NISTIR 7829

XML Validation Website: A User's Guide

Julien Cuvillier
KC Morris

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 7829

XML Validation Website: A User's Guide

Julien Cuvillier
KC Morris
Systems Integration Division
Engineering Laboratory

November 2011

U.S. Department of Commerce
John E. Bryson, Secretary

National Institute of Standards and Technology
Patrick D. Gallagher, Under Secretary of Commerce for Standards and Technology and Director

Abstract

This report describes installation and use of the XML (Extensible Markup Language) Validation Website. The website is one of several tools produced by NIST's XML Testbed project within the Manufacturing Systems Integration Division at NIST. These tools aid in producing high quality XML schemas using a standards-based approach to manufacturing systems integration. The website provides a central site for testing XML schemas outside of a user's own operating environment with a number of externally available XML tools, thereby validating that the schemas will work in other environments before distribution. The XML Validation website which hosts a number of XML parsing tools that can be used to remotely validate XML schemas against the W3C (World Wide Web Consortium) XML Schema standard [1], and XML data against their corresponding XML schemas.
The website is available at `http://www.nist.gov/el/msid/xmltestbed.cfm`.

Contents

1 How to use the XML Validation Website

This section provides instructions for the end user of the XML validation website.

1.1 Schema Validation

Schema validation allows the user to upload their own schema file, or a zip file containing multiple schemas, and validate it against the W3C standard specification for XML schemas [1].

1.1.1 Upload a Schema

The schema can be uploaded as a schema file (".xsd" or ".XSD") or as a zip file containing multiple schemas. (If a zip file is uploaded, the target schema file name, including the path starting from the root directory, must be specified). The target schema must have the extension ".xsd" or ".XSD".

1.1.2 Select a parser

The user can select one or more parsers among the list below.

- Xerces 2.9.1 [2]
- JING 20081028 [3]
- MSV 20081113 [4]

1.2 Instance Validation

Instance validation allows the user to upload their own XML instance and validate its content with either their own uploaded schema, or from a selection of five currently available public schemas. Other schemas may be added.

- OASIS UBL v1.0 & v2.0 [5]
- OAGIS v9.4.1 [6]
- AIAG [7]
- StratML [8]
- MSX-DMS v1.3 [9]

Note that instance validation is one way to validate that an XML schema meets the requirements captured in an XML schema. If errors are encountered during instance validation, it may reflect problems in the XML schema.

1.2.1 Upload an XML file

The XML file uploaded must have the extension ".xml" or ".XML" and its name cannot contain any white spaces.

1.2.2 Select a Schema

There are two ways to select a schema:

- The schema can be uploaded as a schema file (".xsd" or ".XSD") or as a zip file containing multiple schemas. If you uploaded a zip file, you must specify the target schema file name including the path, starting from the root directory. The target schema must have the extension ".xsd" or ".XSD" and its name cannot contain any white spaces.
- Alternatively, a schema can be selected from the list of schemas installed on the system (see section 1.2). For the latter, two options exist:
  - Select a set of schemas below and set the SchemaLocation attribute at the root of the XML instance. The SchemaLocation value must correspond with the path of the file in the expanded listing below.
  - Expand the appropriate listing below and select the schema to use.

1.2.3 Select a parser

The user selects at least one parser among the following list:

- Xerces 2.9.1 [2]
- JING 20081028 [3]
- MSV 20081113 [4]
- JAXP 1.4.2 [10]
- LibXml2 2.7.6 [11]

1.3 Accessing the results

The results for both schema and instance validation can be viewed in two ways:

- As a web page, all the results are displayed on the same page, and you have the following information:
  - XML file
  - Schema file

  - Parser name
  - Validation output

- Alternatively, by entering your email address the results will be emailed to you in the form of an xml file. Figure 1 shows an example of the XML files. The schema for the XML file is provided in Appendix A. Appendix B contains an xslt [12] file for rendering a results file.

```
<?xml version="1.0"?>
<ValidationResults xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    noNamespaceSchemaLocation="ValidationResults.xsd">
   <Results>
      <Parser>
         <Name>Xerces</Name>
         <Version>2.9.1</Version>
         <Output>The output of the validation</Output>
      </Parser>
      <Files>
         <SchemaFile>schema.xsd</SchemaFile>
      </Files>
   </Results>
</ValidationResults>
```

Figure 1: Results of a schema validation in XML format

2 Adding new features

This section provides instructions for an administrator to add new schemas or parsers to the website.

2.1 The configuration file

The configuration file is an XML file that contains information about the schema and the parsers used. This file is used to generate the web pages. Figure 2 is an example of a configuration file.

2.2 Add new default schemas

In order to add a new schema directory, add a new entry in the config file with the following information: (see figure 2 for an example)

- **name**
- **version**
- **path**
- **comment** (optional)
- **url** (optional)

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <parsers>
        <parser name="Parser1" version="1.0" ID="1" instance_validation="true"
            schema_validation="true" needSchemaLocation="true">
            <depLocPath>path</depLocPath>
            <class>util.parser.Parser1</class>
            <comment>comment</comment>
            <url>http://www.parser1.com/</url>
        </parser>
        <parser name="Parser2" version="2.0.1" ID="1" instance_validation="false"
            schema_validation="true" needSchemaLocation="false">
            <depLocPath>path</depLocPath>
            <class>util.parser.Parser2</class>
        </parser>
    </parsers>
    <schemas>
        <schema name="Schemas" version="NA" ID="1">
            <path>path</path>
            <comment>comment</comment>
            <url>http://www.schemas-url.org</url>
        </schema>
    </schemas>
</configuration>
```

Figure 2: Example of a configuration file with the part related to the schemas highlighted

2.3 Add a new parser

To add a new parser, two modifications are needed: creating a new class and adding the parsers to the configuration file.

2.3.1 Creating a new class

Create a class that extends the class util.Parser in the XMLValidation project and override the method:
- public **String** parse(**File** XMLFile, **File** SchemaFile)
You can either use Java code or execute a command line (java -jar, execute an executable file, etc.). To execute a command line, use the method:
String execCommand(**String** command)

Here are the inputs supported by the arguments:

- **XMLFile≠null and SchemaFile≠null** Validates the XMLFile against the SchemaFile.
- **XMLFile≠null and SchemaFile=null** Validates the XMLFile with the Schema given in the SchemaLocation tag. The SchemaFile is not used.
- **XMLFile=null and SchemaFile≠null** Validates the SchemaFile with the W3C standard [1].

2.3.2 Adding parser to config file

Add a new entry in the config file with the following information: (see figure 3 for an example)

- **name**
- **version**
- **ID** is used if you want to use different methods of validation with the same parser. (Ex: DOM and SAX)
- **instance_validation** is used to determine if the parser is able to validate an instance.
- **schema_validation** is used to determine if the parser is able to validate a schema.
- **needSchemaLocation** is used for the instance validation. Some parsers require that a SchemaLocation tag appears in the XML File. If the SchemaLocation tag is not present, an error message will be displayed in the result page.
- **depLocPath** is the path to the dependencies (jar files, executable, etc.)
- **class** is the parser's class (including the packages, ex: util.parser.XERCESParser)
- **comment** (optional)
- **url** (optional)

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <parsers>
        <parser name="Parser1" version="1.0" ID="1" instance_validation="true"
                schema_validation="true" needSchemaLocation="true">
            <depLocPath>path</depLocPath>
            <class>util.parser.Parser1</class>
            <comment>comment</comment>
            <url>http://www.parser1.com/</url>
        </parser>
        <parser name="Parser2" version="2.0.1" ID="1" instance_validation="false"
                schema_validation="true" needSchemaLocation="false">
            <depLocPath>path</depLocPath>
            <class>util.parser.Parser2</class>
        </parser>
    </parsers>
    <schemas>
        <schema name="Schemas" version="NA" ID="1">
            <path>path</path>
            <comment>comment</comment>
            <url>http://www.schemas-url.org</url>
        </schema>
    </schemas>
</configuration>
```

Figure 3: Example of a configuration file with the part related to the parser highlighted

3 The XML Validator library

This section provides instructions to assist a programmer in modifying the tool.

3.1 Intro

XMLValidator is a library used by the validation page to validate XML Files. This library contains all the parsers used in the Validation Page but this package can be used in any other application.

3.2 UML

Figure 4 is a class diagram of the XMLValidator library.

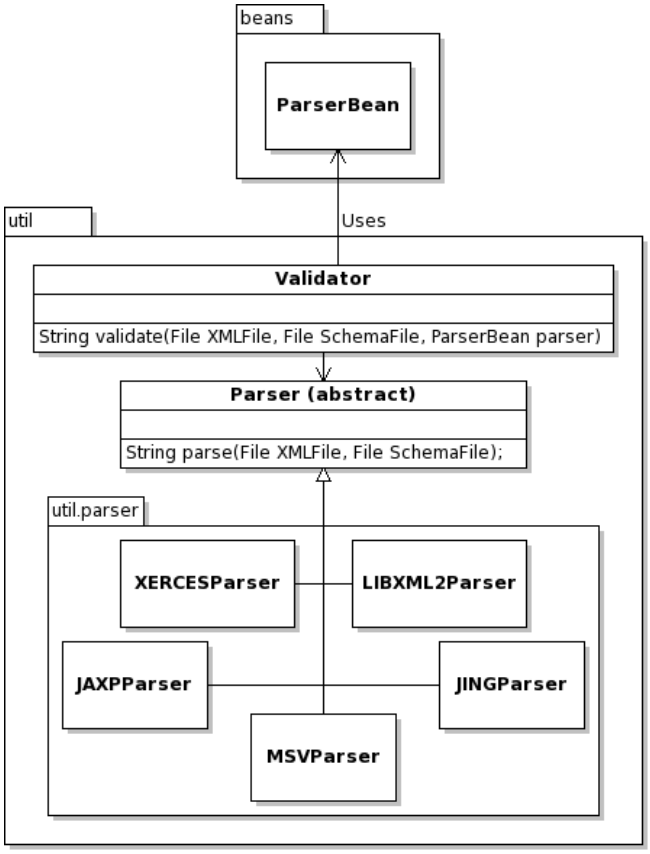

Figure 4: XMLValidator Class Diagram

3.3 Example

In this example (figure 5) we will validate an instance XML file (XMLFile) against a schema (SchemaFile). We will use the parser JING (class: util.parser.JINGParser), its dependencies are in the directory "dependencies/".

```
//Initialize the validator with the temporary directory
Validator v = new Validator("temp/");
//Create a parser
ParserBean parser = new ParserBean("dependencies/","util.parser.JINGParser");
//Run the validation and print the result in the default output.
System.out.println(v.validate(XMLFile, SchemaFile, parser));
```

Figure 5: Exmaple of the use of the XML Validator library

3.4 JavaDoc

3.4.1 Package Util

Class **Parser**

Abstract class used to represent a parser.

Declaration

public abstract class Parser
extends java.lang.Object

Constructors

- *Parser*
  `public` **Parser**`( )`

Methods

- *parse*
  `public abstract String` **parse**`( java.io.File` **XMLFile**`, java.io.File` **SchemaFile** `)`
- *setJarLocation*
  `public void` **setJarLocation**`( java.lang.String` **jarLocation** `)`

Class **Validator**

This class is used to validate an XMLFile against a schema. The parser used is given by the user.

Declaration

public class Validator
extends java.lang.Object

CONSTRUCTORS

- *Validator*
  `public` **Validator**`( java.lang.String` **tempDir** `)`

METHODS

- *validate*
  `public String` **validate**`( java.io.File` **XMLFile**`, java.io.File` **SchemaFile**`, beans.ParserBean` **parser** `)`
  - **Usage**
    - Validates the XML File against the SchemaFile.
  - **Parameters**
    - `XMLFile` - TXML File to validate
    - `SchemaFile` - Schema used for validation
    - `parser` - Parser used.
  - **Returns** - Output of the validation.

3.4.2 Package beans

CLASS ParserBean

Represents a parser used for the validation. It contains:

- Name
- Version
- Location of the directory that contains the dependencies (ex: .jar files)
- Name of class that extends from util.Parser
- Whether it's used for schema or instance validation, or both
- Comments about the parser
- If the parser is able to validate a Schema
- If the parser is able to validate an Instance
- If the parser needs the SchemaLocation tag in the XML file
- URL where you can find the parser

Declaration

public class ParserBean
extends java.lang.Object

Constructors

- *ParserBean*
  `public` **ParserBean**`( )`
- *ParserBean*
  `public` **ParserBean**`( java.lang.String` **depLocPath**`, java.lang.String` **className** `)`

Appendix A

This file is the XML Schema representing the XML file received by the user when he chooses to receive the results by email. (see figure 1)

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
    qualified">
  <xs:element name="ValidationResults" type="resultsType"></xs:element>
  <xs:complexType name="resultsType">
    <xs:sequence>
      <xs:element name="Results" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Parser">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Name" type="xs:string"/>
                  <xs:element name="Version" type="xs:string"/>
                  <xs:element name="Output" type="xs:string"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="Files">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="SchemaFile" type="xs:string"/>
                  <xs:element name="XMLFile" type="xs:string" minOccurs="0"
                      maxOccurs="1"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Figure 6: XML Schema for representing the validation results

Appendix B

This file is the XSLT document used to transform the validation results (see example XML file in figure 1 and XML Schema above) in html format.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <xsl:for-each select="ValidationResults/Results">
    <font color="#369" size="4"><strong>Parser: </strong></font>
    <xsl:value-of select="Parser/Name"/> v<xsl:value-of select="Parser/Version"/
       ><br/>
    <font color="#369" size="4"><strong>Schema File: </strong></font>
    <xsl:value-of select="Files/SchemaFile"/>
    <xsl:if test="Files/XMLFile">
      <br/>
      <font color="#369" size="4"><strong>XML File: </strong></font>
      <xsl:value-of select="Files/XMLFile"/>
    </xsl:if>
    <br/><br/>
    <font color="6633CC" size="5">Output: </font>
    <div style="margin-left: 40px;">
      <xsl:value-of select="Parser/Output" disable-output-escaping="yes"/>
    </div>
    <br/>
    <font color="009933" size="5">Processing Complete</font>
    <br/><br/>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Figure 7: XSLT

References

[1] World Wide Web Consortium. **XML Schema**. http://www.w3.org/XML/Schema.

[2] APACHE. **Xerces Java Parser**. http://xerces.apache.org/xerces-j.

[3] Thai Open Source Software Center Ltd. **JING: A RELAX NG validator in Java**. http://www.thaiopensource.com/relaxng/jing.html.

[4] SUN. **SUN Multi-Schema XML Validator**. https://msv.dev.java.net/.

[5] OASIS (Organization for the Advancement of Structured Information Standards). **OASIS Universal Business Language (UBL) v2.0**. http://docs.oasis-open.org/ubl/os-UBL-2.0/UBL-2.0.html.

[6] Open Application Group. **OAGIS 9.0 Documentation**. http://www.oagi.org/oagi/downloads/oagis/oagis9_doc.htm.

[7] Automotive Industry Action Group. https://www.aiag.org/.

[8] ANSI/AIIM. **Strategy Markup Language (StratML)**. http://www.stratml.net/.

[9] Department of Defense Modeling and Simulation Coordination Office. M&S community of interest (coi) discovery metadata specification (msc-dms). http://www.msco.mil/resource_discovery.html.

[10] SUN. **Java API for XML Processing**. https://jaxp.dev.java.net/.

[11] **The XML C parser and toolkit of Gnome**. http://xmlsoft.org/.

[12] World Wide Web Consortium. Extensible stylesheet language transformations (xslt) version 2.0. http://www.w3.org/TR/xslt20/.