassani

Understanding XML

A Software Development Approach



Understanding XML: A Software Development Approach

1st edition

© 2015 Hossein Hassani & bookboon.com

ISBN 978-87-403-0988-1

Contents

	About the Author	11
	Preface	12
1	Introduction	14
1.1	Data Structure	15
1.2	Data Definition and Data Interchange	18
1.3	XML and Software Development	19
1.4	Alternative Data Interchange Methods	20
2	What is XML?	21
2.1	Advantages of XML	22
2.2	XML Elements	23
2.3	XML Attributes	24
2.4	XML Declaration	25
2.5	XML Comments	26
2.6	XML Ordering	26



 **MTHøjgaard**

BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang



2.7	Well-formedness	27
2.8	Disadvantages of XML	29
2.9	When to use XML?	30
3	XML Validity	31
3.1	Document Type Definition (DTD)	32
3.2	Element Type Declaration	33
3.3	Attribute List Declaration	35
3.4	When to Use Attributes?	36
3.5	Document Validity	38
3.6	Using DTD	38
3.7	Using DTD as a Separate File	39
4	Using XML	41
4.1	Tree-based XML API	42
4.2	Event Driven XML API	43
4.3	DOM versus SAX	47
4.4	Namespace	47



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.



5	XSL – eXtensible Stylesheet Language	51
5.1	XSLT – XSL Transformation	52
5.2	XPointer	55
5.3	XLink	55
6	XML Schema	57
6.1	The <schema> Element	58
6.2	Simple Elements and Simple Types	58
6.3	Complex Elements and Complex Types	59
6.4	XML Schema Indicators	60
6.5	References	61
6.6	New Type Definition	62
6.7	Groups	62
6.8	Constraints	63
6.9	Key Constraints	63
7	XML Query Languages	64
7.1	XQuery	65
7.2	XML Information Set (XML Infoset)	68



CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired

8	XML and Databases	69
8.1	Storing XML in an Attribute	69
8.2	Storing XML in Several Attributes and Relations	70
8.3	Schema-Independent Representation	71
8.4	SQL/XML	71
8.5	Load XML into MySQL	73
8.6	Native XML Databases	74
8.7	XML and Big Data	77
9	XML and Software Development Platforms	78
9.1	XMLWriter with PHP	78
9.2	Using XML in Python	79
9.3	XMLWriter in MS Visual Studio	82
9.4	XML Specialized Languages	82
9.5	XHTML – eXtensible HTML and HTML5	83
9.6	Security in XML	83
9.7	XML and NLP (Natural Language Processing)	84
	Bibliography	86



Max's next Bookboon eBook
Your Boss: Sorted!
 By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

Figures

Figure 1 – Citation data tree	17
Figure 2 – XML Sample	21
Figure 3 – Using attribute	24
Figure 4 – Ordering (1)	26
Figure 5 – Ordering (2)	26
Figure 6 – Well-formed XML	28
Figure 7 – Not Well-formed XML	29
Figure 8 – DTD example	34
Figure 9 – Attributes and Attribute List Declaration	37
Figure 10 – Using DTD inside the XML document	39
Figure 11 – Using DTD as a separate document	40
Figure 12 – XML DOM Example	42
Figure 13 – SAX Example (Java SAX)	46
Figure 14 – XML – Sample 1 (name conflict)	47
Figure 15 – XML – Sample 2 (name conflict)	48
Figure 16 – XML – Sample 1 (using prefix for name conflict resolution)	49
Figure 17 – XML – Sample 2 (using prefix for name conflict resolution)	49
Figure 18 – XML namespace – Example	50
Figure 19 – Creating Table and Inserting a row using XML Type	72
Figure 20 – PHP XMLWriter – Example	79
Figure 21 – Python and DOM – Example	80
Figure 22 – Python and SAX – Example	81

Tables

Table 1 – DTD – element repetition rules	33
Table 2 – DTD (Element Types)	33
Table 3 – DTD (Attributes)	35
Table 4 – DTD (Attributes)	36
Table 5 – XPath Expressions	53
Table 6 – XPath Axes	54
Table 7 – Schema-Independent Representation Relation	71
Table 8 – SQL/XML Operators	72

To Shahin and Nava
(for the times that I have stolen from the family!)

About the Author

Hossein Hassani is a lecturer at the University of Kurdistan Hewlêr since 2007. He joined UKH after nearly twenty years of experience in software industry. He has taught different courses such as Project Management, Principle of Database Systems, Advanced Database Technologies, Software Engineering, Structured and Object Oriented Programming, Management Information Systems, Human Computer Interaction, Operating Systems, and Programming Languages.

Furthermore, He has taught some other courses such as Fundamentals of Programming Languages at other universities. In addition, he has mentored a group of postgraduate students as teacher assistants of software engineering laboratory for undergraduate programs in software engineering.

During his experience in software industry, he has worked in different positions starting from a junior programmer, promoting systematically to a senior programmer, then a designer, an analyst, a team leader, a project manager and finally a senior consultant. Furthermore, during this period he kept his relation with higher education institutions and teaching activities through providing different seminars and intensive courses to the audience of experts in the field. This long path of experience alongside diversity of the projects in which he has been involved and committed have enabled him to have a deep understanding of software and information technology projects, as well as giving him a holistic idea about the dynamic relations of computing with all aspects of humanity and social life. He shares his findings of this amazing journey with his students during lecturing and teaching.

He is interested in Information Accessibility, Computational Linguistics, and Software Quality Assurance.

Preface

Since the new century, the usage of XML (eXtensive Markup Language) has been growing rapidly. Its well-established architecture, fitness for diverse purposes, platform independence, and usability in both desktop and web-based applications are only few among the many attractive features, which have given this amazing invention of computer scientists/experts that “extensive” popularity.

The idea of compiling this book came to my mind as a result of several years teaching the concept in different database courses, mainly to undergraduate and recently preparing it as a part of postgraduate course in advanced database. During these years, whenever I started to introduce the subject I found that it was difficult for students to grasp the importance of this technology. They often would not appreciate it for a while, sometimes even up to after graduation. But, there were some that could find the ability of this technology to be adapted for different purposes and in different applications. I have seen some of my previous students coming back to me and talking about their experiences with XML.

Perhaps I am too enthusiastic to XML technologies. But whether I am or I am not, I believe that this is one of the best products of the people who are involved in the computing advancement. I am teaching programming languages as well, and I have found that XML technologies can also be considered as one of the less flawless products from the programming language perspective. The simplicity, flexibility, coherence, expandability, easy to develop, and understandability have made this technology unique. Unfortunately, with all of these benefits there are drawbacks too. One of the main drawbacks, to my mind, is the scattered resources of this technology. Despite comprehensive resources and tutorials that W3C provides, they are not easy to use for a novice to the technology. This applies to others who have provided rich resources that are very broad, both in terms of breadth and depth.

This book, aims to give you an insight into the XML technology and its applications from a software development perspective. It provides you with the diverse aspects of the technology, how to apply it in different environments and for different applications. You might find the material that the book covers by searching the Internet, but, most probably, not in the way that has been presented in this book. In fact, the amount of documents about XML is so huge, and growing every day, that can be considered as “information overload”, which subsequently may lead to “cognitive overload”.

The book simply tries to provide the readers with a brief and handy composition and collection of what might be scattered in different places, which have targeted different types of audiences, in one short reference with examples. But it does not intend to make you an expert. To become an expert in using XML, like all other topics in computing, needs you to do three important things: **practice, practice, and practice!**

Although the book mainly targets the undergraduate students, postgraduate students and computing practitioners can also use it as a source for software development using XML.

The book has 9 chapters. Each chapter explains a fundamental aspect behind the technology through giving examples. The highlighted sections provide the important points, syntaxes, and summarized guidelines.

The author would be grateful if he receives feedbacks from colleagues who have dedicated some time to review the book and students who have used the book as a companion for their software development courses. You can reach the author at hosseinh@ukh.ac and h.hosseini@rgu.ac.uk.

Last but not least, the author would like to warmly thank Dawand Sulaiman for reading the draft manuscript of the book. Dawand's meticulous review and creative suggestions have significantly improved the final work.

Hossein Hassani

Hewlêr

April 2015

1 Introduction

Managing, processing, exchanging, manipulating, and presenting data are some of the crucial activities in the context of information technology and computing. Indeed, in the first encounter to computing, whatever special field it might be, we are normally introduced to the notion of “input, process, output”, and then we are told that the input can be in any form of “data”. Afterwards, we are introduced to the importance of data, what that is, why it is so important, and how it should be handled. If you are in computer science, or software engineering, and such fields, then you have learned about data structure and databases as well. Some of you might have even heard about “big data”, “data warehouse”, and “data mining”. This is just to remind you how far the concept of data and data manipulation can go.

Since the beginning of computing as a science, data and data manipulation have gone through different stages. Many technologies have been presented, several of which have lasted shortly and some have been around since 1970s, for example. To illustrate, relational data concepts and relational databases have started in the 1970s and they are still among the dominant ones, both in industry and in academia. The concept even theorized in the format of Relational Calculus.

The quest for finding proper ways for collecting, preserving, manipulating, and presenting data has been an ongoing journey in computing. However, there have been several cases when this quest has produced some long-lasting fruits. XML is probably one of them. The XML term is that important that I am sure most of you who have started reading this book either because you have frequently heard about it or you are supposed to use it in one of your projects/assignments or you have been advised to learn and to find out how you can apply it.

But why XML is so important that it has become a *de facto* standard for data communication within software industry? The idea comes from the fact that despite our first understanding of data as a structured, predefined, strictly formatted thing, it is not the case for the majority of objects in the world. That is, the data around us, most of the time for your surprise, cannot be bundled easily in the rigid formats that ordinary databases, such as relational databases, provide.

The concept of **unstructured** or **semistructured** data has been known for a long time. Regardless of the structure and format of data, the growing demand for data exchange between businesses from one side, and the usage of the Internet from the other side, has encouraged researchers, scientists, and technologists to improve the technology for the collection, exchange, presentation, and manipulation of data. The following section would provide you with the characteristics of semistructured and unstructured data through some examples.

1.1 Data Structure

When we think about data, we normally used to think of data that has a predefined structure. Perhaps most of us are already familiar with relational data and relational database. We also might have used structures as a format of organized data and have applied the idea in programming and software development. In database systems, we have been told to design our data model that puts a solid and profound base for the target system. In most of database applications, the underlying data model is fixed.

Although there are different ways to make the system flexible, such as ANSI-Architecture, three-tier programming architecture, MVC, using “Views” concept, dynamic schema modification, and so on, but the underlying system is solid. However, in many cases the structure of data changes more often. Besides, the way that the data structure changes is not easily predictable. This situation leads to the concepts of unstructured and semistructured data.

Let us look at some examples. You might have seen different versions of these examples before.

Example 1:

An address might be composed of different items, which in turn forms different architectures as below:

- Zip Code, City, Country
- Home Number, Street, City, Province, Country
- Apartment Number, Home Number, Alley, Street, City, Province, Country
- ...

Example 2:

A book might have several architectures as below:

- Table of Contents, Preface, Chapters
- Table of Contents, Preface, Chapters, Exercises, Bibliography, Index
- Table of Contents, Preface, Chapter (Section, ...), Chapter Reference, Exercises, Glossary, Index
- ...

Example 3:

A reference entry, (or a library item) would have different citation formats and holds different data item depending on its type:

- Book: Authors, year, title, publisher, place, referenced page(s).
- Article: Authors, year, title, journal, volume, referenced pages, publisher, place.
- Movie: Title, year, director, country, production company.
- ...

Example 4:

An English sentence might have the following structures:

- Subject, Verb
- Subject, Verb, Object
- Subject, Verb, Adjective
- ...

Example 5:

A document can be:

- An invoice.
- An article.
- A user manual.
- A book.
- ...

Looking at these examples carefully, reveals some common themes among them as below:

- They are flexible (not rigid) in terms of their structures.
- They can be presented by means of graphs.
- They have a tree-like structure.



MTHøjgaard

BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang



Figure 1 shows a reference structure.

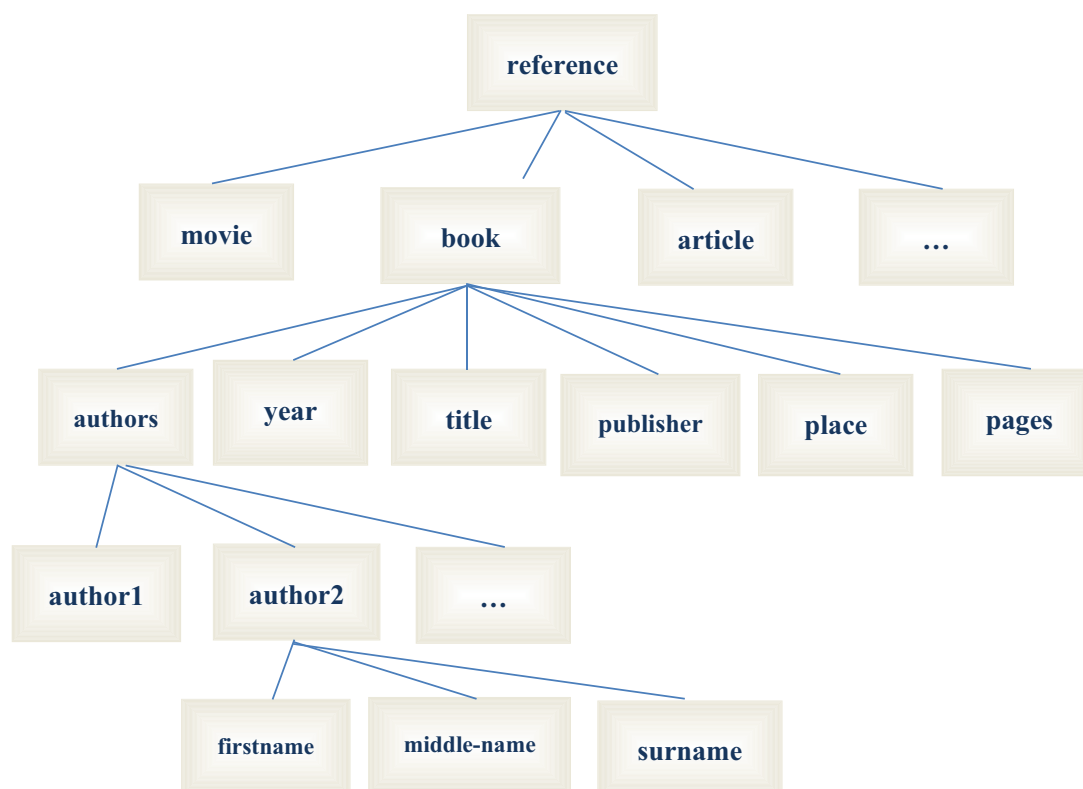


Figure 1 – Citation data tree

The data of the above examples are called “semistructured” data to distinguish it from a structured data. You may have dealt with the above examples, particularly, examples 1 and 3, in which case, most probably you have used a relational model and then a relational database to handle data. Although your approach is completely correct and acceptable, however, this is not the only way to organize this data.

For example, perhaps you have found it complicated when you have made a relational database for a library. Besides, if you want to share your data with another library they need to know about your database design, or you have to extract your data and tell them how they can “interpret” it.

One may argue that object-relational database or object oriented data modeling might help in this situation. I do not want to go into detail and drift of the main path of the book, but I can say that, unfortunately, even these type of data models would not help much with “semistructured” data and complexity would still remain an issue.

Data can be “unstructured” as well. In fact, videos, emails, and plain texts are considered as unstructured data. But in this book our focus is on semistructured data, because XML is a very powerful technology to handle this type of data. It does not mean that XML is not able to handle structured or unstructured data. In fact, XML as a Markup language has been designed to deal with all types of data. The reason that we exclude structured data is that for this type of data there are other powerful technologies and models that work properly. Regarding unstructured data, the concept falls in different category in data manipulation, which is out of the scope of this book, however, this type of data would lightly be discussed through some relevant sections as well.

1.2 Data Definition and Data Interchange

Data should be modeled in order to be properly handled. Data modeling is a phase in software engineering during which the way that data is collected, circulated, updated, and probably, retired is modeled. There are different methods, techniques, diagrams, and approaches that are used in this process. Depending on the situation, it is important that data can be easily shared among different users.

In most approaches to data manipulation, data definition is a separate part of the model that defines the structure of data. Sometimes this part is called “meta data”, which means “data about data”, that is, data by which we describe data. Data definition allows software developers to put a solid base that defines the data and keeps its integrity. Any change to data definition would have ramifications on the system. Hence, data modeling is a crucial phase in software engineering.

To make an analogy, it can be compared to the foundation of a building. If it goes wrong then many other things would probably go wrong. This has led computing researchers and scientists to seek some methods, and the technologies to build proper tools, to make the inevitable changes doable. However, sharing and availability of data in different environments still remains an issue. The reason is, normally these tools are used in an integrated environment, as we can see in Database Management Systems (DBMSs), for example, Oracle, MS SQL Server, and MySQL. Although these tools and systems provide several ways for data migration, data export and import, they still need a great deal of effort and specialties to handle the requests for data exchange among different environments.

But XML provides a flexible and yet easy to understand method to handle data. The main characteristics of this method can be summarized as below:

- It allows data to be “self-described”. That is, data and its description are bundled together.
- Data can be built using a simple text editor.
- Programming tools can use basic input and file operations in order to read data.
- Programming tools can use basic output and file operations in order to write and update data.
- Data can be shared regardless of platform and environment. This is called platform-independence.
- Data can be browsed using most of web browsers or simple text editors.
- It is readable by both machines (i.e. computers) and human beings.

1.3 XML and Software Development

The growing popularity of XML and its flexibility have caused that almost all major DBMS providers to integrate XML technology into their products, one way or another. You will see some examples of the tools and facilities that these providers offer throughout the book.

Similarly, most of providers in software development field have developed ready-made tools to support XML data manipulation. For example, PHP and Microsoft Visual Studio have provided XML readers and writers that can facilitate handling XML data in the systems that you are developing.

XML is developed and maintained by W3C (World Wide Web Consortium). W3C recommends the standards that shall be followed during XML usage. This keeps the XML as a platform-independent technology to encode and format documents. Moreover, major players in computing industry have been contributing to the XML advancement.

To summarize, XML is a technology that would continue to play a great role in computing in the coming years. Some experts even say that it would be the dominant tool in the data exchange and manipulation. Therefore, learning and becoming expert in XML is necessary for anyone who works in computing. The following chapters aim to fulfill a part of this necessity. For this, attempting exercises at the end of each chapter and trying to apply XML in different projects is a key factor to successfully learn and become expert in this simple, beautiful, and yet amazingly powerful technology.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.



Click on the ad to read more

1.4 Alternative Data Interchange Methods

XML is neither the first data interchange method nor will be the last one. The data interchange concept, in a modern sense, dates back to the 1940s. Consequently, the growth of the electronic data interchange requirements led to the appearance of more sophisticated methods in the 1970s and 1980s.

EDI (Electronic Data Interchange) was suggested in 1996 and soon became an international standard for data interchange. EDI is a widely used method for interchanging data among business partners throughout the world. It has many benefits and strengths such as security and simplicity. So one question might be, why we need another method?

The answer is beyond the aim of this book, but just to not leaving you without an answer, compare it with the different programming languages. Although, scientifically, we can program a computer to solve any computationally solvable problem with either one of the several hundreds (actually, thousands!) of programming languages, some can be more efficient, faster, and easy to use for the specific kinds of problems and some for the other kinds of problems. However, we have seen that some of the languages have replaced the others. Is it possible for EDI as well to be replaced by XML? I do not want to predict. I prefer to see both to live in co-existence as it gives more choices to users.

Again, during the time that EDI started emerging and while XML was coming into play, another technology appeared in the market of data interchange: JSON (JavaScript Object Notation). It is an open standard format for data interchange that has become very popular, particularly with the growth of the mobile computing and communication. It uses a text format and is language independent. You can refer to <http://json.org/> for more information. However, for those who are already familiar with JSON, some questions might arise. Should we use JSON instead of XML? Is JSON going to replace XML?

Again, from the users view, the fitness for use is a major factor. For many purposes, JSON can be a very handy choice for data interchange. But, you should not restrict yourself to only one tool. A developer should use tools according to the environment and the problems in hand. You should know that JSON, at least currently, is not a data container, and does not have the variety of supportive technologies that XML provides for different purposes beyond data interchange. It has no database. However, as it was mentioned, in many cases, particularly in mobile apps, it is a very good choice as a light-weight data interchange tool. JSON is emerging rapidly, though it does not seem to target to replace XML. Although fighting over the market always continue, if JSON wants to replace XML, it should develop all the related technologies that XML provide, in which case it is no longer the 'light-weight' JSON.

2 What is XML?

XML stands for eXtensible Markup Language. It means that it is, essentially, a Markup Language. But, what is a Markup Language?

Markup Language

- Markup Language is a system for marking or “tagging” a document in a way that shows the logical presence of the document.
- It gives instructions about how the document should be displayed electronically.
- In other words, it is a set of instructions, which is introduced in the format of “tag”s that would be incorporated into the text of the document and allows any medium, which is able of interpreting the tags to show the document in the way that the designers intended to present it.
- HTML (Hyper Text Markup Language) is the most famous Markup Languages. This is the format based on which most of the Internet text documents are presented.
- HTML has evolved during the years. As a result, several versions have been produced, all of which currently co-exist together, e.g., HTML, DHTML, and HTML5.
- The main purpose of a Markup Language is to allow different documents to be displayed to users in the same way, regardless of the medium that they use. Hence, when you are browsing a page on your desktop, laptop, or your smart phone (with Firefox, IE, Safari, Chrome, Opera, or any other browser) you see the same thing.
- Despite the fact that the XML is a markup language, its main purpose is not rendering (showing) data. This is one of its differences with other markup languages such as HTML.
- Instead, the main purpose of XML is to describe data. Describing data means that XML through its tags allow an XML reader to understand how data has been organized into different parts and what each part includes.

Most of the previous Markup Languages were using a set of per-defined **tags**, whereas in XML you define your own tags. This is why it is called eXtensible. In the recent years some of Markup Languages, particularly HTML, have been reshaped and reconfigured, which augmented them with the capability of allowing users to define new tags. But, interestingly, they have been reshaped based on XML!

Figure 2 shows a sample of an XML data.

```
<sampleDic>
  <word>
    <head> ethic </head>
    <definition>
      rules of behavior based on ideas about what is morally good and
      bad.
    </definition>
  </word>
</sampleDic>
```

Figure 2 – XML Sample

You can easily understand (or at least guess) that this is a document that contains a word (“ethic”) and its definition (“rules of behavior based on ideas about what is morally good and bad.”). This is what is meant when it is said that the XML document is self-descriptive. You will find out about the structure and syntax of the above XML data later in this chapter.

2.1 Advantages of XML

XML is considered as an extremely useful technology in computing, since it was invented in the end of the previous century. In the answer to the question “what are the advantages of this technology?”, one can provide a long list of benefits for using XML. The application of XML has been growing steadily since it was created. A large amount of documents are created and stored in XML format. These documents could be exchanged and shared on the Internet or used for internal purposes within enterprises and organizations. But, what is XML? Before talking about XML, it is important to understand some concepts, which can help us to realize the necessity of something such as XML. Below is a summary XML advantages.



CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired

Advantages of XML

- XML is self-descriptive. This means that XML, basically, does not need any extra documents to interpret it.
- XML is both machine and human readable. Unlike many coded documents, an XML document is easily readable by human beings, while it can be efficiently processed by computers as well.
- Easy to construct (compose). Constructing an XML document does not need any special tool. Any trivial editor (e.g., TextEdit, Notepad, and gedit) can be used for this purpose.
- XML document can be easily shared among different users.
- XML data can be easily transported. There is no need for compatibility of source and destination application.
- XML is platform-independent. No matter what operating environment you use (propriety based such as Windows/Mac or open source such as Ubuntu) XML could be understood in the same way.
- XML can be used as a data container, in which case you do not need any special DBMS to manage your data.
- It is supported and standardized by W3C.
- It supports almost all human languages through supporting Unicode.

As an example that we will follow throughout this book, consider a dictionary data. I have chosen this case for two reasons. First, its data can be very simple and it can grow to become more complex based on our choice and application. Second, its data meets one of previously mentioned condition, which was a data that is mainly for reading and not updating, hence a proper candidate for using XML as data container.

2.2 XML Elements

XML is composed of elements. An XML element is the basic structure of XML documents. Therefore, an XML document can be considered as a forest of elements.

XML Element

- An XML element is a name that is given to a specific part of the XML document.
- It must be presented in the following way:
 - o `<elementname>any text</elementname>`
 - o The `<elementname>` is called **start-tag**.
 - o The `</elementname>` is called **end-tag**.
 - o "`<elementname>`" starts the element definition and "`</elementname>`" ends the element definition.
- Element name:
 - o Can be composed of alphanumeric and other characters.
 - o It cannot start with a number or a punctuation character (e.g., `? # % , "`).
 - o No space is allowed between characters.
 - o It cannot start with any forms of XML letters (xml, xML, and such).
- Example:
 - o *Valid* XML element: `<word>Ethic</word>`
 - o *Invalid* XML element: `<1word>any text</1word>`

In Figure 2, “sampleDic”, “word”, “head”, and “definition” are elements. “sampleDic” is an special element, which is called **root**. Root encompasses all other elements in an XML document.

2.2.1 Element Name Guide

The simple advice for naming is to follow self-descriptive notion of XML. Assign names to elements based on best practices of naming in computing. Follow a single convention in the whole document. XML is not something for coding and encryption in general. Therefore, avoid any mysterious approach unless it is really necessary (if it is at all!). General guideline of programming is applicable here too, though with a big emphasis on understandability and self-descriptiveness.

Consult W3Schools at http://www.w3schools.com/xml/xml_elements.asp to find out more about this.

2.3 XML Attributes

Attributes are **name-value** pairs that contain descriptive information about an element. Attribute is placed inside the start-tag after the corresponding element name. Attribute value must be enclosed in quotes.

An attribute can only occur once within a tag, whereas nested elements (which sometimes are called sub-elements) within the same tag may be repeated. Figure 3 shows the previous example restructured using an attribute. In this example, a new “definition” element has been added to the word “ethic” in order to show the mentioned case of repeatability.

```
<sampleDic>
  <word head = "ethic" >
    <definition>
      rules of behavior based on ideas about what is morally good and bad.
    </definition>
    <definition>
      a set of moral principles, especially ones relating to or affirming a
      specified group, field, or form of conduct.
    </definition>
  </word>
</sampleDic>
```

Figure 3 – Using attribute

2.4 XML Declaration

It is an optional statement at the start of XML document. It can have a simple format or using parameters.


XML Declaration

- General format :
`<?xml version='1.0' encoding='character encoding' standalone='yes|no'?>`
- Example 1: `<?xml version='1.0'>`
- Example 2: `<?xml version='1.0' encoding='UTF-8'?>`
- Example 3: `<?xml version="1.0" encoding='ISO-8859-1'?>`
- Example 4: `<?xml version='1.0' standalone='yes'?>`

Not 0 – If XML declaration is used it must appear as the first line of the document.

Not 1 – The default encoding in the Internet is UTF-8, hence Examples 1 and 2 declare the same thing.

Not 1 – The order of declaration parameters is important. It means that “standalone” must appear last. The aim of “standalone” will be explained in the next chapter.



Max's next Bookboon eBook
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

2.5 XML Comments

Comments in XML documents are enclosed in `<!--` and `-->` tags.

XML Comments
<code><!-- This is an example of an XML comment. --></code>

2.6 XML Ordering

As it was mentioned, XML documents have a tree-like structure. They have a root and elements that are ordered. The order depends on two factors: the purpose of the document and the way that it is processed. To illustrate, Figures 4 and 5 show two XML documents, which are **not** identical.

```
<sampleDic>
  <word>
    <definition>
      rules of behavior based on ideas about what is morally good and bad.
    </definition>
    <source>
      Merriam-Webster, online
    </source>
    <usage>
      Ethics is his chosen field of study.
    </usage>
  </word>
</sampleDic>
```

Figure 4 – Ordering (1)

```
<sampleDic>
  <word>
    <source>
      Merriam-Webster, online
    </source>
    <definition>
      rules of behavior based on ideas about what is morally good and bad.
    </definition>
    <usage>
      Ethics is his chosen field of study.
    </usage>
  </word>
</sampleDic>
```

Figure 5 – Ordering (2)

However, for attributes the order does not matter.

2.7 Well-formedness

It is important to provide an XML document that is structurally “well formed”. Well-formedness, in the context of XML, means that its structure is syntactically correct. In other words, we can say an XML document is well-formed if and only if its structural syntax is correct. For the architecture to be syntactically correct, every element must be well-formed.

It might be considered as a verbose explanation, but because it causes some confusion for the beginners, I have tried to clarify the difference by differentiating between “syntactical correctness” and “structural correctness”. There is another aspect of XML, which is called **validity**, which deals with syntax in a more sensible way, which will be explained in the next chapter.



 **MTHøjgaard**

**BEDRE
LØSNINGER**

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang



XML Well-formedness

- An XML document is well-formed if its architecture is syntactically correct.
- The architecture of an XML is syntactically correct if and only if it has a root element and the architecture of all its elements are syntactically correct.
- The root element must contain all other elements.
- The architecture of an element is syntactically correct if:
 - o It is enclosed in a start and end pair of tags.
 - o The element names follow the naming rules.
- If an element does not have any data, it is called an empty element.
- Empty elements can be shown in both following ways:
 - o `<elementname></elementname>`
 - o `<elementname/>`

2.7.1 Well-formed XML

Figure 6 shows a well-formed XML.

```

<sampleDic>
  <word>
    <head> ethic </head>
    <definition>
      rules of behavior based on ideas about what is morally good and bad.
    </definition>
    <source>
      Merriam-Webster, online
    </source>
    <usage>
      Ethics is his chosen field of study.
    </usage>
  </word>
  <word>
    <head> moral </head>
    <definition>
      concerned with the principles of right and wrong behavior and the
      goodness or badness of human character.
    </definition>
    <source>
      The Internet, online
    </source>
    <usage>
      the moral dimensions of medical intervention
    </usage>
  </word>
</sampleDic>

```

Figure 6 – Well-formed XML

2.7.2 Not Well-formed XML

Figure 7 shows an XML, which is **not** well-formed.

```

<sampleDic>
  <word>
    <head> ethic </head>
    <definition>
      rules of behavior based on ideas about what is morally good and bad.
    </definition>
    <source>
      Merriam-Webster, online
    </source>
    <usage>
      Ethics is his chosen field of study.
    </usage>
  </sampleDic>
  <word>
    <head> moral </head>
    <definition>
      concerned with the principles of right and wrong behavior and the
      goodness or badness of human character.
    </definition>
    <source>
      The Internet, on-line
    </source>
    <usage>
      the moral dimensions of medical intervention
    </usage>
  </word>

```

Figure 7 – Not Well-formed XML

There are two issues with the document. The first issue is that there is no root element. The second issue is that the first <word> element does not have a closed tag.

2.8 Disadvantages of XML

Disadvantages of XML in comparison to its advantages are not considerable. However, like other tools and technologies, no matter how useful they might be, some drawbacks can be identified.

First, rendering XML is still not something like HTML documents. Although most of the browsers in the market can render an XML document in a well-structured, tree-like, shape, some may not do this properly. So, the tools and techniques still are growing around XML. For a list of web browsers that support XML-based documents consult http://www.xml.com/pub/rg/XML_Browsers. However, the best way to understand the level of support that each browser offer, is to test an XML document with the target browser. Furthermore, keep in mind that a most of browsers are updated regularly. As a result, their XML support feature might change accordingly.

Second, XML is easy if one uses it at a simple level. It becomes a bit complicated if it is used in a fully-fledged manner. Other technologies around XML, such as XML Stylesheet, XML Schema, XML Query, and XML Writing and Reading tools within the applications can take quite a long time to be fully understood.

Last but not least, XML security is still a problem. Although there are tools and techniques to overcome this issue, it adds to the complexity of the technology as well.

2.9 When to use XML?

XML can be used for variety of purposes. The usage range varies from scientific/technical purposes to ordinary business applications. Yet, XML should not be taken as an ultimate answer to everything. It may perform very well in some circumstances and poorly in some others. Below are some rule of thumb for when to choose XML as a data container.

When to choose XML as a data container?

- XML works well with a data that has the following characteristics:
 - o Immutable data – Data that is not changed (such as dictionaries).
 - o Intended for frequent read but is rarely updated (such as library catalogs).
 - o Has very complex format (such as computers/applications configuration files).

Tips on using XML

- XML is mainly a technology to be used to organize/standardize data exchanges
- Consider using XML as a data container in if:
 - o Your data is semi-structured or unstructured
 - o You do not have heavy update on your database
 - o Security is not the main concern
 - o You do not deal with a large amount of data
- Ignore all the above and use XML if you are not obliged to use a traditional database! This is an opportunity for you to practice an amazing software technology.

3 XML Validity

In the previous chapter, the “well-formedness” of an XML document was presented as a concept that addresses the syntax correctness of the document. But, it was pointed out that the well-formedness only investigates a document by considering its structure. That is, beyond the structural format, it does not check any specific other conditions that the XML composer might be interested in applying it to some or all elements.

For example, how can we tell that the dictionary document must have at least one <word> element? Or how can we tell what kind of data the element can accept? These kind of checks are important, not only because they help the composer to create a proper document, but also in sharing data. In the latter case, different parties that share a data might agree on a specific format, based on which they can control whether they have received and/or generated a right (valid) data. The inventors of XML introduced an accompanying structure in order to allow the mentioned checks to be performed. Then named this structure Document Type Definition (DTD).



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.



Click on the ad to read more

3.1 Document Type Definition (DTD)

DTD allows XML composer to define a valid syntax for an XML document. It can be included in the XML document or saved as a separate file. In the latter case, the extension of the file should be “dtd”.

Document Type Definition (DTD)

- DTD helps to define the following:
 - o List of element names that can occur in an XML document
 - o The way that elements that can appear in combination with other elements
 - o The structure of nesting elements
 - o The name of the attributes, which are available for each element type

Note – Term “vocabulary” sometimes used to refer to the elements used in a particular application.

DTD uses an EBNF (Extended Backus-Naur Form) grammar, not XML. This grammar is a context-free grammar. The subject of context-free grammars are beyond the scope of this book. What is necessary here is to understand how we are able to construct a correct DTD and how we are able to check the **validity** of an XML document against a DTD.

As it was mentioned, an XML document is meant to be self-descriptive, hence having a DTD is optional. However, if you are planning to share the document and you want to make sure that the document conforms a certain syntax then it is better to compose a proper DTD to accompany your XML document. This way all parties who use the document have a mean by which they can check whether the document meets certain syntax rules or not.

How to build a DTD?

- General form of DTD:
 - o **<!ELEMENT** root (element1, element2, ...)>
 - o **<!ELEMENT** element1 (**#ELEMENT TYPE**)>
 - o **<!ELEMENT** element2 (**#ELEMENT TYPE**)>
 - o ...
- It starts with **<!DOCTYPE** root
- The rest of the definition is enclosed in [].

(Developing SGML DTDs – From Text, To Model, To Markup) by Eve Maler and Jeanne El Andaloussi provides a complete reference on DTDs. It can be found at <http://www.xmlgrrl.com/publications/DSDTD/index.html>

3.2 Element Type Declaration

When you define an element in a DTD, its type must be declared. Declaring this type means that certain rule must be applied where an element is presented in an XML document. As it was mentioned, an element can be repeated in an XML document. Indeed this is the way that multiple occurrences of an element are presented. Element type declaration allows you to specify the rules that should be followed for element repetition.

3.2.1 Element Repetition Rules

Table 1 shows the element repetition rules, the corresponding symbols that must be used in the declaration, and an example of their usage.

Repetition Rule	Symbol	Example
Zero or more	*	<!ELEMENT sampleDic (word)*>
One or more	+	<!ELEMENT word (head, definition+)>
Either zero or exactly one	?	<!ELEMENT word (head, definition+, source?)>
Exactly one		<!ELEMENT word (head, definition)>
Elements must come in specific order	,	<!ELEMENT word (head, definition+)>
Order of occurrence does not matter	Space	<!ELEMENT word (head definition+)>
Either/Or		<!ELEMENT word (head, (definition sence)+)>

Table 1 – DTD – element repetition rules

3.2.2 Element Types

The types of data of the elements of an XML can be defined in the correspondent DTD.

Table 2 shows different types that can be assigned to elements.

Type	Description	XML Parser Action
#PCDATA	Parsed Character Data	The data would be parsed.
#CDATA	Character Data	No parsing happens.
ANY	Any combination of parsable data	Any element with any content.
EMPTY	Empty Element	An empty element is allowed.

Table 2 – DTD (Element Types)

Below the usage of these types is explained through an example.

```
<!ELEMENT sampleDic (letter*)>
<!ELEMENT letter (word*)>
<!ELEMENT word (head, definition*, synonym*, antonym*)>
<!ELEMENT definition (sense+, description*)>
<!ELEMENT sense (sorder, stext)>
<!ELEMENT description (dtext, usage+)>
<!ELEMENT usage (utext, source+)>
<!ELEMENT synonym (synorder, syntext)>
<!ELEMENT antonym (antorder, anttext)>
<!ELEMENT head (#PCDATA)>
<!ELEMENT sorder (#PCDATA)>
<!ELEMENT stext (#PCDATA)>
<!ELEMENT dtext (#PCDATA)>
<!ELEMENT utext (#CDATA)>
<!ELEMENT source (#CDATA)>
<!ELEMENT synorder (#PCDATA)>
<!ELEMENT syntext (#PCDATA)>
<!ELEMENT antorder (#PCDATA)>
<!ELEMENT anttext (#PCDATA)>
```

Figure 8 – DTD example

A APOLLO HOTEL

CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired



Click on the ad to read more

In the above example, some elements' types have been declared as #PCDATA and some as #CDATA. The reason is, for example, I want the XML sub-elements "order" and "text" for the "sense" to be processed by XML and to not contain any illegal symbols such "<" and "&". However, as the "usage" examples of a "sense" might include any character including the mentioned symbols, that are interpreted by XML parser in a different way, they have been declared as #CDATA, in which case whatever is written as the value of those elements would be sent to the output intact (unparsed). In other words, CDATA instructs XML processor to ignore markup characters and pass enclosed text directly to application. This will be shown through examples in the coming sections.

3.3 Attribute List Declaration

As it was mentioned, attributes are information that appear in an element. To keep the validity of the XML document, attributes that are intended to be used in the document should be declared using an "attribute list declaration".

An attribute can be declared using the following syntax:

```
<!ATTLIST element-name attribute-name attribute-type default-value>
```

The syntax has the following parts:

- element-name – This is the element within which the attribute appears.
- attribute-name – This is the name that is assigned to the attribute.
- attribute-type – This is the type of the attribute.
- default-value – This is the default value of the attribute.

Attribute *default-type* can be one of the items that are shown in Table 3.

Type	Description
CDATA	The value is of Character Data
ENTITY	The value is an Entity, i.e. an abbreviation or a reference to an external location.
ENTITIES	The value can be a multiple Entities, which are separated using white spaces.
(en1 en2 ...)	Enumerated list; the attribute value must be one item of the list.
ID	The attribute has a fixed value, which cannot be changed throughout the XML document.
IDREF	The attribute is a reference to another element.
IDREFS	The attribute includes several references, each separated by a whitespace, to several other elements.
NMTOKEN	The value must be a valid XML name.
NMTOKENS	A list of NMTOKENs separated by whitespace.
NOTATION	The value is the name of a notation.
xml:	The value is a predefined XML value.

Table 3 – DTD (Attributes)

Attribute *default-value* can be one of the items that are shown in Table 4.

Type	Description
#DEFAULT	The attribute has a default-value
#FIXED value	The attribute has a fixed value
#IMPLIED	Providing attribute is optional
#REQUIRED	The attribute must be provided alongside element

Table 4 – DTD (Attributes)

DEFAULT – If used it means that there is a default value for an attribute in which case even if it is forgotten in the XML, it receives this default value through the “dtd”.

FIXED – If used it means that the attribute is assigned a fixed value, the one that is given in the ATTLIST, and it cannot be changed in the XML document.

IMPLIED – If used it means that the attribute does not have a default value. It also means that including this attribute is not obligatory.

REQUIRED – If used it means that the attribute does not have a default value, but its presence in the element is obligatory.

3.4 When to Use Attributes?

A general suggestion for using attributes is this:

“Avoid using attributes until it is absolutely necessary!”

In other words, use attributes when they add more readability to your document, and avoid using them when they cause confusion.

Several reasons are behind this suggestion. First, you can do whatever you need to do with a normal XML document compilation, without adding more confusion into it. Second, attributes are not expandable and they cannot have a tree structure. Finally, attributes cannot hold multiple values. You might comment on this latter reason by addressing NMTOKENS, IDREFS, and Enumerated. Indeed, in all those cases, the attribute can take one value among many, at the end. Therefore, the suggestion is worth taking it.


```

<!ELEMENT sampleDic (letter*)>
<!ELEMENT letter (word*)>
<!ELEMENT word (head, definition*, synonym*, antonym*)>
<!ELEMENT definition (sense+, description*)>
<!ELEMENT sense (sorder, stext)>
<!ELEMENT description (dtext, usage+)>
<!ELEMENT usage (utext, source+)>
<!ELEMENT synonym (synorder, syntext)>
<!ELEMENT antonym (antorder, anttext)>
<!ELEMENT head (#PCDATA)>
<!ELEMENT sorder (#PCDATA)>
<!ELEMENT stext (#PCDATA)>
<!ELEMENT dtext (#PCDATA)>
<!ELEMENT utext (#CDATA)>
<!ELEMENT source (#CDATA)>
<!ELEMENT synorder (#PCDATA)>
<!ELEMENT syntext (#PCDATA)>
<!ELEMENT antorder (#PCDATA)>
<!ELEMENT anttext (#PCDATA)>

<!ATTLIST head part-of-speech #PCDATA #REQUIRED>

```

Figure 9 – Attributes and Attribute List Declaration





Max's next Bookboon eBook
Your Boss: Sorted!
 By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com



3.5 Document Validity

As we introduced the concept of well-formedness in the previous sections, XML documents can be *validated* in order to assure that the document not only followed the rules of XML document generation but also it is complied with an agreed format for its elements. This validity is an important part of data exchange and sharing. That is, by preparing a DTD that acts as a protocol for checking the compatibility of exchanged (sharing) XML documents. It also helps the XML composers to have the prior knowledge about how to compile their documents.

An XML document is considered to be *valid* if its definition follows the rules that have been presented in its associated DTD.

3.6 Using DTD

DTD can be used in two forms, as a part of XML files, or as a separate file. Both methods are explained in the following sections.

3.6.1 Using DTD Inside an XML Document

DTD can be used as part of the XML document. Figure 10 shows this method. However, using DTD in the XML document restricts its usage to that specific document only. This is somehow in contradiction to the DTD purposes, which are to define a base for the validation of the XML documents and to act as a guideline to compile and generated different documents.

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

<!DOCTYPE TVSCHEDULE [
  <!ELEMENT sampleDic (letter*)>
  <!ELEMENT letter (word*)>
  <!ELEMENT word (head, definition*, synonym*, antonym*)>
  <!ELEMENT definition (sense+, description*)>
  <!ELEMENT sense (sorder, stext)>
  <!ELEMENT description (dtext, usage+)>
  <!ELEMENT usage (utext, source+)>
  <!ELEMENT synonym (synorder, syntext)>
  <!ELEMENT antonym (antorder, anttext)>
  <!ELEMENT head (#PCDATA)>
  <!ELEMENT sorder (#PCDATA)>
  <!ELEMENT stext (#PCDATA)>
  <!ELEMENT dtext (#PCDATA)>
  <!ELEMENT utext (#CDATA)>
  <!ELEMENT source (#CDATA)>
  <!ELEMENT synorder (#PCDATA)>
  <!ELEMENT syntext (#PCDATA)>
  <!ELEMENT antorder (#PCDATA)>
  <!ELEMENT anttext (#PCDATA)>
]>
```

continued on the next page

```
<sampleDic>
  <word>
    <head> ethic </head>
    <definition>
      rules of behavior based on ideas about what is morally good and bad.
    </definition>
    <source>Merriam-Webster, online </source>
    <usage>Ethics is his chosen field of study.</usage>
  </word>
  <word>
    <head> moral </head>
    <definition>
      concerned with the principles of right and wrong behavior and the
      goodness or badness of human character.
    </definition>
    <source>The Internet, online</source>
    <usage>
      the moral dimensions of medical intervention
    </usage>
  </word>
</sampleDic>
```

Figure 10 – Using DTD inside the XML document

3.7 Using DTD as a Separate File

DTD can be used as a separate document in which case it must be referred to within the XML document. This way, any XML document that is intended to follow the same rules for their generation can refer to it as a base for validity checking. Figure 11 shows this method, assuming the DTD has been saved in a separate file named “SampleDic.dtd”. If you are planning to have several XML documents that follow the same DTD, certainly this method would more efficient than the previous one. However, the previous method has its own application when it is desired to have the DTD close to the XML document.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sampleDic SYSTEM "SampleDic.dtd">

<sampleDic>
  <word>
    <head> ethic </head>
    <definition>
      rules of behavior based on ideas about what is morally good and bad.
    </definition>
    <source>Merriam-Webster, online </source>
    <usage>Ethics is his chosen field of study.</usage>
  </word>
  <word>
    <head> moral </head>
    <definition>
      concerned with the principles of right and wrong behavior and the
      goodness or badness of human character.
    </definition>
    <source>The Internet, online</source>
    <usage>
      the moral dimensions of medical intervention
    </usage>
  </word>
</sampleDic>

```

Figure 11 – Using DTD as a separate document

If it is checked for validity, the XML document in Figure 11 is considered a valid document, because it is compatible with the stated DTD. The check is not easy to be performed manually. There are different tools, which can be used for both well-formedness and validity. Below are two handy tools that can be used. Both tools are open source and able to check the well-formedness and validity of XML documents:

- XML Copy Editor – A simple open source tool that allows you to create XML documents, DTDs, XML Schemas and more.
- NetBeans – An IDE, essentially for Java programmer, which is able to do a variety of functions.

4 Using XML

Different technologies have been developed around XML. Each one of these technologies serves different purposes. We will introduce the most common ones in this and the following chapters.

To use XML in a software system, one needs to use an Application Program Interface (API). The reason is, also the XML document, as we mentioned before, is human readable, needs a tool to make it understandable by machines. This tool is a “parser” that is able to traverse the tree-structure of the XML document and provide a presentation that can be dealt with by a software. These tools (APIs) are divided into two forms:

- Tree-based
- Event driven

The following sections explain these methods.



MTHøjgaard

BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang



4.1 Tree-based XML API

As the name implies, this type of API reconstructs the XML tree, as it is presented in an XML file. That is, a complete tree representation of the XML is built in the memory. W3C has provided a standard for traversing documents such as XML and HTML. This standard is called “Dynamic Object Model” or DOM. DOM has three levels, namely, Core DOM, XML DOM, and HTML DOM.

4.1.1 XML DOM (Dynamic Object Model)

XML DOM is standard object model and API for XML, which is platform-independent and language-neutral. It performs as below:

- Creates the XML as a tree structure in memory.
- The tree is seen as a compound structure of nodes.
- An object-oriented interface allows the tree to be traversed and processed.
- It works based on the concept of *node*.
- A *node* might be:
 - The XML document – *document node*
 - Each element – *element node*
 - Texts in element – *text node*
 - Attributes – *attribute node*
 - Comments – *comment node*

Figure 12 shows an XML DOM tree and the definitions of its nodes.

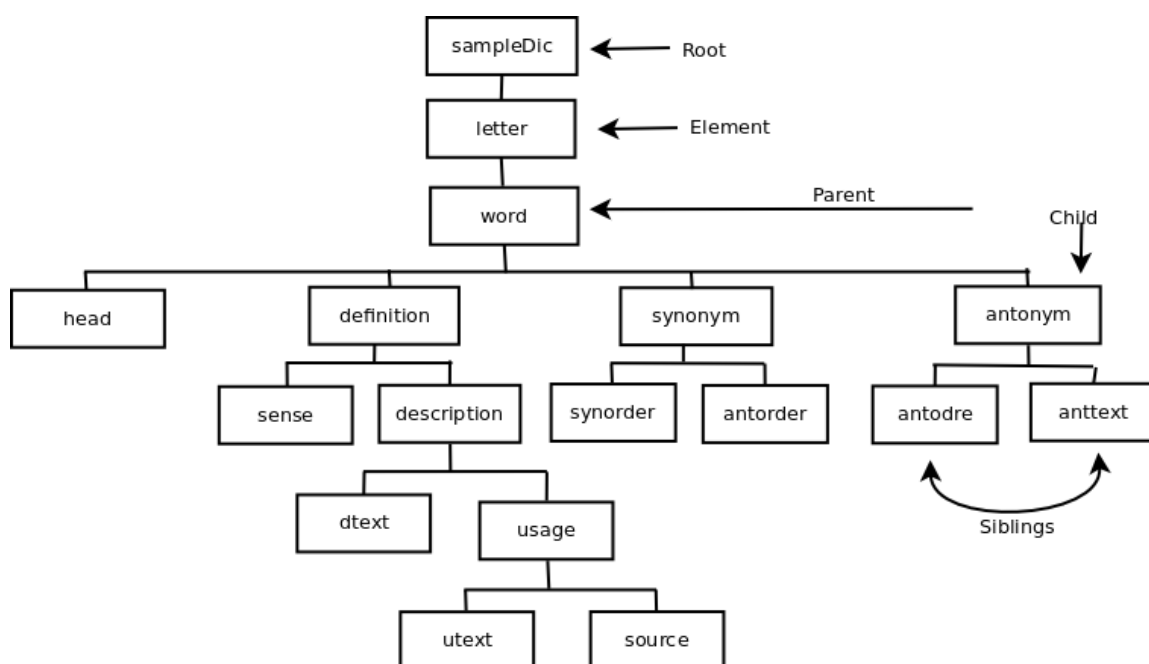


Figure 12 – XML DOM Example

The created tree by XML DOM is called a *node-tree*. As it is shown in Figure 12 the nodes follow a “family” sort of relationship. That is:

- There is a *root*, that is the final *parent* of all nodes.
- A node can be a *parent* for some children.
- Every *child* has a *parent* (only one parent).
- *Root* is not a *child*, hence does not have a parent.
- The children of one parent are called *Siblings*.
- A node with no children is called a *leaf*.

As it was mentioned, there is an API, based on XML DOM, that provides the interface by which the XML tree can be traversed and processed. As this interface follows an object-oriented approach, it is composed of a set of *attributes* and a set of *methods*. The *attributes* hold the information about the node-tree and the *methods* provide functions, such as *append* and *remove* that can be performed on the *attributes*.

In fact, when one intends to process an XML document using XML DOM, one should follow the traditional way that is followed in object-oriented programming. That is to create a DOM object and to deal with it in a way that they do in object-oriented approach.

Creation of large objects might be costly, from both memory size and time-to-create perspectives. Therefore, the XML DOM, regardless of its capability to handle the structural view of an XML document, in many cases might not be the best choice for programmers. To overcome this issue, an event-driven API has been developed. The following section discusses this approach.

4.2 Event Driven XML API

As the name implies, this interface traverses the XML document based on a series of events instead of creating the whole tree in the memory. This method was not developed by W3C. It was developed by a community of XML developers. OASIS international, a non-profit standards consortium, hosts xml.org, which is responsible for “advances the use of open standards by providing educational information, collaborative resources, and discussion areas”. It maintains the “XML-DEV mailing list”, which was responsible for developing an event-driven API for XML processes (<http://www.xml.org/xml-dev>). We will look at this API in the following section.

4.2.1 Simple API for XML (SAX)

“XML-DEV mailing list” developed a Simple API for XML (SAX). SAX is a universally accepted *de facto* standard for XML parsers. SAX is an event-driven parser. Instead of creating the whole XML tree in memory, it keeps the minimum required data that is necessary for parsing and traversing process. This amount of memory is considerably less than the amount that is required by DOM APIs. The first release of SAX, SAX 1.0, was released in 1998. Consequently, SAX 2.0.1 was released in 2001, which is still a current version.

SAX uses *callbacks* to report *events*. *Callback* function is a function that is called through a pointer. That is, when one sends an address of a function as an argument to another function, which in turn calls the targeted function via the passed pointer, a *callback* happens. The function that is called via the passed pointer, is a *callback function*. *Event* is something that is happening.

In an XML parsing case, events are specific situations that the parser encounters, such as detecting the “start of document”, “start of element”, “end of element”, and “characters” (or raw text). When such an event happened, the parser callback system, would call a function that has been developed by the API user. Different implementation of SAX exist.

Figure 13 shows an example of SAX usage.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den 9. og 10. oktober 2019.

Vi giver en is og fortæller om jobmulighederne hos os.

banedanmark




```

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ReadXMLFile {
    public static void main(String argv[]) {
        try {
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
            DefaultHandler handler = new DefaultHandler() {

                boolean bletter = false;
                boolean bword = false;
                boolean bhead = false;
                boolean bsorder = false;
                boolean bstext = false;

                public void startElement(String uri, String localName, String xName,
                    Attributes attributes) throws SAXException {
                    System.out.println("Start Element:" + xName);
                    if (xName.equalsIgnoreCase("letter")){
                        bletter = true;
                    }
                    if (xName.equalsIgnoreCase("word")){
                        bword = true;
                    }
                    if (xName.equalsIgnoreCase("head")){
                        bhead = true;
                    }
                    if (xName.equalsIgnoreCase("sorder")){
                        bsorder = true;
                    }
                    if (xName.equalsIgnoreCase("stext")){
                        bstext = true;
                    }
                }

                public void endElement(String uri, String localName,
                    String xName) throws SAXException {

                    System.out.println("End Element:" + xName);
                }

                public void characters(char ch[], int start, int length) throws SAXException {

                    if (bletter) {
                        System.out.println("Letter:" + new String(ch, start, length));
                        bletter = false;
                    }
                }
            };

```

continued on the next page

```

        if (bword) {
            System.out.println("Word:" + new String(ch, start, length));
            bword = false;
        }

        if (bword) {
            System.out.println("Word:" + new String(ch, start, length));
            bword = false;
        }

        if (bsorder) {
            System.out.println("Order:" + new String(ch, start, length));
            bsorder = false;
        }
    }

};

saxParser.parse("SampleDic-3.xml", handler);

} catch (Exception e) {

    e.printStackTrace();

}

}

```

Figure 13 – SAX Example (Java SAX)

As you can see from the code in Figure 13, the Java SAX API uses the following methods:

- startDocument() – This method is called when the start of XML document is detected.
- endDocument() – This method is called when the end of XML document is detected.
- startElement() – This method is called when the start of an element is detected.
- endElement() – This method is called when the end of an element is detected.
- characters() – This method is called with passing the text that is located between a start-tag and end-tag as an input parameter.

Note – One might find that it was much readability if the above code was written using “switch” statement. As “switch” statement doesn’t support “string” variables as the condition variable in Java prior to 1.7, I preferred to use a general “if then else” statement in order to let the code to be compiled and run on different Java versions.

4.3 DOM versus SAX

DOM or SAX? Which one should be our choice? The decision depends on several parameters as below:

- Size of the XML documents
- Number of XML documents
- The computing resources, particularly, memory
- The expected speed of data retrieval
- The nature of XML document: complex versus simple
- The computing process nature: update versus read

Below is some tips on choosing one of the APIs. However, no one can forbid you from using both if the analysis tells you to do so.

DOM or SAX
Choose DOM if:

- Your XML structure is complex and your computing system has a large amount of memory.
- You need to update the XML (in memory).
- You need to deal with XML nodes as objects.

Choose SAX if:

- The initial data retrieval is not important.
- You do not need to update the XML data in memory.
- You don not need to have the entire structure in memory.
- Your computing system has a limited memory resources.
- Your application needs to deal with the data based on their occurrence and events that they trigger when they are parsed.

Choose a combination if:

- You have a combination of the above situations.

4.4 Namespace

As it was mentioned, XML documents are easily developed for different applications using a simple editor. The commonality of XML makes XML documents very vulnerable to name conflicts. In other words, as it is possible for using the same name in different contexts, it is quite possible for XML documents to use an element name for specifying different things. This would cause a name conflict when one wants to merge these documents.

Figures 14 and 15 show a name conflict.

```
<word>
  <head> ethic </head>
  <definition>
    rules of behavior based on ideas about what is morally good and bad.
  </definition>
</word>
```

Figure 14 – XML – Sample 1 (name conflict)

```
<word>
  <filename> test.doc </filename>
  <catefory> Technical Report 1 </category>
</word>
```

Figure 15 – XML – Sample 2 (name conflict)

The reason for the conflict is because the element “word” has been defined in two different ways in these documents, where neither their sub-elements names nor their contents match. If one tries to merge these two documents, they would receive an error indicating that a name conflict has happened that cannot be resolved.

In order to resolve these kind of conflicts, XML uses a mechanism that is called **namespace**. Programmers in some programming languages such as C++, C#, and Python are already familiar with the mechanism. In some other languages, for example, Java, the mechanism is applied through the concept of modules.



CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired

A *namespace* is simply a label to an environment, including addresses, module names, file names, file locations, and such, within which a “name” is unique. *Namespace* allows programmers to make sure that certain names are unique across an application and not just in a single program. In other words, a *namespace* allows programmers to define a set of names without any ambiguity. In programming language it is used as a scope resolution as well.

The following sections explain the mechanism in the XML context. But before jumping into the XML namespaces subject, let us have a look at a method of resolving the raised name conflict.

4.4.1 Name Conflict Resolution using a Prefix

An straightforward method to resolve naming conflict is to use a prefix to the names. For example we can use two prefixes for the previous examples to avoid the name conflict. This method is shown in Figures 16, 17.

```
<dic:word>
  <head> ethic </head>
  <definition>
    rules of behavior based on ideas about what is morally good and bad.
  </definition>
</dic:word>
```

Figure 16 – XML – Sample 1 (using prefix for name conflict resolution)

```
<proc:word>
  <filename> test.doc </filename>
  <category> Technical Report 1 </category>
</proc:word>
```

Figure 17 – XML – Sample 2 (using prefix for name conflict resolution)

The prefixes “dic” and “proc” now qualify the words, providing two new names that are different from each other, hence they have no conflict any more. However, we have to specify how the parser could find “dic”, and “proc”.

4.4.2 XML Namespace

Technically speaking, we should specify a *namespace* where these prefixes could be found.

In XML the *namespace* is defined by using an attribute that is called **xmlns** (to remember it, you can see it as a short name for “XML namespace”). Below are the main rules for XML namespace definition:

- The value of the attribute specifies the location that the namespace could be found.
- The namespace attribute can appear in the start-tag of an element, in which case all children of that element would be associated to the same name space.

- The namespace can appear in the *root* element, in which case the whole document would be associated to the defined namespace.
- As **xmlns** is an attribute, it can have several prefixes making each one unique, hence it is possible to define several namespace within a tag.
- The children (sub-elements) should be prefixed by the defined *prefix*.

XML Namespace

Syntax:

xmlns:prefix="URI"

prefix: is any valid string.
URI: is an abbreviation for "Uniform Resource Identifier", which must be a valid Internet Resource. It is called URN – Universal Resource Name – as well.

Example:

xmlns="http://www.w3.org/HTML/1998/html4"

4.4.3 Default Namespace

A useful way to avoid repeating prefixes in front of every child element is to define a default namespace. Default namespace does not have any prefix.

Default XML Namespace

Syntax:

xmlns="URI"

Example:

xmlns="http://www.w3.org/1999/xhtml"

In the above example all children implicitly use the define namespace that is given in the start-tag of the XML element.

Figure 18 shows an example of the namespace application.

```
<dic:word>
  <head> ethic </head>
  <definition>
    rules of behavior based on ideas about what is morally good and bad.
  </definition>
</dic:word>
```


Figure 18 – XML namespace – Example

5 XSL – eXtensible Stylesheet Language

Most of the readers, who have already developed web applications, are familiar with the concept of CSS (Cascading Style Sheet). CSS is a useful way to predefine some tags that allow HTML documents to be able to uniformly display the intended material. However, HTML uses predefined tags that are interpreted by browsers in the same manner. CSS helps the colors, fonts, and other properties to be unified.

Unlike HTML, XML does not have predefined tags. In fact, one of the main aims of XML is to allow users to define their own tags. Accordingly, XML does not have a predefined way to render different tags. Therefore in majority of browsers, XML documents would be displayed as they are defined in a text document though there might be a “luxury” of “collapse-expand” signs, that are normally shown with “+” and “-” in order to allow to expand or to collapse selected elements or the entire document. As a result, W3C developed a Style Sheet for XML documents called eXtensible Stylesheet Language, abbreviated as XSL.

XSL is not a single tool or language. It consists of several parts together, which allow users to define the way that they want their XML documents to be rendered and displayed. The following sections will explain XSL and its different parts.



Max's next Bookboon eBook
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com



Click on the ad to read more

5.1 XSLT – XSL Transformation

As we mentioned, XML does not provide any specific way for its information to be rendered and displayed. Therefore, the fundamental part of XSL was developed in order to satisfy this necessity. XSL Transformation (XSLT for brief) is a language that allows XML developers to define the way that their XML document could be “transformed” into another format, such as HTML, that is recognizable by browsers. XSLT is also able to transform an XML document into another XML document.

XSLT offers several functionalities such as:

- Add elements to the output file.
- Delete elements from the output file.
- Sort elements.
- Test elements and make decisions (e.g., to show/hide some elements).

But before we discuss the XSL further, a tool and notion must be explained: XPath, which allows users (in this case XSLT) to retrieve elements of an XML document.

5.1.1 XPath

XPath was developed based on **path expressions**. A *path expression* is a method in query languages that allows navigation into an object. The most common example of a *path* is what you have seen when you specify a location of a file on a computer disk.

For example:

- “/home/user1/temp/xmlsample.xml” in UNIX environment, or
- “C:\home\user1\temp\xmlsample.xml” in a Microsoft Windows environment.

The same idea is used for defining *paths* in XSL tools. XPath provides different functionalities. It offers more than 100 built-in functions. The functions are organized in different categories as below:

- Different types of value comparisons
- Boolean value
- Sequence manipulation
- ...

XPath has its own terminology which can be found below.

XPath Terminology

- **Node** – A general term for the following objects:
 - element
 - attribute
 - text
 - namespace
 - processing-instruction
 - comment
 - document nodes
- **Atomic Value** – A node without any parent or children.
- **Item** – A node or an atomic value.
- **Parent** – An element or attribute has one parent.
- **Children** – An element may have zero, one, or more children.
- **Siblings** – Nodes with the same parent.
- **Ancestors** – A node's parent, parent of parent, and so on.
- **Descendants** – A node's children, children of children, and so on.

To retrieve information, XPath uses the path expressions as shown in Table 5.

Expression	Description
<i>nodename</i>	Select all nodes with the name <i>nodename</i>
/	<ul style="list-style-type: none"> • Select the root node • It also acts as a separator between different parts of a path expression
//	Select
.	<ul style="list-style-type: none"> • Select the context node • Select the current node
..	Select the parent of the current node
@	Select attribute
*	Select unknown element (matches any element node)
@*	Select unknown attribute (matches any attribute node)
node()	Select unknown node (matches and kind of node)
[<i>predicate</i>]	<p>Finds a node that satisfies a specific condition, according to the <i>predicate</i>.</p> <p>Some functions such as <i>last()</i> and <i>position()</i> can be used in predicates.</p>

Table 5 – XPath Expressions

5.1.2 XPath Axes

XPath has another powerful feature that allows users to retrieve a node-set based on their relationship to the current node. This feature is called **axis**. W3C defines an axis as a node-set relative to the current node. Table 6 shows XPath axes.

Axis	Result Node-set
<i>self</i>	Current node
<i>parent</i>	Parent of the current node
<i>child</i>	All children of the current node
<i>attribute</i>	All attributes of the current node
<i>descendant</i>	All children and children of children and so on of the current node
<i>following</i>	All nodes after the closing tag of the current node
<i>preceding</i>	All nodes before the current node, except ancestors, attributes, and namespaces.
<i>ancestor</i>	All parent, parents of parent, and so on of the current node
<i>descendant-or-self</i>	Descendant + the current node
<i>ancestor-or-self</i>	Ancestor + the current node
<i>following-sibling</i>	All siblings after the current node
<i>preceding-sibling</i>	All siblings before the current node
<i>namespace</i>	All namespace nodes of the current node

Table 6 – XPath Axes



 MTHøjgaard

BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang



Notice to the difference between *descendant* and *following*. The first provides the children and children of children and so on, but not those nodes that are not “descending” of the current node. While the latter, provides everything that are coming after the close tag of the current node. This difference has been shown below.

5.1.3 XPath Operators

XPath supports several operators that allows complex expressions to be devised.

XPath Operators

+, -, *, div, =, !=, <, <=, >, >=, or, and, mod, |

The operators are self-explanatory. The only one that might need explanation is '|', which retrieves (computes) two node-sets.

To summarize, XPath is considered a *declarative language*, from programming languages perspective. That is, by using XPath one asks a computer *what* one wants to be done with an XML document, but without specifying *how* to do it. XPath plays a central role in many XML technologies such XPointer, XSL, XML Schema, and XQuery. This role will be discussed in the following sections and chapters.

5.2 XPointer

XPointer is another XML supporting tool. It also facilitates the XML information retrieval. It basically uses XPath for its information retrieval. In fact, XPointer has been developed upon XPath. However, it has some extra facilities, which gives more simplicity in practice. This is mainly accomplished by utilizing URIs, which we discussed in section 4.4.2. Simply, an XPointer can be considered as an XPath expression that appears in a URI.

XPointer allows user to link to sections of text, select particular elements or attributes, and navigate through elements. With XPointer one can also select data contained within more than one set of nodes, which cannot be done with XPath.

5.3 XLink

You are probably familiar with the “link” concept in HTML documents or if not in HTML in other text documents, whereby one can establishes a “link” to a source that is either located on the local computer or somewhere else, on the Internet, or your private network. XLink has utilized the mentioned concept, and has generalized it to a level that serves to handle more complex and sophisticated links, not just into the hypertexts, but also into XML documents in general.

XLink, uses XPointer for addressing and adds more computational facilities that allow elements to be inserted into XML documents and to create “links” between variety of resources. XLink, as it is a general approach with almost all XML technologies, uses the XML syntax to create links.

Two types of XLink exist:

- Simple XLink – connects a source to a destination resource.
- Extended XLink – connects any number of resources.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.

banedanmark



6 XML Schema

In chapter 3 we introduced the concept of XML validation through Document Type Definition (DTD). Despite simplicity DTDs were considered as strong tool in XML validation and it had a great role in keeping uniformity among different parties who intended to share information using XML documents. However, using DTD imposes several restrictions on the user as below:

- DTD has a non-XML syntax.
- Its data types are very restricted.
- It does not support namespaces.

W3C has developed XML Schema in order to provide a more sophisticated and comprehensive technology through which users are able to describe the structure of XML documents. W3C has recommended this technology under the name of **XML Schema Definition (XSD)**.

As the name implies, one can look at XML Schemas as a counterpart of *schemas* in the Database design context. As you know, in the database context, schemas have been widely used. In this context, a schema is a formal way of defining a database and a basis by which one is able to create a database. Also, a *relation schema* is the formal definition of a relation (a table, in other words), through which one can create a *relation* and define its characteristics. Schemas have a central role in databases and provide a basis for modification, manipulation, and information retrieval in databases.

XML Schema, by analogy, plays a similar role in XML technology. The characteristics of XML Schema and its ability for sophisticated definitions of XML structures allow applications to share and exchange XML data in a well-managed manner. XML schema's fundamental roles, for a specific XML document, are:

- To define the organization of the XML structure. For example, this includes the way that elements, children, and attributes must appear in the document. Also whether an element could be empty.
- To define data types that are supposed to be used in the XML document.
- To define the elements that can be used in the XML document.
- To specify the data type that each element must have.
- To specify any fixed and possible default values for elements and attributes.
- To define other criteria such as *case sensitivity* for element-names.

Like almost all XML technologies, XML Schema itself is an XML document, therefore, it can be created, edited, and processed in the same way and by the same tools that are used for creation, edition, and process of XML documents.

However, there are some critical opinions about XML Schema, pointing to its complexities and technical issues (for example, see <http://www.informit.com/articles/article.aspx?p=367637&seqNum=2>).

In the following sections, the different parts of XML Schema will be explained.

6.1 The <schema> Element

Every XML Schema starts with a <schema> element as its root. It can have some attributes if required.

6.2 Simple Elements and Simple Types

A simple element indicates that the element cannot contain anything but text, when it appears in the XML document. This means that the element cannot include any other elements or attributes. However, it does mean that it only includes a “text” in its traditional meaning. In fact, you should be careful with the understanding of the concept, because a simple element can be a string, a decimal, an integer, a date, a time, a boolean, and such. Later in this chapter, you will see that you can add some restrictions and patterns that makes the definition of simple types very robust and sophisticated.

Elements and attributes are of some types. The fundamental XML Schema types are: simpleType and complexType. Users also able to define their own types based on these to fundamental types.



The advertisement features a night-time photograph of the Apollo Hotel building. Overlaid on the image is a red lightbulb icon with the text 'CISO Conference' and 'Produced by Inspired'. A white text box on the right provides the event details: 'Apollo Hotel 1, Groenlandsekade Vinkeveen, Amsterdam, NL' and 'Dec 5th 2019'. At the bottom, a white banner contains the text 'Listen, learn & build relationships with our Network of CISOs & Cyber Security Leaders' and the 'Inspired' logo.

CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired

The simple elements can only take **simpleType**.

XML Schema – Simple Element
Syntax:

`<xs:element name="name" type="xs:simpleType"/>`

Attributes must be defined last:

`<xs:attribute name="attribute" type="xs:simpleType"/>`

simpleType:

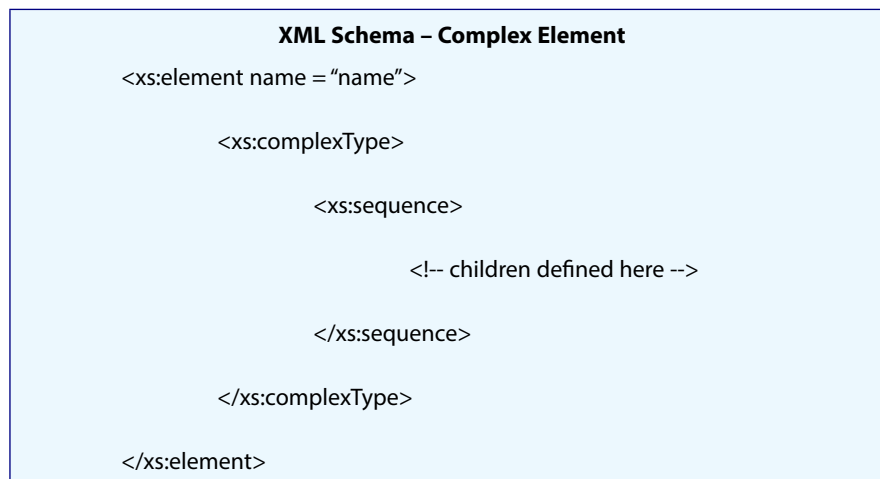
- Numeric:
 - string
 - decimal
 - float
 - double
 - ...
- String
 - string
 - token
 - NMTOKEN
 - ...
- Duration
 - time
 - date
 - datetime
 - ...
- Others
 - boolean
 - ID
 - IDREFS
 - ...

Refer to <http://www.w3.org/TR/xmlschema-2/> and <http://www.xmlschema-reference.com/builtInDatatype.html> to find more on XML Schema built-in data types.

6.3 Complex Elements and Complex Types

Unlike simple elements, complex elements are elements that contain other elements and/or attributes. They are defined using **complexType** element.

List of children of complex type are described by sequence element.



6.4 XML Schema Indicators

XSD's general syntax includes an option that is called *indicator*.

There are three categories of indicators as below:

- Order indicators
- Occurrence indicators
- Group indicators

The occurrence indicators are very common in XML Schemas. This type of indicators will be explained in the next section. The interested users can refer to the following address, for more information on other types of indicators.

http://www.w3schools.com/schema/schema_complex_indicators.asp

6.4.1 Cardinality

As the relationships among elements can be defined in an XML Schema, the technology offers a concept similar to the concepts that is used in database schema, for defining the **cardinality** of an element.

Cardinality is defined through *occurrence indicators*. Occurrence indicators are used to define cardinalities of elements through the following attributes:

- *minOccurs*
- *maxOccurs*

XML Schema – Element Cardinality

- *minOccurs* = "0" – To represent an optional element.
- *maxOccurs* = "unbounded" – To indicate that there is no maximum number of occurrences.

6.5 References

In some cases one might need to refer to another element in an XML Schema. This can be achieved by using *ref* attribute. We can use references to elements and attribute definitions.

XML Schema – Element Cardinality

Syntax:

```
<xs:element ref="referenced element"/>
```

Example:

```
<xs:element name="sorder" type="xs:decimal"/>  
....  
<xs:element ref="sorder"/>
```



 **Max's next Bookboon eBook**
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

6.6 New Type Definition

As it was mentioned, XSD allows user to define new types.

New type has been defined as a restriction of string (to have maximum length of 9 characters).

XML Schema – New Type Definition

Syntax:

```
<xs:simpleType name = "Mobile">  
  <xs:restriction base = "xs:string">  
    <xs:maxLength value = "9"/>  
  </xs:restriction>  
</xs:simpleType>
```

Example:

```
<xs:simpleType name = "Mobile">  
  <xs:restriction base = "xs:string">  
    <xs:maxLength value = "9"/>  
  </xs:restriction>  
</xs:simpleType>
```

6.7 Groups

Elements can be grouped together. The same is correct for attributes. Although **Group** is not a data type, it plays a container role to hold a set of elements or attributes.

XML Schema – Group

Syntax:

```
<xs:group name="group-name">  
  <xs:sequence>  
    <xs:element name="name" type="type"/>  
  </xs:sequence>  
</xs:group>
```

Example:

```
<xs:group name = "synonyms">  
  <xs:sequence>  
    <xs:element name="synonym" type="string"/>  
  </xs:sequence>  
</xs:group>
```

6.8 Constraints

XML Schema provides features based on XPath to specify uniqueness constraints and corresponding reference constraints that hold within a certain scope.

XML Schema – Constraints

Syntax:

```
<xs:unique name="constraint_name">
  <xs:selector xpath="selector_name"/>
  <xs:field xpath="field_name1"/>
  <xs:field xpath="field_name2"/>
  ...
</xs:unique>
```

Example:

```
<xs:unique name="unique_">
  <xs:selector xpath="guest"/>
  <xs:field xpath="Name/Surname"/>
  <xs:field xpath="DoB"/>
</xs:unique>
```

6.9 Key Constraints

Similar to uniqueness constraint except the value has to be non-null. Also allows the key to be referenced.

XML Schema – Constraints

Syntax:

```
<xs:key name="key_name"/>
```

Example:

```
<xs:key name="NAMEDOBUNIQUE">

  <xs:selector xpath="guest"/>

  <xs:field xpath="Name/Surname"/>

  <xs:field xpath="DoB"/>

</xs:key>
```

7 XML Query Languages

Chapter 5 presented some methods of information retrieval, using XPath. Although, as it was mentioned, XPath is a straightforward and powerful tool for retrieving information from an XML document, it is not as powerful as it is expected in query processing in data manipulation context. The concept of querying has a long history in database system regardless of their architecture. Operations such as data extraction, transformation, and integration mainly rely on query processing.

Query processing are performed using query languages. Standard Query Language (SQL) and Object Query Language (OQL) are two well-know query languages, which are used for relational and object-oriented databases, respectively. However, these query languages are not applicable, as they are, in the XML context. This is because these languages assume structured or object-based data, while XML data is not regular data.



MTHøjgaard

**BEDRE
LØSNINGER**

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang



However, as we have seen in previous chapters, XML data is similar to semistructured data. Fortunately, query languages have been developed for semistructured, which are applicable to XML documents. Different adaptation of this kind of query languages have been developed, for example:

- XML-QL (XML Query Language) was submitted to W3C in 1998 (<http://www.w3.org/TR/NOTE-xml-ql/>).
- UnQL (Unstructured Query Language), which is pronounced Uncle
- XQL

These query languages all have the notion of a path expression for navigating nested structure of XML.

7.1 XQuery

XQuery is a widely used XML query language. It has been derived from Quilt, another XML query language. Quilt has used features from different technologies such as XPath, XML-QL, SQL, OQL, Lorel (yet another query language for semistructured data), XQL, and YATL (Yet Another Transformation Language).

XQuery is a functional language in which a query is represented as an expression. XQuery supports several kinds of expression, which can be nested (supporting notion of a subquery).

7.1.1 Path Expression

Path expression is a fundamental part of XQuery. Path Expression uses XPath syntax.

In XQuery:

- The result of a path expression is an ordered list of nodes.
- The result list includes the descendant nodes as well.
- The list is ordered according to:
 - their position in the original hierarchy
 - top-down
 - left-to-right
- The result may contain duplicate values.
- Each step in the path expression represents movement through document in particular direction.
- Each step can eliminate nodes by applying one or more predicates.
- The result of each step is a list of nodes that serves as starting point for the next step.
- Path expression can begin with an expression that identifies a specific node, such as function `doc(string)`, which returns root node of named document.
- Query can also contain path expression beginning with `/` or `//`, which represents an implicit root node determined by the environment in which query is executed.

Example:

Find all usages of the first word of the XML document of Figure 6.

`doc("simpleDic.xml")/letters/word[1]//usage`

This query would be performed in the following four steps:

- It first opens “simpleDic.xml” and returns its document node.
- It uses “/letters” to select “letter” element at top.
- It locates the first “word” element that is the child of the root element.
- It finds “usage” element occurring anywhere within this “word” element.

7.1.2 FLOWR Expressions

XQuery has defined a special syntax, which is called FLWOR (pronounced “flower”). FLOW stands for FOR, LET, WHERE, ORDER BY, RETURN clauses. These are used in expressions to construct the desired queries.

To construct a FLOWR expression:

- Start with one or more FOR or LET clauses in any order
- Use a WHERE clause (optional)
- Use an ORDER BY clause (optional)
- Use a RETURN clause.

The purpose of each clause is as below:

- FOR clause
 - Binds values to one or more variables using expressions, such as path expressions.
 - Used for iteration, associating each specified variable with expression that returns list of nodes.
 - Can be interpreted as iterating over nodes returned by its respective expression.
- LET clause
 - Binds values to one or more variables using expressions, such as path expressions.
 - It does not have iteration, therefore, does a single binding for each variable.
- WHERE clause
 - Specifies one or more conditions to restrict tuples generated by FOR and LET.

- ORDER BY clause
 - Determines order of the tuple stream.
 - The determined tuple stream, then, determines order in which RETURN clause is evaluated using variable bindings in the respective tuples.
- RETURN clause
 - Evaluated once for each tuple in tuple stream and results concatenated to form result.

Example:

List “words” whose source are “Miriam-Wbseter”.

```
LET $source := “Miriam-Wbseter”
```

```
RETURN doc(“simpleDic.xml”)//word[source = $source]
```

Note that condition (predicate) seems to compare an element (source) with a value (\$source). However, in this case the ‘equality’ operator, first, returns a string value, by extracting a typed value of element, then, compares this string with \$source. In fact, ‘equality’ operator (=) is a general comparison operator that works this way.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den 9. og 10. oktober 2019.

Vi giver en is og fortæller om jobmulighederne hos os.

banedanmark



XQuery also defines value comparison operators as below:

- 'eq' – checks if two atomic nodes are equal.
- 'ne' – checks if two atomic nodes are **not** equal.
- 'lt' – checks if one atomic value is less than the other.
- 'le' – checks if one atomic value is less than or equal to the other.
- 'gt' – checks if one atomic value is greater than the other.
- 'ge' – checks if one atomic value is greater than or equal to the other.

In all the above cases, if either operand is a node, the operation uses atomization process to convert it to an atomic value.

If the FLOWR tries to compare an atomic value to an expression that returns multiple nodes, then a general comparison operator returns true if any value satisfies the condition, but the value comparison operator would fire an exception (raises an error).

7.2 XML Information Set (XML Infoset)

XML Information set (Infoset) was recommended by W3C. It provides a consistent set of definitions to allow users to refer to an XML document information. An XML document has an Infoset only if it is well-formed and satisfies namespaces. It means, there is no need for validity of the document. An Infoset includes information about some or all of the following items:

- The document
- Elements
- Attribute
- Processing Instruction
- Unexpanded Entity Reference
- Characters
- Comments
- The DTD
- Unparsed Entities
- Notations
- Namespaces

8 XML and Databases

Although XML can be considered as a data container, it is not considered a database. However, XML database also exists, which will be discussed later in this chapter. But before that let us have a look at the relationship between XML documents and traditional Relational Databases.

First of all, what would be the reason for storing an XML in a database, particularly, in a relational database? Isn't this in contradiction with the aim of relation database and its basis, relational algebra? There are different opinions about this issue. Some professionals are totally against the idea, some do not reject it in special cases, and some support it for most of the cases.

I am standing somewhere in the middle. I believe that XML has its full power in storing semi-structured data. On one hand, relational database has proven its capability for many cases, particularly, in those cases that the schema can be considered solid and changes are rare. On the other hand, there are cases that this situation is not valid. One might face systems that use kind of data that are structurally changing very often. Those who have been in database field are quite familiar with the difficulty of maintaining such databases and their related schemas. Again there are cases that a system exchanges huge amount of data with different parties, whom they have different platforms and different types of databases and data containers. In the mentioned cases, the storage of XML document in a database could be considered as a choice and I believe in most of such situations it can be a good choice.

If the decision is made to store an XML document in a relational database, then different approaches exist to do so, as below:

- To store the XML as the value of some attribute within a record (tuple).
- To fragment the XML in several fields (attributes) and tables (relations).
- To store the XML in a schema independent form.
- To store the XML in a parsed form; i.e., convert the XML to internal format, such as an Infoset or PSVI representation, and store this representation.

8.1 Storing XML in an Attribute

In this approach, the XML would be stored in a field whose data type is of Character Large Object (CLOB) type. However, some Database Management Systems (DBMSs), such as MS SQL and Oracle, provide a native XML data type, which can be used instead of using CLOB.

In this approach because raw XML is stored in serialized form, it would be efficient to insert documents into database and retrieve them in their original form. This makes the implementation and usage of full-text indexing to easier, which in turn make the contextual and relevant information retrieval more efficient.

However, some issues still remain. For example, what would be the performance when the XML document must be parsed on-the-fly? In addition, it is not possible to partially update an XML document in this case. In fact, any update means the replacement of the entire document.

8.2 Storing XML in Several Attributes and Relations

In this approach, which is also called shredded form, the XML decomposed into its constituent elements and data distributed over number of attributes in one or more relations. Storing shredded documents has the advantage of easier indexing of those elements that would be put in separate relation attributes, over the previous method, provided these elements are placed into their own attributes.

Moreover, this approach makes it possible for the developer to add some additional data that allows the hierarchical format of the XML document to be saved in the database, which in turn provides the necessary data to reconstruct the XML document as it originally was. However, this approach need some efforts for designing an appropriate database schema, which is supposed to be able to fulfill the mentioned requirements.



CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired



Click on the ad to read more

8.3 Schema-Independent Representation

Another approach is called schema-independent representation, to emphasize on the idea that the data would be stored in a relational representation which is independent of its XML schema. The design of relational representation is normally based on the DOM concept. That is, DOM is a tree-based structure, each node has only one parent.

The simple way to build a relation that is able to handle the tree structure will be as below:

parent_id	child_id	element_name	value

Table 7 – Schema-Independent Representation Relation

8.4 SQL/XML

SQL/XML is, like SQL, a declarative language as an extension to ANSI/ISO SQL 2003. It provides XML publishing functions by which users are able to create XML documents.

SQL/XML has:

- A native XML data type, which allows users to treat the XML elements as relational values in columns of tables.
- A set of operators that operate on XML data type.
- An implicit set of mappings from relational data to XML.

However, the standard does not define any rules to decompose an XML document into an SQL format.

Microsoft SQL Server uses a technology that is called SQLXML and must be differentiated from SQL/XML. The former is not a standard, rather a propriety of Microsoft that enables its database to handle XML documents.

```

CREATE TABLE simpleDic (
    letter CHAR(1), word XML,
    PRIMARY KEY letter, word );

INSERT INTO simpleDic VALUES ('E', 'ethic',

    XML('<word>

        <head> ethic </head>
        <definition> rules of behavior based on ideas about what is
            morally good and bad.
        </definition>
        <source>
            Merriam-Webster, online
        </source>
        <usage>
            Ethics is his chosen field of study.
        </usage>
    </word>') );

```

Figure 19 – Creating Table and Inserting a row using XML Type

8.4.1 SQL/XML Operators

SQL/XML offers several operators to handle XML data type as shown in Table 8.

Operator	Operation
XMLELEMENT	To generate an XML value with a single element as a child of its root item. XMLATTRIBUTES can be used as a subclause to specify the attributes of elements, if they have any.
XMLFOREST	To generate an XML value with a list of elements as children of a root item.
XMLCONCAT	To concatenate a list of XML values.
XMLPARSE	To perform a non-validating parse of a character string to produce an XML value.
XMLROOT	To create an XML value by modifying the properties of the root item of another XML value.
XMLCOMMENT	To generate an XML comment.
XMLPI	To generate an XML processing instruction.
XMLSERIALIZE	To generate a character or binary string from an XML value.
XMLAGG	An aggregate function, to generate a forest of elements from a collection of elements.

Table 8 – SQL/XML Operators

8.4.2 SQL/XML Mapping Functions

As it was mentioned, SQL/XML allows users to map data from relations (tables) into XML documents. The source of mapping might be a relation, a whole schema, or all relations in a catalog. SQL/XML provides mapping functions to perform these mappings. However, the standard does not specify any particular syntax for the mapping.

8.4.3 Mapping SQL Data Types to XML Schema

SQL/XML maps each SQL data type to its closest match in XML Schema. The ISO/IEC 9075-14:2008 Part 14 XML-Related Specifications (SQL/XML) defines a mapping from SQL data types to XML Schema data types. For more details refer to:

http://www.w3.org/2001/sw/rdb2rdf/wiki/Mapping_SQL_datatypes_to_XML_Schema_datatypes

8.5 Load XML into MySQL

XML documents can be imported into MySQL databases. This can be done using LOAD XML command. The general forms of this command is as below:

Load XML into MySQL


```
LOAD XML LOCAL INFILE 'file_name'  
INTO TABLE [dbname.]tbl_name
```


Example:

```
LOAD XML LOCAL INFILE 'mySimpleDic.xml'  
INTO TABEL langDB.enSimpleDic
```

for more information on this topic, log on into:

<http://dev.mysql.com/doc/refman/5.5/en/load-xml.html>





Max's next Bookboon eBook
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

8.6 Native XML Databases

In the previous sections, we discussed the concept of storing XML documents in relational databases. This can be done in any relational database, using one of the methods presented. However, as it was mentioned, some data bases offer facilities to store XML data. This kind of database is called “XML-enabled” in some contexts.

There is another type of database that allows user to specify an XML format for the data. Some even store the data in XML format. This kind of database is called a Native XML Database (NXD) and sometimes simply an XML database. Most of NXDs are able to respond to queries and transform data to other formats. As these databases mainly deal with XML documents, they are categorized under document-oriented databases. They are also called NoSQL (Not (only) SQL) because they do not use SQL as their main DDL (Data Definition Language) and DML (Data Manipulation Language). However, the NoSQL term is now refers to a wider area, covering databases that are able to store different formats of data, which XML is only one of them.

An NXD should define a logical data model based on XML technology. The model should, at least, provide supports for elements, attributes, PCDATA type, and document order definition. Generally, they are divided into two categories:

- Text-based : which stores XML as text, e.g. as a file in file system or as a CLOB in an RDBMS
- Model-based : which stores XML in some internal tree representation, e.g., an Infoset, PSVI, or representation, possibly with tags tokenized.

A good FAQ about using Native XML Databases can be found here:

<http://www.rpbouret.com/xml/UseCases.htm>

The following sections introduce some NXDs currently in use in the industry.

8.6.1 Oracle XML DB

Oracle started introducing Oracle XML DB when it released Oracle 9i Release2, a very powerful relational database, supporting large scale systems. This was intended to “provide high-performance database-native storage, retrieval, and management of XML data”. Oracle improved this technology alongside releasing its yet new database 11g in which it added several new capabilities to its XML DB to simplify the tasks of XML and Service Oriented Architecture (SOA) developers.

Oracle XML DB now supports:

- Multiple database-native XML storage models and XML indexing schemes
- SQL/XML standard operations
- W3C standard XQuery data model and XQuery/XPath languages
- Database-native web services
- High performance XML publishing
- XML DB repository
- XML versioning
- XML access control

Interested users can refer to the following link, for more information on the above topics.

<http://www.oracle.com/webfolder/technetwork/tutorials/obe/db/11g/r1/prod/datamgmt/xmlldb/xmlldb.htm>

8.6.2 BaseX

BaseX is a light-weight open source XML Database. As its official website says” it is a high-performance and scalable XML Database engine and XPath/XQuery 3.1 Processor, which includes full support for the W3C Update and Full Text extensions.”

Interested users can refer to the <http://basex.org/>, for more information about this Native XML Database.

8.6.3 Sedna

Sedna is a Native XML Database. It provides a full range of core database services including:

- Persistent storage
- ACID transactions
- Security
- Indices
- Hot backup.
- Flexible XML processing facilities include:
 - W3C XQuery implementation
 - Tight integration of XQuery with full-text search facilities
 - Node-level update language

Sedna is available for free in open source form under [Apache License 2.0](#).

Interested users can refer to the <http://www.sedna.org/> for more details.

8.6.4 Apache Xindice

Apache Xindice (pronounced as zeen-dee-chay) is another native XML database, designed originally to handle XML data. The Xindice developers appeal for pronouncing it correctly!

Xindice says that it is intended to change it to the data storage approach in general, they are only interested in providing a better solution to handle XML data. They say” If you don’t have any XML data, don’t want any XML data or think XML is the most over-hyped technology of the new millennium, then Xindice is not for you.” Instead they encourage those who face XML data very often to try Xindice.

Xindice is better used for small and medium size XML data. With Xindice developers are able to:

- Map XML data to some other data structures.
- To insert and retrieve data as XML.
- To use the flexibility of semi-structured nature of XML.
- To have schema independent model.

Interested users can refer to the following link, for more information on the above topics.

<http://xml.apache.org/xindice/>

An advertisement for MTHøjgaard. On the left, a woman with long blonde hair is wearing a black VR headset and looking upwards with a smile. On the right, there is a red banner with white text. The banner contains the MTHøjgaard logo, the heading 'BEDRE LØSNINGER', a paragraph of text, and a URL.

MTHøjgaard

**BEDRE
LØSNINGER**

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang



8.6.5 eXistdb

eXistdb is another NoSQL database. It is a Native XML Database, developed using Java, that is considered to be a “Document database” as well. It uses XQuery and XSLT as its query processing technology. The database has been developed based on XML technology. It is available for the three major platforms (Windows, Linux, and Mac).

Interested users can refer to the following link, for more details:

<http://exist-db.org/exist/apps/homepage/index.html>

8.7 XML and Big Data

The growth of data, particularly with the growth of the data emerging on the Internet, has pushed researchers to suggest methods and tools that make applications able to handle this kind of data. The concept of Big Data refers to sets of data that have become so extremely large that traditional applications cannot handle. Actually, XML has become one of the main sources of Big Data. Therefore, some of the major players in the field have suggested tools for XML processing in the Big Data environment.

For example, Hadoop project (<https://hadoop.apache.org/#What+Is+Apache+Hadoop%3F>) that was initiated to deal with Big Data and distributed processing, has provided a tool for parsing XML data.

Again, Google has provided the BigQuery (<https://cloud.google.com/bigquery/what-is-bigquery>) to perform queries on massive data. However, because handling complex data is not easy in BigQuery, it provides some scenarios to transform the XML contents into the formats that can be more efficiently handled. Refer to the following link for more details:

(<https://cloud.google.com/bigquery/transforming-data>)

9 XML and Software Development Platforms

Due to the rapid growth in XML usage and its popularity as well as usage diversity, almost all important software development platforms have developed different tools to support XML-based application development. Below some of these tools are presented.

9.1 XMLWriter with PHP

PHP has a class that is called **XMLWriter**, which allows programmers to create XML documents. Figure 20 shows a straightforward example of using XMLWriter in PHP. There are other ways to write the code, using some features of PHP such as iterators (using *for each*, for example) and recursive functions, that might seem more smart and efficient to some readers. However, for many cases, taking this approach could be a tedious effort with no actual reward.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.

banedanmark



```
<?php
$writer = new XMLWriter();
$writer->openURI('\\home\\user\\sampleDic.xml'); // Output file
$writer->startDocument('1.0','UTF-8');
$writer->setIndent(4);
$writer->writeRaw('<?xml-stylesheet href="sortedguest.xml" type="text/xml" ?>');
$writer->startElement('letter');
$writer->startElement('word');
$writer->startElement('head');
$writer->writeElement('head','Ethic');
$writer->endElement(); // end head
$writer->startElement('definition');
$writer->startElement('sense');
$writer->startElement('sorder');
$writer->writeElement('sorder','1');
$writer->endElement(); // end sorder
$writer->writeElement('stext','rules of behavior based on ideas about what is morally good and bad.');
```

\$writer->endElement(); // end stext
\$writer->endElement(); // end sense
\$writer->startElement('description');
\$writer->startElement('dtext');
\$writer->endElement(); // end dtext
\$writer->startElement('usage');
\$writer->startElement('utext');
\$writer->endElement(); // end utext
\$writer->startElement('source');
\$writer->endElement(); // end source
\$writer->endElement(); // end usage
\$writer->endElement(); // end description
\$writer->endElement(); // end definition
\$writer->endElement(); // end word
\$writer->endElement(); // end letter
\$writer->endDocument();
\$writer->flush();
?>

Figure 20 – PHP XMLWriter – Example

For more information refer to the following link:

<http://php.net/manual/en/book.xmlwriter.php>

9.2 Using XML in Python

Python is powerful for many kind of applications. As the main example of this book was formed around an XML based dictionary, it is appropriate if we have a look into using XML in Python, which is a well-known language for Natural Language Processing (NLP).

Python has a package for XML, which includes both DOM and SAX support. Below are two samples of codes for both cases. The codes are simple and show the part of capabilities of each API. Also you notice the differences between tree-based and event-driven approach of each API and get a better understanding and sense of how they work in a simple program.

Figure 21 shows a Python code that uses DOM to process and print the SampleDic.xml that we used for our discussion in the previous chapters. Figure 22 shows the process using SAX API for Python.

```
from xml.dom.minidom import parse
import xml.dom.minidom
DOMTree = xml.dom.minidom.parse("SampleDic.xml")
collection = DOMTree.documentElement
entries = collection.getElementsByTagName("letter")
for entry in entries:
    print "Letter:"
    if entry.hasAttribute("letter"):
        print "Letter: %s" % entry.getAttribute("letter")
    word = entry.getElementsByTagName('word')[0]
    print "Word: %s" % word.childNodes[0].data
    head = entry.getElementsByTagName('head')[0]
    print "Head: %s" % head.childNodes[0].data
    sorder = entry.getElementsByTagName('sorder')[0]
    print " %s" % sorder.childNodes[0].data
    sense = entry.getElementsByTagName('sense')[0]
    print "Sense: %s" % sense.childNodes[0].data
    utext = entry.getElementsByTagName('utext')[0]
    print "Usage: %s" % utext.childNodes[0].data
    source = entry.getElementsByTagName('source')[0]
    print "Source: %s" % source.childNodes[0].data
    description = entry.getElementsByTagName('description')[0]
    print "Description: %s" % description.childNodes[0].data
```

Figure 21 – Python and DOM – Example

```

import xml.sax
class sampleDicHandler( xml.sax.ContentHandler ):
    def __init__(self):
        self.CurrentData = ""
        self.head = ""
        self.word = ""
        self.sorder = ""
        self.stext = ""
        self.utext = ""
        self.usage = ""
    # Call when an element starts
    def startElement(self, tag, attributes):
        self.CurrentData = tag
        if tag == "letter":
            print "*****Letter*****"
    # Call when an elements ends
    def endElement(self, tag):
        if self.CurrentData == "letter":
            print "Letter:", self.letter
        elif self.CurrentData == "head":
            print "Head:", self.head
        elif self.CurrentData == "sense":
            print "Sense:", self.sense
        elif self.CurrentData == "definition":
            print "Definition:", self.definiton
        elif self.CurrentData == "usage":
            print "usage:", self.usage
        elif self.CurrentData == "description":
            print "Description:", self.description
        self.CurrentData = ""
    # Call when a character is read
    def characters(self, content):
        if self.CurrentData == "head":
            self.head = content
        elif self.CurrentData == "sorder":
            self.sorder = content
        elif self.CurrentData == "stext":
            self.stext = content
        elif self.CurrentData == "utext":
            self.utext = content
        elif self.CurrentData == "source":
            self.source = content
        elif self.CurrentData == "description":
            self.description = content
        print content
    # create an XMLReader
    parser = xml.sax.make_parser()
    # turn off namespaces
    parser.setFeature(xml.sax.handler.feature_namespaces, 0)
    # override the default ContextHandler
    handler = sampleDicHandler()
    parser.setContentHandler( handler )
    parser.parse("SampleDic.xml")

```

Figure 22 – Python and SAX – Example

9.3 XMLWriter in MS Visual Studio

MS Visual Studio provides a class that is called **XMLWriter**. It can be used in a similar way that was explained in the previous section about using XML in PHP. The class can be used in different languages that are supported by MS Visual Studio, such as C# and VB .NET. In addition, the class has variety of public methods that are supported in .NET for Windows Store apps. This feature allows users to conveniently develop mobile apps that can get benefit of XML technologies.

9.4 XML Specialized Languages

XML has found its way in many fields in science. In some fields, a specialized format of XML has been developed to serve the particularity of the field data. Below some of these specialized forms are introduced. The samples are just selected to show the variety and extensions of the specialization attempts and does not show any preference or particular importance.

9.4.1 Chemical Markup Language (CML)

“CML provides support for most chemistry, especially molecules, compounds, reactions, spectra, crystals and computational chemistry (compchem).” The interested user can refer to <http://www.xml-cml.org/> for more details.

CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired



Click on the ad to read more

9.4.2 Mathematical Markup Language (MathMLTM)

“MathML is an XML application for describing mathematical notations and capturing both its structure and content.” The interested user can refer to <http://www.w3.org/TR/REC-MathML/> for more details.

9.4.3 Keyhole Markup Language

Originally developed by Google, this XML based markup language focuses on geographical visualization and is used in GIS (Geographic Information system). The interested user can refer to <http://www.opengeospatial.org/standards/kml/> for more details.

9.4.4 Office Open XML (OOXML)

“Office Open XML (OpenXML) is a proposed open standard for word-processing documents, presentations, and spreadsheets that can be freely implemented by multiple applications on multiple platforms.”

The interested user can refer to the following addresses for more details:

<http://www.ecma-international.org/publications/standards/Ecma-376.htm>

http://www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf

9.5 XHTML – eXtensible HTML and HTML5

HTML has gone through severe enhancements and evolutions since it was introduced. HTML4 was a great step among the others. However, it did not meet all expectations of developers. XHTML was introduced as a reconstructing HTML 4 in XML 1.0 format. It was intended to be the next generation of HTML. But even this one was replaced by HTML5, which as of October 2014 has become the fifth complete and final version of HTML that is recommended by W3C.

It is, basically, a stricter and cleaner version of HTML. However, this book does not intend to discuss the HTML subject. The topic is much wider than that. Instead, I wanted to emphasis on the importance of XML again and to mention the backbone of web browsing is now base on this technology as well.

The interested users can refer to W3C documents and many other available resources on the Internet.

9.6 Security in XML

The issue of security with regard to XML is one of the main concerns of the professionals who deal with data. In this section, this issue will be briefly addressed.

W3C has provided a secure environment for manipulating data using XML. The fundamental technologies that form this environment are XML Signature, XML Encryption, and XKMS (XML Key Management Specification).

XML Signature deals with the integrity and authentication processes. XML Encryption deals with the process of encryption of a whole or a part of an XML document. XKMS is a complement for XML Signature and XML Encryption that provides protocols for the distribution and registration of public keys, which will be used in XML-based data interchange.

These technologies together address the security issue with regard to XML documents. These standards have been improved and revised during the past decade.

Furthermore, some software development environments, for example, Apache Camel and Apache Santuario, have provided specific set of tools to handle XML security. Refer to (<http://camel.apache.org/what-is-camel.html>) and (<http://santuario.apache.org/>) accordingly for more details.

The interested user can refer to <http://www.w3.org/standards/xml/security>, Petkovic and Willem (2007), and <http://camel.apache.org/xmlsecurity-dataformat.html> for more details on W3C XML security standards.

9.7 XML and NLP (Natural Language Processing)

We formed our basic example to discuss XML, around the concept of managing data for a dictionary, throughout this book. So it would be appropriate if we end the book by having a look into the usage of XML in the context of text processing and Natural Language Processing (NLP). This section provides some examples of these cases.

XML has been used by different research groups who are active in the NLP and text processing fields. For example, The Stanford Natural Language Processing Group at Stanford University (<http://nlp.stanford.edu/software/corenlp.shtml#About>) has provided a set of tools, some of which provide XML presentations of input texts, whereby the user can process different aspects of the presented text (refer to <http://nlp.stanford.edu:8080/corenlp/> for a demo).

NLTK (Natural Language Toolkit), a well-known toolkit for NLP, based on Python, has also utilized XML in different contexts such as Corpus Readers. Corpus is one of the main pillars in NLP that holds a large amount of texts, which have been collected from different sources. The NLTK Corpus Readers provide a variety of classes that can be used to access diverse set of corpora. Refer to (<http://www.nltk.org/howto/corpus.html>) for more details.

Also GATE (general architecture for text engineering), which is mainly supported by The University of Sheffield, has utilized XML in NLP and text processing (<https://gate.ac.uk/>).

To summarize, XML is considered as an appropriate environment for text and natural language processing. As it was mentioned in the Introduction chapter, for semi-structured data, XML is a proper candidate to contain and present the data. This fact and emerging XML tools that serve the related purposes, has been appreciated by researchers and developers in NLP and text processing fields.



Max's next Bookboon eBook
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com



Bibliography

Ambler, S.W. & Sadalage, P.J., 2007. *Refactoring databases*. Upper Saddle River, USA: Addison-Wesley.

Bird, S., Klein, E., and Loper, E. 2009. *Natural language processing with Python*. O'Reilly Media, Inc.

Chakrabarti, S. et al., 2009. *Data mining*. Burlington, USA: Morgan Kaufmann Publishers.

Connolly, T.M. & Begg, C., 2005. *Database systems*. 4th ed. Harlow, England: Addison-Wesley.

Connolly, T.M., Begg, C.E. & Hirsch, M. (.), 2010. *Database systems*. 5th ed. ed. Boston, USA: Addison-Wesley.

Date, C.J., 2004. *An introduction to database systems*. 8th International ed. ed. Boston, USA: Pearson Addison Wesley.

Frederick, G.R. & Lal, R., 2009. *Beginning smartphone web development*. New York, USA: Apress.

Garmany, J. & Burleson, D.K., 2004. *Oracle application server 10g administration handbook*. New York, USA: McGraw-Hill/Osborne.

Giudici, P. & Figini, S., 2009. *Applied data mining for business and industry*. 2nd ed. ed. Chichester, England: John Wiley & Sons.

Harold, E.R., 2004. *Effective XML*. Boston, USA: Addison-Wesley.

Hart, M. & Jesse, S., 2004. *Oracle database 10g*. New York, USA: McGraw-Hill/Osborne.

Haselden, K., Baker, B. (.) & Gettman, K.), 2006. *Microsoft SQL server 2005 integration services*. Indianapolis, USA: Sams Publishing.

Hoffer, J.A., Prescott, M.B. & McFadden, F.R., 2007. *Modern database management*. 8th ed. ed. Upper Saddle River, USA: Pearson Prentice Hall.

Hoffer, J.A., Prescott, M.B. & Topi, H., 2009. *Modern database management*. Upper Saddle River, USA: 9th International ed.

- Katz, H., and Chamberlin, D.D., eds., 2004. *XQuery from the experts: a guide to the W3C XML query language*. Addison-Wesley Professional.
- Kifer, M., Bernstein, A. & Lewis, P.M., 2006. *Database systems*. 2nd International ed. Boston, USA: Pearson Addison-Wesley.
- Kroenke, D.M., 2006. *Database processing*. 10th ed. ed. Upper Saddle River, USA: Pearson Prentice Hall.
- Kroenke, D.M., 2009. *Database Processing: Fundamentals, Design and Implementation*. 10 ed. NY: Prentice Hall.
- Kroenke, D.M. & Auer, D.J., 2010. *Database Concepts*. 4 ed. s.l.: Pearson Education Inc.
- Lewis, J. & Loftus, W., 2007. *Java software solutions*. 5th ed. ed. Boston, USA: Pearson Addison-Wesley.
- Loney, K., 2004. *Oracle database 10g: the complete reference*. New York, USA: McGraw-Hill/Osborne.
- Loney, K. & Bryla, B., 2005. *Oracle database 10g DBA handbook*. New York, USA: McGraw-Hill/Osborne.
- Mannino, M.V., 2007. *Database design, application development, and administration*. 3rd ed. ed. Boston, USA: McGraw-Hill Irwin.
- Mannino, M.V., 2009. *Datasbase design, application development, and administration*. USA: Michael V Mannino.
- Petkovic, M & Willem J. 2007. *Security, Privacy, and Trust in Modern Data Management*. Springer.
- Pratt, P.J., Adamski, J.J. & DiMassa, C. (., 2008. *Concepts of database management*. 6th ed. Boston, USA: Cengage Learning .
- Price, J., 2004. *Oracle database 10g SQL*. New York, USA: McGraw-Hill/Osborne.
- Rob, P. & Coronel, C., 2002. *Database systems*. 5th ed. Boston, USA: Thomson Course Technology.
- Rob, P. & Coronel, C., 2009. *Database systems*. 8th ed. Australia : Thomson Course Technology.
- Rob, P. & Cronell, C., 2007. *Database Systems: Design, implementation and management*. 7 ed. MA: Course Technology.
- Scott, M.L., 2009. *Programming Language Pragmatics*. 3rd ed. Morgan Kaufmann

Silberschatz, Korth & Sudarshan, 2006. *Database System Concepts*. 5 ed. s.l.:McGraw-Hill Higher Education.

Ullman, J. d. & Widom, J., 2002. *A First Course In Database Systems*. 2nd ed. Upper Saddle River, USA: Prentice Hall.

Ullman, L. & Gulick, R. (., 2008. *PHP 6 and MySQL 5 for dynamic web sites*. Berkeley, USA: Peachpit Press.

Venugopal, S., 2007. *Data structures outside in*. Upper Saddle River, USA: Pearson Prentice Hall.

Welling, L., Thomson, L. & AB, M., 2004. *MySQL tutorial*. Indianapolis, USA: MySQL Press.

Welling, L., Thomson, L. & Clapp, C. (., 2005. *PHP and MySQL web development*. 3rd ed. Indianapolis, USA: Sams Publishing.

Yee, R. & Moodle, M. , 2008. *Pro web 2.0 mashups*. Berkeley, USA: Apress.

Zakas, N.C., McPeak, J. & Fawcett, J., 2007. *Professional Ajax*. 2nd ed. Indianapolis, USA: Wiley Publishing Incorporated.

An advertisement for MTHøjgaard. On the left, a woman with long blonde hair is wearing a black VR headset and looking upwards with a joyful expression. The background is a solid grey. On the right, there is a large red triangular graphic. Inside this red area, the MTHøjgaard logo is at the top, followed by the text 'BEDRE LØSNINGER' in bold white capital letters. Below this, there is a paragraph of white text describing the company's focus on developing new methods and technologies for the construction and infrastructure sectors. At the bottom of the red area, the website 'mth.dk/vorestilgang' is listed in white.

MTHøjgaard

**BEDRE
LØSNINGER**

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang

