Research Article

XML Schema Validation Using Java API for XML Processing

Shene Jalil Jamal [1,a,*], Chnoor M. Rahman [2,b], Mzhda S. Abdulkarim [2,c]

[1] Computer Department, College of Science, University of Sulaimani, Sulaymaniyah, Kurdistan Region, Iraq
[2] Department of Applied Computer, College of Medical and Applied Sciences, Charmo University, Chamchamal, Sulaymaniyah, Kurdistan Region, Iraq

[a] shene.jamal@univsul.edu.iq, [b] chnoor.rahman@charmouniversity.org, [c] mzhda.sabir@charmouniversity.org

Access this article online

Received on: 30 November 2021	**Accepted on:** 20 June 2022	**Published on:** 30 June 2022
DOI: 10.25079/ukhjse.v6n1y2022.pp33-41	**E-ISSN:** 2520-7792	

Abstract
Extensible Markup Language (XML) is a markup language that is developed to organize the structure of information in a text file. The data in XML formatted documents are represented by specifying a number of tags and determining the structural relationship between those tags. It has a simple structure and can be handled by any text editor. Therefore, XML formatted data is being commonly used to transfer and share data between different applications and organizations without having to convert the format of the data (Yang, 2019).

In the XML world, "well-formed" and "valid" are the two most frequently used terms. A well-formed XML document is free from errors that can cause the document to not parse, such as: spelling, punctuation, grammar, and syntax errors. While in addition to having a well-formed markup, a valid XML must conform to a document type definition, this means the document must be semantically correct and matches a described standard of schemas and relationships (Appel, 2020).There are two standards of document type definition that can be used to validate an XML document, one is DTD or Document Type Definition which is used to identify the legal structure and names the legal elements of an XML document (Dykes and Tittel, 2011), and the other is XSD or XML Schema Definition. XSD is a diagrammatic representation that defines the valid structure of an XML document, it enables specifying the building blocks of an XML data set such as elements and attributes and their data types, number of child elements, fixed and default values of the elements and attributes that can appear in the documents (XML Schema Tutorial, 2020). In some applications the process of validating XML documents is combined with parsing the document. However, in some other cases the process of parsing and validating the XML documents need to be separated. This study focuses on constructing a separate XML document validator and validating XML documents against the defined XSD rules. A Java program is used to perform this experiment. Furthermore, the critical differences between XSD and DTD are also mentioned.

Keywords: XML Schema, XSD, JAXP, DTD, Java.

1. Introduction
In general, a schema in XML represents the specifications of objects and identifies the relationships between the objects. Schema definition languages are the recommendations of the World Wide Web Consortium (W3C) which itself is represented by XML. The language provides facilities to describe the structure of an XML document. It is described as rules that apply to a group of XML documents (Lawton, 2015). The main purpose and most popular use of schema is validation. Different kinds of validation can be performed with different kinds of schema. This means the validation

requirements might be different in different situations. There are many circumstances which requires validating XML documents, for example when testing the output of an application to make sure that the data meets the specified requirements, and when receiving an XML document from an external source (in case of Web Services) the data must be validated before inserting it into a legacy system (Gandhi, 2014). At several levels of processing data, validation could occur. For instance, structural validation makes sure that the structure of XML elements and attributes in an XML document satisfy the identified requirements, however it does not illustrate the textual content of the document. Data validation is another kind of validation process, which ensures that the content of an XML document follows the rules which specify the type of information that should be presented (Gandhi, 2014). This paper focuses on constructing a tool to validate XML documents against XSD in terms of structure and data content validations. The validation process is performed using Java programming language, as java provides useful and easy to use Application Programming Interface (API) for the purpose of validating XML against XSD. Furthermore, the data model of XSD and the differences between XSD and DTD are discussed.

The research starts by studying XML schema abstract data model as an approach to determine how the schema of an XML document and the data elements should be organized. Next, two available and common methods to describe the structure and content of XML documents are; Document Type Definition (DTD) and XML Schema are discussed and compared. Based on the advantages of XML Schema over DTD, XML Schema has been used for the validation process. In section five 'validation technique' the proposed validation method is explained. Java API for XML Processing (JAXP) is utelized to develop the validator, and all the necessary steps and phases of the validation process is presented in this section. Finally, in the results and discussion section the research mentiones the requirements of an application that should be considered while choosing a validation technique.

2. Materials and Methodology

In this paper, the technique used to study XML validation is implemented in Java programming language. Java supports most of the XML related technologies. The proposed validation technique supports those systems and applications that require the process of validation and parsing XML documents to be separated. Therefore, the chosen Application Programming Interface (API) is JAXP Validation API. This API allows the applications to validate an XML document against an XML Schema Definition (XSD) without having to parse the XML document. The most important procedures of making this validation program are presented with details in section "The Validation Technique".

3. XML Schema Abstract Data Mode

The information set that is expressed in XML schema components identifies the abstract data model of the XML schema (Geroimenko, 2012). According to W3C one fundamental abstract data model can describe all XML Schemas. That model specifies the structure and the data content of the schemas. However, The XML Schema abstract data model is only conceptual, and it does not impose any specific style or structure on the XML documents or schemas. It only provides a large collection of components and dictates how the components should be linked (McKinnon and McKinnon, 2003). As shown in Table 1, W3C describes several kinds of schema components, which can be classified in three groups, which are Primary components, Secondary components and Helper components (Henry and Gao, 2012). The entire Primary component and some of the Secondary components are discussed in this section.

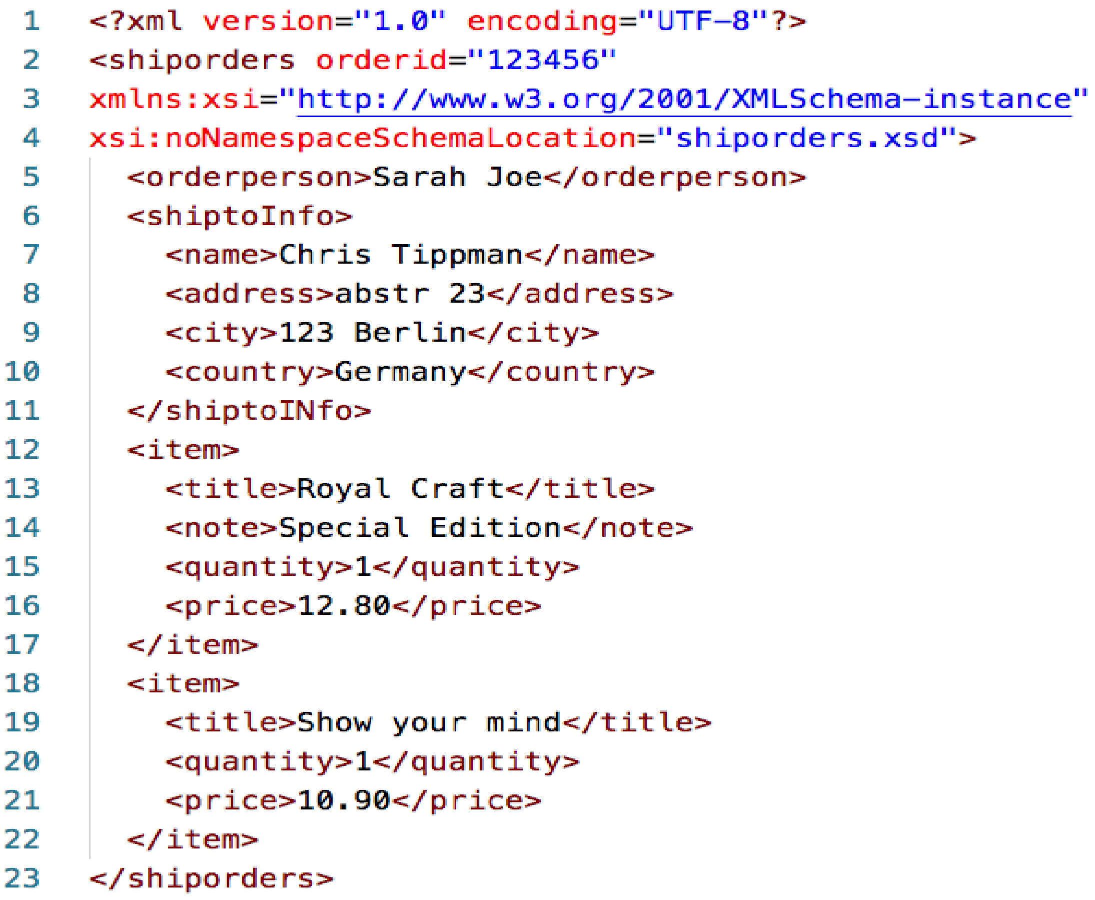
Table 1. XML Schema Component Groups.

Component Group	Schema Components
Primary	Simple type definitions
	Complex type definitions
	Attribute declarations
	Element declarations
Secondary	Attribute group definitions
	Identity-constraint definitions
	Model group definitions
	Notation declarations
Helper	Annotations
	Model groups

	Particles
	Wildcards
	Attribute Uses

The root element of an XML Schema is schema, which is usually defined in the schema namespace. Within a schema construct, elements can be declared using element construct <xsd:element> , attributes are also specified using attribute construct <xsd:attribute> (elements and attributes can be declared using xs or xsd prefix. In practice there is no difference between xs and xsd.). Two type constructs are described in XML Schema language, a simple type and a complex type. Complex type is used in almost all meaningful document structures. An element that contains an attribute, a child element or both make a complex type. Complex types are defined in the element declaration using a combination of <xsd:element> and <xsd:complex-Type>. In contrast, elements that only contain numbers or character strings with child elements are said to be simple types. To declare a simple type a combination of <xsd:element> and <xsd:simpleType> is used (Henry and Gao, 2012) and (Vohra, 2007).

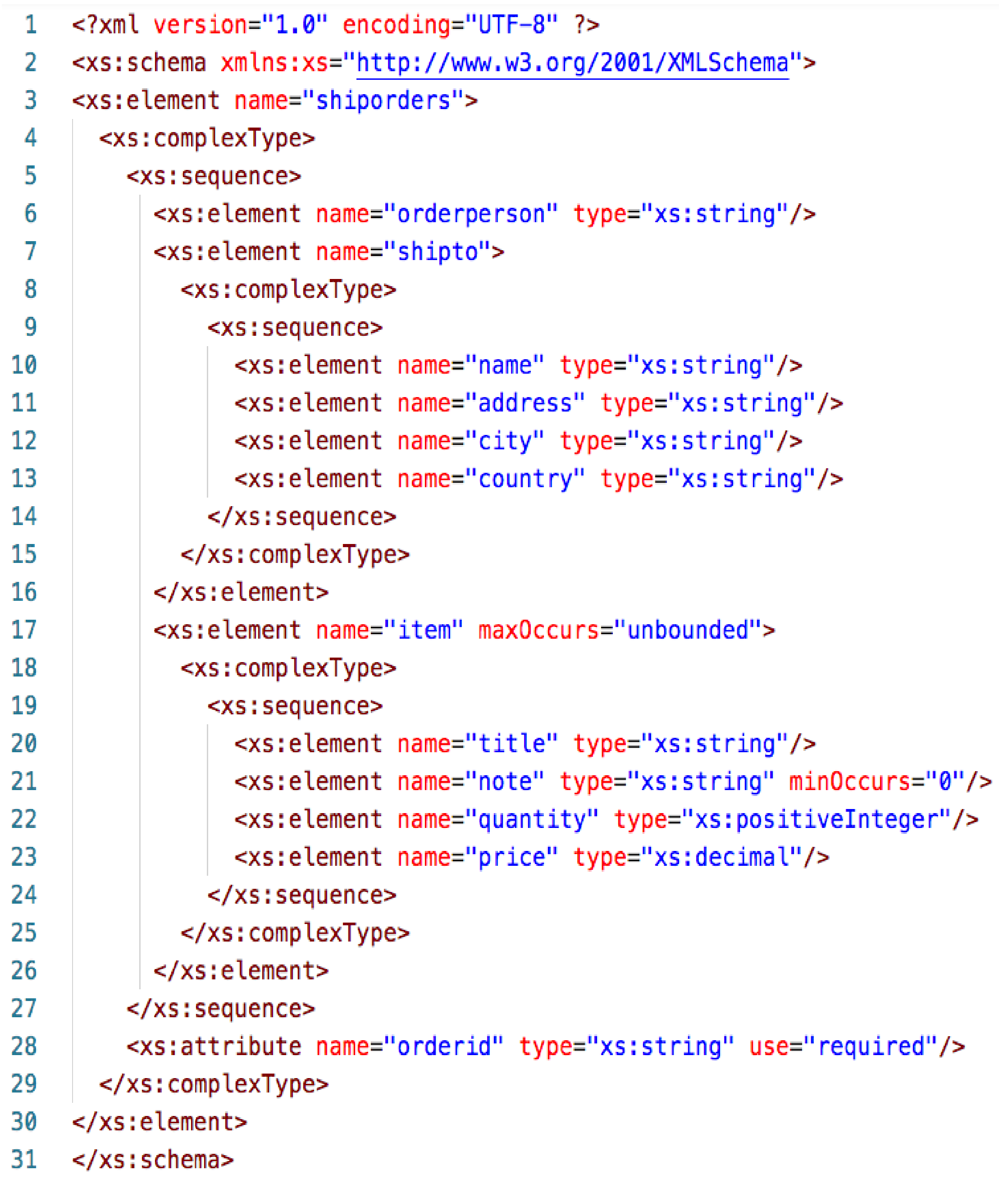
Generally, Schema models are described in terms of constraints to determine what a specific document or language can contain. XML Schemas describe two kinds of constraints: The first is Content Model Constraints, which define the elements that can occur in an XML document. It also specifies the grammar of the language in the documents. The second constraint is Data Type Constraints. These constraints specify the types of data that the schema recognizes as valid (Henry and Gao, 2012), and (Owen and Boyer, 2011). An example of an XML document ("shiporders.xml") and its XML Schema ("shiporders.xsd") are shown in Figure 1 and 2.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<shiporders orderid="123456"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporders.xsd">
  <orderperson>Sarah Joe</orderperson>
  <shiptoInfo>
    <name>Chris Tippman</name>
    <address>abstr 23</address>
    <city>123 Berlin</city>
    <country>Germany</country>
  </shiptoINfo>
  <item>
    <title>Royal Craft</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>12.80</price>
  </item>
  <item>
    <title>Show your mind</title>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
</shiporders>
```

Figure 1. An XML Document "shiporders.xml" (W3Schools, 2021).

In the above XML Document "shiporders" is the root element, the attribute "orderid" is identified in this element. The root element "shiporders" specifies three child elements "orderperson", "shipto" and "item", these elements also contain other different child elements. The sentence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" indicates that an XML parser should validate the XML document based on a schema, and the line xsi:noNamespaceSchemaLocation="shiporders.xsd" informs the parser about the schema file (in this example the schema file is “shiorders.xsd”) and the location of the file (in this example it is in the same directory as “shiporders.xml”). Figure 2 shows the schema file, which starts with the standard XML declaration and is also organized in XML format.



```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="shiporders">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Figure 2. An XML Schema “shiporders.xsd” (W3Schools, 2021).

In the above XML Schema, the root element <xs:schema> defines the attribute xmlns:xs which is associated with the URI "http://www.w3.org/2001/XMLSchema" to specify the namespace.

Next, the root element “shiporders”, its attribute and child elements are defined. This element makes a complex type because it contains attribute and child elements. To display the three child elements (“orderperson”, “shipto”, “item”) of “shiporders” in an ordered sequence, the elements are enclosed between <xs:sequence> elements. The “orderperson” is considered as simple type as it does not contain any attribute or child element. However, “shipto” and “item” elements are of complex types (Owen and Boyer, 2011). The "maxOccurs" and "minOccurs" attributes determine minimum and maximum possible appearance of an element. The default value of these attributes is 1, and if the occurrence of the element is optional then "minOccurs" is set to have the value 0. Here In the declaration of "item" element the value of "maxOccurs" is "unbound" which means the item element can occur many times. "shiporders" has a required attribute "orderid", the attribute is declared in the end of the schema.

4. XML Schema versus DTD

Document Type Definition (DTD) and XML Schema are two approaches to organize the structure and define the content of an XML document. Although DTD was founded first, there are some significant differences between the two methods, and XML Schema has considerable advantages over DTD. The main advantage of XML Schema is that they are strongly typed and they have the ability to define data types of elements and specify their values, length and define other complex structures. This facility can confirm that the data stored in an XML document is valid. While DTD lacks this ability, it is weakly typed and cannot validate data contents to their data types (Joan, 2011), and (Tidwell, 2008). Another difference which makes XML Schema to be more powerful is the fact that XML Schema is written in XML format. Therefore, XML Schemas can be parsed and manipulated like any XML document. In contrast DTD is defined in SGML (Standard Generalized Markup Language). Accordingly, defining the structure and content of an XML document in DTD requires the need to learn a new language (Joan, 2011). Supporting namespace is another characteristic of XML Schema, it can use a set of namespaces to define and organize the Schema. While on the other hand DTD is not aware of namespaces, instead it defines the building blocks of XML documents using its own set of keywords (Ali, 2012). Considering the advantages of XML Schema over DTD, this study focuses on validating XML documents against XML Schema rather than DTD.

5. The Validation Technique

The process of validating XML documents is often necessary to make sure that any system or application which uses the document receives the correct and expected form of data. The validation tool which is developed for this purpose must confirm that the XML documents adhere to the structures and rules specified by a Schema. This section illustrates the steps of developing an XML Schema validator to validate XML documents against XSD using XML related technologies in Java. The validator could then be used when the process of validation needed to be decoupled from parsing. The Application Programming Interface (API) used to develop the validator is Java API for XML Processing (JAXP). JAXP allows the applications to parse, validate and query XML documents. It can be divided into two groups: The first group contains JAXP SAX and DOM parser APIs. SAX (Simple API for XML) is an event based API, it generates a series of events while parsing documents which are handled by callback methods. While DOM (Document Object Model) is an object based API, it represents the XML document in a tree structure so that the application can use the tree nodes for manipulating and querying the data in the document (Li, 2009). These APIs are suitable to be used in applications, which require parsing and validating the XML documents to be combined. In contrast, the second group contains JAXP Validation API. This API is suitable to be used in applications, it require parsing and validating XML documents to be decoupled (Vohra, 2007). This second group API is used to examine the validation process in this research. To validate XML Documents with JAXP Validation API three steps must be taken. Figure 3 is a flowchart diagram that represents the general steps of the validation process.

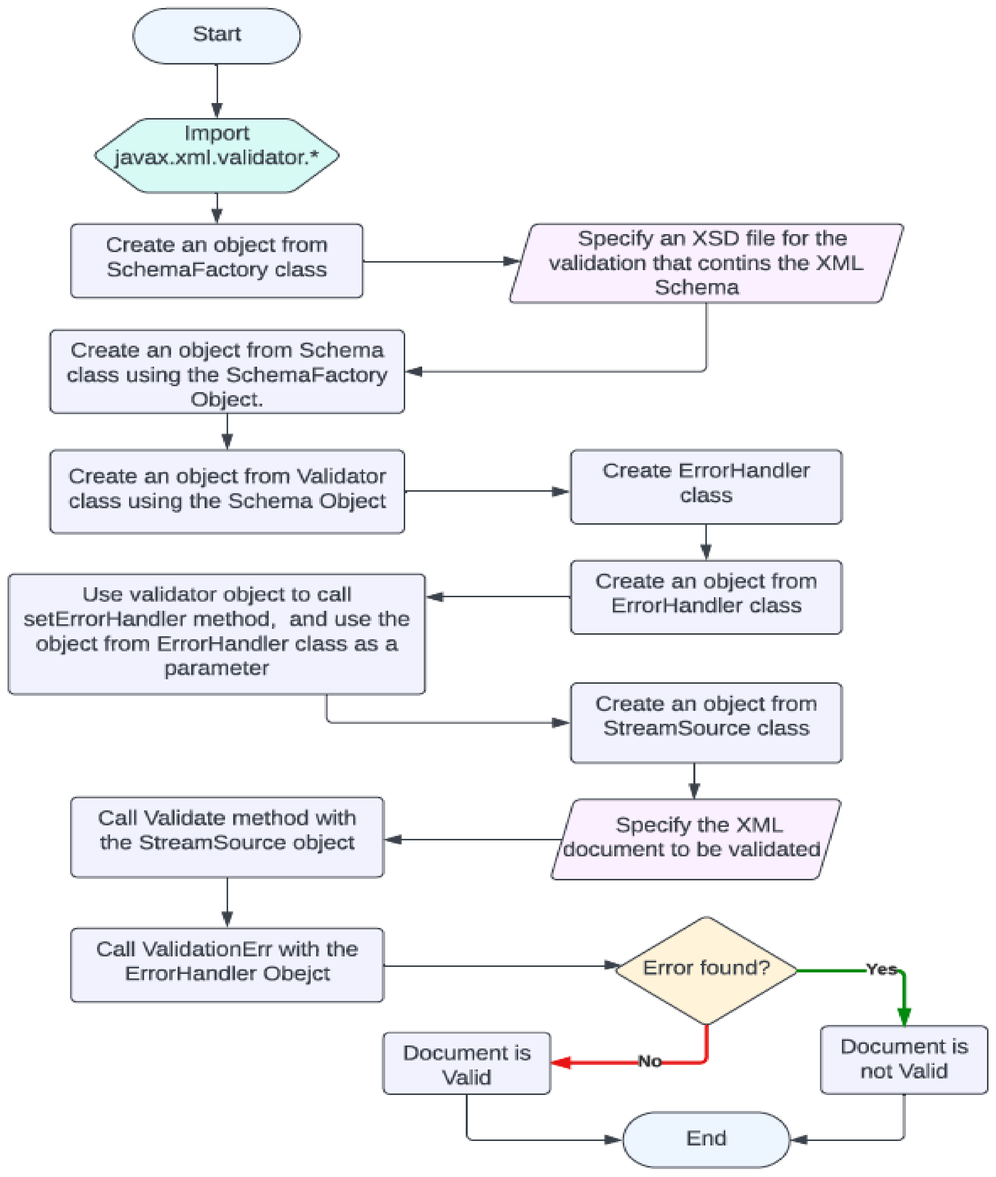

Start
Import javax.xml.validator.*
Create an object from SchemaFactory class
Specify an XSD file for the validation that contins the XML Schema
Create an object from Schema class using the SchemaFactory Object.
Create an object from Validator class using the Schema Object
Create ErrorHandler class
Create an object from ErrorHandler class
Use validator object to call setErrorHandler method, and use the object from ErrorHandler class as a parameter
Create an object from StreamSource class
Specify the XML document to be validated
Call Validate method with the StreamSource object
Call ValidationErr with the ErrorHandler Obejct
Error found?
Yes
No
Document is Valid
Document is not Valid
End

Figure 3. Flowchart of the Validation Process.

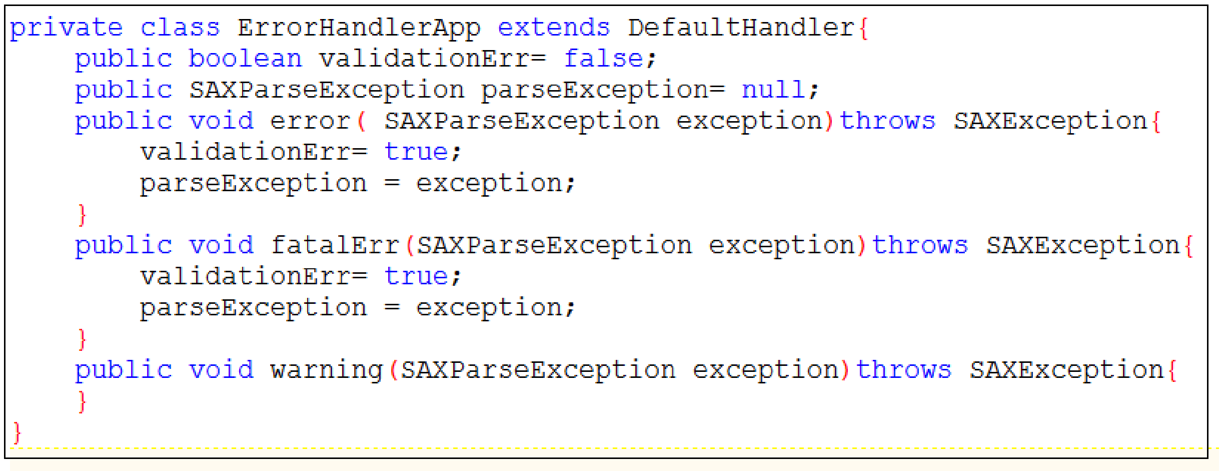
First: Instantiating three necessary objects.
The three required objects to be obtained are objects from SchemaFactory , Schema, and Validator classes. To be able to create these objects the javax.xml.validation.* package must be imported first. A Schema object representation is necessary to be able to validate with XML Schema based definition (Vohra, 2007). The Schema object is created from SchemaFactory class as shown below:

```
SchemaFactory factory= SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
Schema schema= factory.newSchema(newFile("XMLSchemaFile.XSD"));
```

The parameter to the newSchema() method is the name of an XML Schema file which the validation depends on to check if an XML document follows the specified rules and structures that are defined in the file. The validator object is then created from the schema object as shown below.

```
Validator validator = schema.newValidator();
```

Second: Reporting Validation Error.
The validator should be able to handle validation errors. To accomplish this task an ErrorHandler class must be defined. This class extends DefaultHandler as shown in Figure 4. An object from this class is set as an argument to the setErrorHandler() method, which is invoked by the "validator" object created in the first step.



```
private class ErrorHandlerApp extends DefaultHandler{
    public boolean validationErr= false;
    public SAXParseException parseException= null;
    public void error( SAXParseException exception)throws SAXException{
        validationErr= true;
        parseException = exception;
    }
    public void fatalErr(SAXParseException exception)throws SAXException{
        validationErr= true;
        parseException = exception;
    }
    public void warning(SAXParseException exception)throws SAXException{
    }
}
```

Figure 4. The ErrorHandlerApp Class.

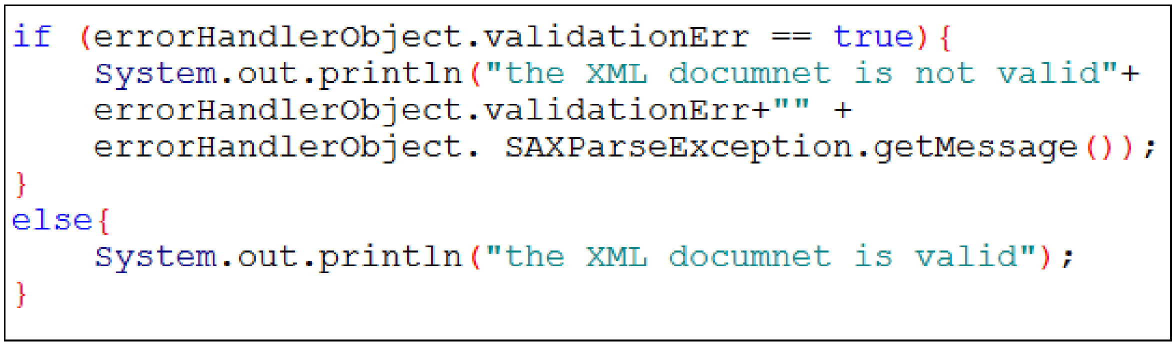
Creating an object from this class and using it with the setErrorHandler() method is shown below:

```
ErrorHandlerApp errorHandlerObject= new ErrorHandlerApp();
validator.setErrorHandler(errorHandlerObject);
```

Third: Validating the XML document.
The last step to complete the validation process is validating the XML document. To process the validation, the XML document does not need to be parsed. Alternatively an StreamSource object is created from the XML document, and the validator objects is used to call the validate() method. This method takes the StreamSource object as an argument.

```
StreamSource streamsrc = new StreamSource (XMLDocument.xml);
Validator.validate(streamsrc);
```

Lastly, the following lines of code were added to check for errors and show the result of the whole validation process. Figure 5 shows the result of the process, if the XML document is not valid then error messages will be shown, otherwise the program indicates that the document is valid.



```
if (errorHandlerObject.validationErr == true){
    System.out.println("the XML documnet is not valid"+
    errorHandlerObject.validationErr+"" +
    errorHandlerObject. SAXParseException.getMessage());
}
else{
    System.out.println("the XML documnet is valid");
}
```

Figure 5. The Result of the Validation Process.

6. Result and Discussion

Selecting an approach to validate an XML document depends on satisfying the additional functionality of the validation application. For instance, if parsing an XML document requires being associated with validating the document with a schema, then SAX parser is recommended. However, if the entire tree structure of an XML document needs to be accessed and modified repeatedly, then DOM parser is recommended (Nagrare, 2020b). Furthermore, the parsing process can be performed along with validation if the chosen XML parser can implement validation too. Nevertheless, sometimes the parsing and validation process is required to be separated. The XML validation method examined in this research satisfies this requirement. Separating validation from parsing using JAXP Validation API could be a practical solution for the applications that need to validate an XML document that is not supported by the accessible parser. In addition, with JAXP Validation API, an object is instantiated to represent a schema. This object can be used to validate multiple XML documents. According to (Vohra, 2007) using a single object to validate multiple XML documents is an efficient process. Therefore, JAXP Validation API could be an appropriate method to utilize if for any reason the parsing process of an XML document requires being decoupled from the validation process.

7. Conclusion

With increasing the number of XML related technologies, XML is becoming more popular and is used in a large number of various applications and systems. Therefore, it is vitally important to organize the XML formatted documents in well formed and validated structure and content. This way the systems and applications will be able to predict which kind of data format is received as input, and which structure of data will be produced as output. DTD and XSD are two available standards to define structure and content of XML documents and validate the XML documents against described structures. This study mainly focused on using XSD to accomplish the process of validation, as XSD has more advantages than DTD. For example, XSD has the same structure as XML documents and it is parsed and processed the same way as XML. Furthermore, XSD supports multiple namespaces, and it also has the ability to specify the elements data types and their values to make the validation a reliable process. There are various programming languages that provide techniques to implement the XML validation process. The validation system proposed in this study was implemented in Java Programming language. JAXP is the API which is used for this purpose. This API could be divided into two classes. The first class includes JAXP SAX and DOM, which perform XML validation as a part of parsing. While the second class includes JAXP Validation API, which could be used in those applications that need the process of validation and parsing to be decoupled. The experiment discussed in this paper uses the API of the second class, as the technique is proposed to support systems that separate validation from parsing. A number of scenarios might involve separating these two processes, for example: if the Schema was available from an external source, if the schema language was not supported by the available parser or if the application needed to validate multiple XML documents against the same schema definition. The JAXP Validation API was used to construct a separate validation program to support validating XML documents against external Schema whenever it is required.

References

Ali, U. (2012). Difference between dtd and xsd. Slideshare. Retrieved 01 November 2021 from https://www.slideshare.net/umarali1981/difference-between-dtd-and-xsd.

Appel, R. (2020). Create well-formed XML and schema efficiently | The .NET Tools Blog. JetBrains Blog. Retrieved 01 October 2021 from https://blog.jetbrains.com/dotnet/2020/09/29/create-well-formed-xml-and-schema-efficiently-rider.

Dykes, L. and Tittel, E. (2011). *XML For Dummies* (4th ed.). Wiley.

Gandhi, M. (2014). *W3c XML Schema 1.1 for Beginners* (1st ed.). Reed Business Education.

Geroimenko, V. (2012). *Dictionary of XML Technologies and the Semantic Web* (illustrated ed.). Springer Publishing.

Henry, S. and Gao, S. (2012). W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures -- Review Version. W3C Recommendations. Retrieved 12 October 2021 from https://www.w3.org/TR/xmlschema11-1/structures.diff-1.0.html.

Joan, B. (2011). XML Schema and DTD. Difference Between. Retrieved 02 November 2021 from http://www.differencebetween.net/technology/difference-between-xml-schema-and-dtd.

Lawton, G. (2015). XSD (XML Schema Definition). WhatIs.Com. Retrieved 08 October 2021 from https://whatis.techtarget.com/definition/XSD-XML-Schema-Definition.

Li, C. (2009). XML Parsing, SAX/DOM. University of Texas at Arlington. Retrieved 15 June 2021 from https://ranger.uta.edu/~cli/pubs/2009/XMLParsing_ChengkaiLi.pdf.

McKinnon, L. and McKinnon, A. (2003). *XML in 60 Minutes a Day* (1st ed.). Wiley.

Nagrare, S. (2020a). Difference Between DOM and SAX ? DOM vs Sax Easy Explain. Easy Difference Between. Retrieved 20 November 2021 from https://easydifferencebetween.com/difference-between-dom-and-sax/.

Nagrare, S. (2020b). Difference Between DOM and SAX ? DOM vs Sax Easy Explain. Easy Difference Between. Retrieved 20 November 2021 from https://easydifferencebetween.com/difference-between-dom-and-sax/.

Owen, D. and Boyer, W. (Eds.). (2011). *Exam 70–432: Microsoft SQL Server 2008 Implementation and Maintenance with Lab Manual Set* (1st ed.). Wiley.

Tidwell, D. (2008). *XSLT: Mastering XML Transformations* (2nd ed.). O'Reilly Media.

Vohra, A. and Vohra, D. (2007a). *Introducing Schema Validation. In Pro XML Development with Java Technology* (1st ed. pp. 65–66)). Apress.

Vohra, A. and Vohra, D. (2007b). *Introducing XML and Java. In Pro XML Development with Java Technology* (1st ed., pp. 1–31). Apress.

W3Schools. (2021). An XML Document [Illustration]. W3Schools. Retrieved 20 November 2021 from http://www.w3schools.com/xml/schema_example.asp.

XML Schema Tutorial. (2020). W3schools. Retrieved 02 October 2021 from https://www.w3schools.com:443/Xml/schema_intro.asp.

Yang, H. (2019). XML Tutorials - Herong's Tutorial Examples. [online] Google Books. HerongYang.com. Retrieved 15 June 2021 from https://books.google.iq/books?id=zKoDEAAAQBAJ&pg=PA13&dq=introduction+to+xml&hl=en&sa=X&ved=2ahUKEwjZs43ega_4AhXXkmoFHY31DaY4HhDoAXoECAoQAg#v=onepage&q=introduction%20to%20xml&f=false.